

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kardinaal Mercierlaan 94, 3001 Leuven (Heverlee)

DYNAMICAL NEURAL NETWORKS AND COMPOSITION NETWORKS FOR SYSTEM MODELING

Promotor:
Prof. dr. ir. J. Vandewalle

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Yves MOREAU

Mei 1998



KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kardinaal Mercierlaan 94, 3001 Leuven (Heverlee)

DYNAMICAL NEURAL NETWORKS AND COMPOSITION NETWORKS FOR SYSTEM MODELING

Jury:
Prof. J. Delrue, voorzitter
Prof. J. Vandewalle, promotor
Prof. B. De Moor
Prof. A. Barbé
Prof. R. Sépulchre (Université de Liège)
Prof. D. Roose
Prof. L. Brenig (Université Libre de Bruxelles)

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Yves MOREAU

U.D.C. 681.3*I2

Mei 1998

© Katholieke Universiteit Leuven – Faculteit Toegepaste Wetenschappen
Arenbergkasteel, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag vermenigvuldigd of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/1998/7515/34

ISBN 90-5682-133-4

Foreword

Arriving at the end of my doctoral work, I want to thank many people.

Vooreerst bedank ik mijn promotor Prof. Joos Vandewalle. De steun en de vele opportuniteiten waarvan ik deze vier jaren heb genoten, hebben van mijn doctoraatswerk een buitengewone ervaring gemaakt. Joos heeft de gave onze SISTA groep rond zich te laten floreren door aan iedereen de juiste ontplooiingsmogelijkheden te geven en samenwerkingsgeest aan te moedigen. Ik besef hoe gelukkig ik ben om in zo'n vooraanstaande wetenschappelijke groep in zulke goede omstandigheden te mogen werken.

Ik dank ook Prof. Bart De Moor en Prof. André Barbé, die deel uitmaakten van mijn leescomité, voor het lezen van mijn proefschrift op korte termijn. De 'Let's do it!'-geest van Bart is één van de drijvende krachten van SISTA en zal mij altijd blijven inspireren om nog een uitdaging meer aan te gaan.

Je remercie le Prof. Rodolphe Sépulchre de l'Université de Liège pour le soin avec lequel il a lu ma thèse. Je le remercie pour l'intérêt qu'il porte à mon travail et pour son soutien. Notre collaboration a été des plus agréables et j'espère la voir se développer plus avant.

Je remercie le Prof. Léon Brenig de l'Université Libre de Bruxelles d'avoir accepté de faire partie du jury. Je le remercie (ainsi que Stéphane Louies) pour son accueil immédiatement chaleureux lors de nos premières rencontres. Notre récente collaboration s'est révèlée des plus fructueuses et j'attends avec impatience de pouvoir la poursuivre.

Ik dank ook Prof. Dirk Roose, die aanvaard heeft om deel uit te maken van de jury en Prof. Jan Delrue, als voorzitter van de jury.

During my research work, I benefited from opportunities to present my work to international colleagues and I want to thank those who helped create these opportunities.

I thank Dr. Miroslav Kárný, Prof. Kevin Warwick, and Dr. Jiří Kádleč for inviting me as a keynote speaker at the Second IEEE Workshop on Computer Intensive Methods in Control and Signal Processing.

I thank Dr. Joan Feigenbaum for inviting me to present our work on fraud detection in mobile communications at the DIMACS Workshop on Massive Datasets in Telecommunications.

Ik dank ook Dr. Ann Nowé van de Vrije Universiteit Brussel voor haar uitnodiging om ons werk over fraudedetectie op de KVIV Studiedag voor te stellen.

Je dois encore remercier Prof. Léon Brénig pour son invitation à faire une présentation lors du Workshop 1998 du projet Européen CATHODE.

I must also acknowledge the sources of funding that allowed me to attend important conferences. I thank Prof. Trevor Clarkson and Ms. Terhi Garner of NEuroNet for the studentships that allowed me to participate to the 1996 NEuroFuzzy Workshop and the 1997 International Conference on Artificial Neural Networks. I thank Prof. Bartlett Mel and Prof. Arun Jagota of the NIPS organization for the travel grant at NIPS 1997.

The ASPeCT project on the security of mobile communications has been an exciting endeavour. I thank our workpackage leaders, Prof. Bart Preneel and Prof. Shawe-Taylor, as well as the Indians: Dr. Peter Burge, Christof Störmann, and Herman Verrelst.

The adventure of this doctoral work began with my studies at the Division of Applied Mathematics of Brown University. This stay had a major influence both on my work and on my person. I thank the Commission for Educational Exchange, in particular Mrs. Margareth Nicholson, for the Fulbright grant they awarded me. I thank Prof. Paul Symonds for accepting me as his research assistant. Above all, I miss all my good friends from Brown.

Je remercie aussi les Prof. Henri Leich, Christian Wellekens et Hervé Bourlard pour m'avoir introduit aux réseaux de neurones et pour leur soutien dans mon projet d'études.

Alle leden van SISTA maken het hier een aangename plaats om te werken. Ik dank mijn bureaugenoten Jeroen, Peter en Lieven voor de ontspannen werksfeer. Ik dank Herman, Johan en Jeroen voor het steeds boeiender werk dat wij rond neurale netwerken doen. Bart DS had altijd een antwoord op mijn UNIX-problemen. Bart M 'keeps the ball rolling' met veel talent. Ik dank ook onze secretaresses Ingrid, Ida en Inge. Ik denk hierbij ook speciaal aan Rita.

Ik dank vooral ook het Fonds voor Wetenschappelijk Onderzoek voor de financiële steun.

Mais je voudrais surtout remercier tous mes amis (que je n'ai pas beaucoup pu voir durant la rédaction de cette dissertation) et ma famille. Mes parents m'ont toujours soutenu et cela a été pour moi une aide incommensurable.

Tot slot wil ik Sonia bedanken. Zij heeft steevast geloofd dat dit proefschrift tot stand zou komen : haar enthousiasme heeft mijn eigen twijfels overwonnen. Maar ik moet Sonia vooral bedanken voor het immense geluk in ons huis.

Abstract

We consider two categories of models of nonlinear systems in this dissertation: dynamical neural networks and composition networks. Dynamical neural networks are continuous-time systems consisting of a number of simple saturating units interacting with each other. They can serve as models of any smooth nonlinear system. Our first goal is to anchor dynamical neural networks deeper in the fields of system theory and control theory. Composition networks are a new class of models of nonlinear systems. We describe these composition networks as sequences of invertible transformations. Our second goal is to extend neural networks by introducing ideas from Lie algebra theory, which form the basis of our composition networks.

Our thesis goes as follows: (1) dynamical neural networks and composition networks are adequate models of nonlinear systems, (2) the study of changes of coordinates gives us a deeper understanding of dynamical neural networks, and (3) Lie algebra theory is the right tool for the analysis of compositions of transformations used for the modeling of dynamical systems.

This dissertation contains four original contributions. We give a representation of dynamical neural networks as Lotka–Volterra systems. The equivalence between these systems is a new approach to the analysis of neural networks. We present an equivalence theorem for dynamical neural networks with and without hidden layer. This theorem permits a more precise analysis of neural networks. We introduce Lie algebra theory into the field of neural networks. More specifically, we introduce methods using compositions to integrate ordinary differential equations. Finally, we present composition networks as a new approach to system modeling based on composition methods and demonstrate the effectiveness of such networks.

Korte Inhoud

We beschouwen twee categorieën van modellen van niet-lineaire systemen in dit proefschrift : dynamische neurale netwerken en compositienetwerken. Dynamische neurale netwerken zijn tijdscontinue systemen die uit een aantal eenvoudige verzadigingseenheden bestaan, die met elkaar interageren. Deze netwerken kunnen dienen als modellen van willekeurige niet-lineaire systemen. Ons eerste doel is om dynamische neurale netwerken verder te verankeren in de systeemtheorie en de regeltechniek. Compositienetwerken vormen een nieuwe klasse modellen van niet-lineaire systemen. We beschrijven deze compositienetwerken als reeksen inverteerbare transformaties. Ons tweede doel is om neurale netwerken uit te breiden door ideeën uit de Lie-algebratheorie in te voeren. Deze ideeën vormen de basis voor onze compositienetwerken.

Onze thesis luidt als volgt : (1) dynamische neurale netwerken en compositienetwerken zijn doeltreffende modellen voor niet-lineaire systemen, (2) de studie van coördinatentransformaties geeft ons een dieper inzicht in dynamische neurale netwerken, en (3) Lie-algebratheorie geeft ons de adequate werktuigen voor de analyse van de composities van transformaties, die wij gebruiken om dynamische systemen te modelleren.

Dit proefschrift levert vier originele bijdragen. We geven een voorstelling van dynamische neurale netwerken als Lotka–Volterra systemen. De equivalentie tussen deze twee types systemen levert een nieuwe aanpak voor de analyse van neurale netwerken. We introduceren een equivalentiestelling voor neurale netwerken met en zonder verborgen laag. Deze stelling laat een meer nauwkeurige analyse van neurale netwerken toe. We voeren de Lie-algebratheorie in op het gebied van neurale netwerken. Meer bepaald voeren we methoden, die composities gebruiken, in voor de integratie van gewone differentiaalvergelijkingen. Tot slot stellen we compositienetwerken voor als een nieuwe, op compositiemethoden gebaseerde aanpak voor het modelleren van systemen en we tonen hun doeltreffenheid aan.

Notation*

List of symbols

$N(x)$	the function realized by a neural network
$\sigma(\xi), \rho(\xi)$	the activation function of a neural network
A^T	transpose of the matrix A
A^{-1}	inverse of the matrix A
$A^\dagger$	Moore–Penrose pseudoinverse of the matrix A
$O^{n\times m}$	$(n \times m)$-dimensional matrix of zeros
$I^{n\times m}$	$(n \times m)$-dimensional matrix with ones on the diagonal and zeros elsewhere
$[v; w]$	stackup of the vectors v and w to form a matrix
$\left[\begin{array}{c} A; \\ \end{array}\right.$ B $\left.\begin{array}{c} \\ \end{array}\right]$	stackup of the matrices A and B
∇f	gradient of a real-valued function or Jacobian of a vector-valued function
$L_g f$	Lie derivative of the function f with respect to the vector field g
$[f, g]$	Lie bracket of two vector fields f and g
$\mathrm{ad}_f^k g$	iterated Lie bracket of g with f
$\lvert \cdot \rvert$	absolute value of a real number
$\lVert \cdot \rVert_p$	L_p norm of a function $(1 < p < \infty)$
$\lVert \cdot \rVert_\infty$	L_∞ norm of a function
$\Re\mathrm{e}(z), \Im\mathrm{m}(z)$	real and imaginary parts of the complex number z
$\delta(\xi)$	Dirac's distribution
$\mathrm{degree}(P)$	degree of the polynomial P
$\mathrm{diag}(v)$	diagonal matrix having the elements of v on its diagonal
$\mathrm{rank}(A)$	rank of the matrix A
$\mathrm{span}(\mathcal{S})$	span of the set of vectors $\mathcal{S}$
$\dim(\mathcal{G})$	dimension of the distribution $\mathcal{G}$
$\mathrm{degree}(P)$	degree of the polynomial P
$\mathrm{order}(L)$	order of the formal Lie monomial L
$\mathbb{R}^n$	set of n-dimensional real vectors
$\mathbb{R}^{n\times m}$	set of real matrices of dimension $n \times m$
$\mathbb{C}$	set of complex numbers

*See also the remarks about style and notation in Section 1.4

L_p	set of functions of finite L_p norm ($1 < p < \infty$)
L_∞	set of functions of finite L_∞ norm
$C(\mathcal{K})$	set of real-valued continuous functions defined on $\mathcal{K}$
C^∞	set of infinitely differentiable functions from $mathbbR^d$ to $\mathcal{R}$
C_c^∞	set of infinitely differentiable functions with compact support
$\mathcal{N}_\sigma^d$	set of neural networks with d inputs and activation function σ
$\mathcal{P}$	set of permutation matrices
$\mathcal{S}$	set of diagonal matrices whose diagonal elements are 1 or -1
$\mathbb{B}^{n\times m}$	set of $(n \times m)$-dimensional irreducible matrices
$\mathcal{R}(x_0)$	reachability set of a control system at the point x_0

Acronyms

BCH	Baker–Campbell–Hausdorff
BPTT	Backpropagation through time
CNN	Cellular neural network
MIMO	Multiple input multiple output
MLP	Multilayer perceptron
NARMAX	Nonlinear autoregressive moving average with external input
NARX	Nonlinear autoregressive with external input
ODE	Ordinary differential equation
SISO	Single input single output
SSE	Sum of squared errors
SVD	Singular value decomposition
RTRL	Real-time recurrent learning
VLSI	Very–large-scale integration

ACTS	Advanced communication technologies and services
ASPeCT	Advanced security for personal communication technologies
GNU	GNU's not UNIX
GSM	Global system for mobile communications
IMSI	International mobile subscriber identity
PABX	Private automatic branch exchange
PDAT	Protocol data analysis tool
ROC	Receiver operating characteristic
UMTS	Universal mobile telephone system

Contents

Foreword i

Abstract iii

Notation v

Contents vii

Nederlandse samenvatting xi

1 Introduction 1
- 1.1 What neural networks are . . . 1
- 1.2 Structure of this dissertation . . . 3
- 1.3 Chapter-by-chapter overview . . . 5
- 1.4 A few remarks about style and notation . . . 14

2 Neural networks for function approximation 17
- 2.1 Early results . . . 17
- 2.2 Weierstrass's theorem . . . 21
- 2.3 Necessary and sufficient conditions for universal approximation 23
- 2.4 Rates of convergence . . . 29
- 2.5 Optimal nonlinear approximation . . . 31
- 2.6 The curse of dimensionality . . . 32
- 2.7 Uniqueness of the weights of static neural networks . . . 35
  - 2.7.1 The independence property . . . 36
  - 2.7.2 Irreducibility and sign–permutation equivalence . . . 38
- 2.8 Conclusion . . . 40

3 Neural networks as dynamical systems 43
- 3.1 Dynamical system theory . . . 43
- 3.2 Convergence and stability . . . 46
- 3.3 The Cohen–Grossberg theorem . . . 49

3.4 Cascades 54
3.5 Global asymptotic stability 57
3.6 Lur'e–Postnikov Lyapunov functions 59
3.7 Conclusion 65

4 Representation of neural networks as Lotka–Volterra systems 67
4.1 Lotka–Volterra systems 67
4.2 Dynamical perceptrons 68
4.2.1 Sigmoidal change of coordinates 69
4.2.2 Transformation into quasi-monomial forms 70
4.2.3 Matrices of the quasi-monomial forms 72
4.3 Transformation into Lotka–Volterra systems 75
4.4 Dynamical perceptrons with self-connections 77
4.4.1 Setup of the quasi-monomial forms 78
4.4.2 Matrices for perceptrons with self-connections 78
4.4.3 Singularities of quasi-monomial transformations 80
4.5 Dynamical multilayer perceptrons 82
4.6 Choice of the activation function 84
4.7 Conclusion 85

5 Control theory of neural networks 87
5.1 Learning in continuous-time neural networks 88
5.2 Observability 92
5.3 Controllability 95
5.4 Uniqueness of the weights of dynamical neural networks 99
5.5 Discrete-time systems 103
5.5.1 Input–output models 104
5.5.2 State-space models 108
5.6 Conclusion 111

6 Approximation capabilities of dynamical neural networks 113
6.1 Immersibility 114
6.2 Changes of coordinates between systems of different dimensions 116
6.2.1 Linear changes of coordinates 118
6.2.2 Rank $B = d$ 119
6.2.3 Rank $B \leq d$ 121
6.2.4 Systems with a nonlinear output map 124
6.3 Equivalence between networks with and without hidden layer . 125
6.4 Extension to networks with self-connections 126
6.4.1 Networks with self-connections where rank $B = d$ 126
6.4.2 Networks with self-connections where rank $B < d$ 127
6.5 Dynamical system approximation with neural networks 128
6.5.1 Approximation of a dynamical system 128

6.5.2 Hopfield-type networks . . . 130
6.5.3 Dynamical perceptrons . . . 131
6.5.4 Other dynamical neural networks without hidden layer . 132
6.6 Conclusion . . . 133

7 Lie algebra theory of nonlinear control systems 135
7.1 Differential-geometric control . . . 135
7.2 Frobenius's theorem . . . 139
7.3 Controllability and observability . . . 142
7.3.1 Rank conditions . . . 142
7.3.2 Application to neural networks . . . 144
7.4 Inverses of recurrent neural networks . . . 145
7.4.1 Relative degree and inverses of nonlinear systems . . . 146
7.4.2 Application to Hopfield-type networks . . . 148
7.5 Canonical form of SISO systems . . . 151
7.6 The zero dynamics . . . 155
7.6.1 Computation of the zero dynamics . . . 155
7.6.2 Canonical dynamical neural networks . . . 157
7.7 Feedback linearization . . . 159
7.8 Conclusion . . . 162

8 Integration of ODEs using composition methods 163
8.1 Exponential maps and the Baker–Campbell–Hausdorff formula 163
8.2 Composition methods . . . 165
8.3 Control of the ball and beam . . . 168
8.4 Formulas for composition integrators . . . 171
8.5 Convergence of decompositions of exponential operators . . . 173
8.6 Simulation of arrays of Chua's circuits . . . 174
8.6.1 Splitting . . . 174
8.6.2 Integration of one-dimensional ODEs . . . 176
8.6.3 Numerical results . . . 177
8.7 Conclusion . . . 179

9 Composition networks for system modeling 183
9.1 Compositions in system modeling . . . 183
9.2 Prediction of dynamical systems . . . 185
9.2.1 Computation of the gradients . . . 186
9.2.2 Prediction of the Lorenz attractor . . . 188
9.2.3 Irregularly sampled systems . . . 191
9.3 System identification with recurrent composition networks . . . 192
9.3.1 Identification of the Lorenz attractor . . . 193
9.3.2 Identification of control-affine systems . . . 193
9.4 MLPs in dynamics space . . . 196

9.4.1 One-dimensional sigmoidal ODEs 197
9.4.2 Multidimensional sigmoidal ODEs 198
9.4.3 Universal approximation by compositions 199
9.5 Compositions and the problem of motion-planning 200
9.5.1 Nilpotent Lie algebras and Hall bases 201
9.5.2 Hall coordinates and compositions of exponential maps 202
9.5.3 Links with approximation by compositions 206
9.6 Conclusion 207

10 General conclusions **209**
10.1 Contributions 209
10.2 Suggestions for further research 211

A Detection of mobile phone fraud using neural networks **213**
A.1 Fraud scenarios and fraud indicators 214
A.2 Absolute and differential analyses 217
A.3 Architecture of the fraud detection tools 218
A.4 User monitoring 220
A.5 Description of the user profiles 221
A.6 Supervised neural network for fraud detection 223
A.7 Evaluation of the supervised neural network 224
A.8 Rule-based system 225
A.9 Unsupervised neural network 227
A.10 Conclusion 229

Bibliography **231**

Index **245**

Dynamische neurale netwerken en compositienetwerken voor systeemmodelleren

Nederlandse samenvatting

In dit proefschrift bestuderen we neurale netwerken en compositienetwerken voor het modelleren van niet-lineaire systemen. We maken gebruik van de technieken van dynamische systeemtheorie, regeltechniek, en Lie-algebra theorie om deze systemen te analyseren en om nieuwe toepassingen te ontwikkelen.

Algemeen overzicht

Het idee dat tot dit proefschrift leidde is dat inverteerbare transformaties een belangrijke rol spelen in de studie van tijdscontinue systemen (door het uitdrukken van hun oplossingen als eenparametergroepen, door coördinatentransformaties, door de wederopbouw van hun toestand uit hun uitgangssequentie, enzovoort). Inverteerbare transformaties zijn tot nu toe weinig bestudeerd, wat ruimte laat voor originele ontwikkelingen. We bespreken dit idee niet rechtstreeks in dit werk omdat het te algemeen is. Maar wij kunnen welopmerken dat de originele bijdragen in Hoofdstukken 4 en 6 gebruik maken van coördinatentransformaties en dat Hoofdstukken 8 en 9 het gebruik bespreken van composities van inverteerbare transformaties voor de benadering van dynamische systemen.

Onze thesis luidt als volgt : (1) dynamische neurale netwerken en compositienetwerken zijn doeltreffende modellen voor niet-lineaire systemen, (2) de studie van coördinatentransformaties geeft ons een dieper begrip van dynamische neurale netwerken, en (3) Lie-algebratheorie geeft ons de adequate

werktuigen voor de analyse van de composities van transformaties die wij gebruiken om dynamische systemen te modelleren. In dit proefschrift, streven we naar het leggen van verbanden tussen onderwerpen uit verschillende vakgebieden om een breed en gestructureerd beeld van neurale netwerken te geven.

Om onze thesis te verdedigen en in zijn juiste context te plaatsen, organizeren wij ons werk als volgt. In Hoofdstuk 2, bestuderen we de eigenschappen van statische neurale netwerken voor de benadering van functies. Deze eigenschappen garanderen dat, hoewel neurale netwerken een eenvoudige structuur hebben, hun gedrag algemeen kan zijn. In Hoofdstuk 3 voeren we dynamische neurale netwerken in. Deze netwerken zijn dynamische systemen met afleidbare niet-lineariteiten. We bestuderen hun convergentie en stabiliteit door gebruik te maken van de werktuigen van de dynamische systeemtheorie. In Hoofdstuk 4 bespreken wij een nieuwe voorstelling van dynamische neurale netwerken als Lotka–Volterra-systemen. Lotka–Volterra-systemen zijn klassieke kwadratische modellen uit de theoretische biologie en dynamische systeemtheorie. In Hoofdstuk 5 voegen we ingangen toe aan onze dynamische neurale netwerken en bekomen zo regelsystemen. We bespreken resultaten uit de regeltheorie die toepasbaar zijn op neurale netwerken. In Hoofdstuk 6 bestuderen we de benaderingseigenschappen van dynamische neurale netwerken met ingangen door de ontwikkeling van een algemene lineaire coördinatentransformatie voor neurale netwerken. In Hoofdstuk 7 beschouwen we deze systemen in het kader van differentiaalmeetkundige controle en voeren we voor de eerste keer in dit proefschrift Lie-algebratheorie in. In Hoofdstuk 9 ontwikkelen we deze toepassing van Lie-theorie verder en stellen we compositienetwerken voor om dynamische systemen te voorspellen. Ten slotte stellen we in een appendix apart werk voor over de detectie van fraude in mobiele telecommunicatie met behulp van neurale netwerken.

De volgorde van onze bespreking is natuurlijk. We evolueren van statische netwerken naar dynamische netwerken, en dan naar regelsystemen. Daarna breiden we onze bespreking uit naar differentiaalmeetkundige methoden en Lie-algebratheorie. Een andere belangrijke trend is de overgang van theoretische onderwerpen (benaderingstheorie en dynamische systeemtheorie) naar toegepaste onderwerpen (regeltechniek, voorspelling van dynamische systemen en fraudedetectie).

Overzicht per hoofdstuk

Het proefschrift telt twee belangrijke delen. In het eerste deel, bestuderen we de eigenschappen van verschillende soorten netwerken van het perceptrontype (statisch, dynamisch en dynamisch met ingangen). In het tweede deel, voeren we het compositienetwerk in als nieuwe methode voor het modelleren van

dynamische systemen. Hoofdstukken 2, 3, 5 en 7 geven een overzicht van de bestaande literatuur terwijl Hoofdstukken 4, 6, 8, 9 en de appendix onze originele bijdragen bevatten. We schetsen de inhoud van ieder hoofdstuk hieronder.

Hoofdstuk 2. Neurale netwerken voor functiebenadering

We bestuderen neurale netwerken voor de benadering van functies van reële variabelen. Meer bepaald, geven we een overzicht van resultaten over universele benaderingseigenschappen en convergentiegraden van voorwaartse perceptrons met één verborgen laag.

Dit soort netwerk berekent een vectorfunctie van de ingang in drie stappen. Ten eerste, berekent het netwerk een affiene transformatie van de ingangsvector naar een ruimte van verschillende dimensie. We noemen het resultaat de activeringsvector. Ten tweede, berekent het netwerk een gladde niet-lineaire transformatie van iedere component van de activeringsvector om de verborgen vector te bekomen. De niet-lineaire transformatie, die component per component wordt uitgevoerd, is vaak een verzadigingsfunctie die we een sigmoïde noemen. Deze verzadigingsfunctie is continu, monotoon en heeft eindige limieten naar oneindig. Ten derde, berekent het netwerk een lineaire transformatie van de verborgen vector om de uitgangsvector te produceren. Deze uitgangsvector kan een dimensie hebben die verschillend is van de dimensie van de ingangsvector en van de verborgen vector. We kunnen dit type berekening als een netwerkdiagram voorstellen. We noemen de componenten van de verschillende vectoren eenheden of neuronen. We kunnen de waarde van de ingangsneuronen en uitgangsneuronen direct meten terwijl de anderen neuronen verborgen blijven.

De sigmoïdale niet-lineariteiten spelen de belangrijkste rol in de bepaling van de niet-lineaire eigenschappen van deze netwerken. Hoewel er andere types neurale netwerken bestaan die geen sigmoïdale eenheden gebruiken, is de fundering van neurale netwerken het idee van een rekenmechanisme dat opgebouwd wordt als een netwerk van niet-lineaire eenheden, die met elkaar interageren. De sigmoïdale eenheid zal meermaals terugkeren in dit proefschrift, ook al hebben de neurale netwerken die hen gebruiken verschillende beschrijvingen en toepassingen.

Het doel van Hoofdstuk 2 is eerst te bepalen welk soort functies van reële variabelen neurale netwerken kunnen benaderen. Er wordt aangetoond dat perceptrons met één verborgen laag brede klassen van functies kunnen benaderen. We zeggen dat neurale netwerken de universele benaderingseigenschap hebben. We beginnen met de klassieke resultaten van Cybenko [28] en Hornik, Stinchcombe en White [62]. Daarna, bestuderen we een meer recent resultaat van Leshno [75] dat stelt dat een perceptron met één verborgen laag functies universeel kan benaderen als en slechts als de activeringsfunctie geen veelterm is. Een belangrijk punt in de uiteenzetting is dat we dit resultaat kunnen

bewijzen door gebruik te maken van een klassiek resultaat uit de wiskundige analyse : de stelling van Weierstrass. Deze stelling toont aan dat veeltermen continue functies universeel kunnen benaderen. Deze aanpak is leerrijk omdat hij aantoont dat universele benadering een vrij natuurlijke eigenschap is.

De studie van universele benaderingseigenchappen geven neurale netwerken geen duidelijk voordeel ten opzichte van andere benaderingsmethoden. Maar er werd herhaaldelijk in de praktijk aangetoond dat neurale netwerken beter bruikbaar zijn dan veeltermreeksen voor hoogdimensionale problemen. We beschouwen dan de mogelijkheid dat neurale netwerken compactere voorstellingen kunnen leveren dan veeltermen. We geven een overzicht van het werk van Barron [13] dat de convergentiegraden van neurale netwerken bestudeert. De convergentiegraad karakteriseert hoe de benaderingsfout van een netwerk daalt met het aantal verborgen neuronen. Er wordt aangetoond dat, voor bepaalde interessante klassen van functies, perceptrons met één verborgen laag compactere voorstellingen leveren dan veeltermen. Wij bespreken daarna de noodzakelijke nuances voor de interpretatie van deze resultaten.

Ten slotte beschouwen we het probleem of twee verschillende neurale netwerken dezelfde functie kunnen voorstellen omdat, als dit het geval is, we bijzonder voorzichtig moeten zijn bij de analyse van neurale netwerken. De studie van neurale netwerken toont aan dat twee verschillende netwerken dezelfde functie kunnen voorstellen maar dat twee zulke netwerken altijd door een eenvoudige symmetrie verbonden zijn. Dit eenvoudige verband zorgt ervoor dat de problemen veroorzaakt door deze redundantie beperkt blijven.

Hoofdstuk 3. Neurale netwerken als dynamische systemen

We geven hier een overzicht van de eigenschappen van convergentie en stabiliteit van dynamische neurale netwerken. Terwijl de statische neurale netwerken uit Hoofdstuk 2 een niet-lineaire transformatie van de ingang als uitgang gaven, evolueren dynamische neurale netwerken met de tijd en beschrijven we deze netwerken met behulp van gewone differentiaalvergelijkingen. We bestuderen verschillende types neurale netwerken vanuit het standpunt van dynamische systeemtheorie. De sigmoïdale activeringsfunctie speelt nog steeds een centrale rol aangezien de gewone differentiaalvergelijkingen, die de neurale netwerken beschrijven, sigmoïdale transformaties bevatten.

We bespreken de elementaire definities van convergentie van de banen van een systeem naar een evenwichtspunt en van de stabiliteit van een evenwichtspunt. We voeren dan de werktuigen in die we gebruiken om deze eigenschappen te karakteriseren : de stelling van Lyapunov en het principe van onveranderlijkheid van LaSalle.

Ten eerste, beschouwen we voorwaarden die de convergentie van een dynamisch neuraal netwerk verzekeren. Een algemeen resultaat dat we bestuderen

is de stelling van Cohen en Grossberg [26]. Deze stelling geeft voldoende voorwaarden voor de convergentie van een klasse van concurrerende systemen. Deze klasse is een veralgemening van neurale netwerkmodellen. De stelling is toepasbaar op een breed gamma van systemen en het bewijs steunt op de constructie van een Lyapunov functie voor de systemen die we bestuderen. Maar de veronderstelling dat de interactiematrix symmetrich is en dat de elementen positief zijn, beperkt de practische toepasbaarheid van deze stelling. De volgende resultaten die we beschouwen zijn de resultaten van Hirsch [58] over de convergentie van stromen van neurale netwerken. Deze systemen worden gebouwd als een reeks subnetwerken die hun eigen dynamica hebben en waarvan de uitgang de volgende subnetwerken in de stroom aandrijft. De resultaten over de convergentie van stromen van netwerken steunen ook op de constructie van Lyapunov functies voor deze stromen.

Ten tweede, beschouwen we voorwaarden die de globale asymptotische stabiliteit van een neuraal netwerk verzekeren. We volgen de argumenten van Forti en Tesi [38] en bestuderen Lur'e–Postnikov Lyapunov functies voor neurale netwerken. We gebruiken hier opnieuw Lyapunov functies om stabiliteit te bewijzen, maar we beschouwen een 'reverse-engineering' aanpak. Inderdaad, we bepalen hier de algemene vorm van de Lyapunov functie a priori en proberen na te gaan of er parameterwaarden bestaan waarvoor de functie aan de voorwaarden van de stelling van Lyapunov voldoet.

Hoofdstuk 4. Voorstelling van neurale netwerken als Lotka-Volterra-systemen

We bewijzen hier dat we verschillende types teruggekoppelde neurale netwerken als Lotka–Volterra-systemen kunnen voorstellen. Dit resultaat is een originele bijdrage van dit proefschrift en we hebben het in [88] voorgesteld. Lotka–Volterra-systemen zijn klassieke modellen uit de theoretische biologie. Hun voornaamste eigenschap is hun vereenvoudigde kwadratische vorm, die hun analyse eenvoudiger maakt dan de analyse van dynamische neurale netwerken.

We beginnen met het eenvoudige geval van dynamische perceptrons. Hun eenvoudige structuur maakt dat hun gedrag beperkt is, maar de werkwijze voor deze systemen geeft een duidelijk zicht op de algemene procedure. De eerste stap is de transformatie van het dynamische perceptron in wat we een quasi-monomiale vorm noemen. Deze vorm is een systeem van gewone differentiaalvergelijkingen waar de dynamica van iedere variabele gelijk is aan het produkt van deze variabele met een lineaire combinatie van produkten van alle variabelen. De tweede stap bestaat erin de quasi-monomiale vorm in een Lotka–Volterra-systeem te transformeren. We gebruiken hiervoor de procedure ontwikkeld door Brenig [21]. We voeren de eerste stap uit door een eenvoudige verandering van variabelen. We gebruiken de sigmoïdale transformatie van de oude variabelen als nieuwe variabelen. Omwille van bijzondere eigenschappen

van de afgeleide van de twee meest voorkomende activeringsfuncties (tangens hyperbolicus en de logistische sigmoïde), bekomen we met deze verandering van variabelen direct een quasi-monomiale vorm. In deze quasi-monomiale vorm, bepalen produkten van de variabelen de gewone differentiaalvergelijking van het systeem. We voeren de tweede stap uit door deze produkten van variabelen als nieuwe variabelen van het systeem te beschouwen. Door deze uitdrukkingen te vereenvoudingen, neemt het systeem een Lotka–Volterra-vorm aan. Deze transformatie betekent dat we, voor ieder dynamisch perceptron, een Lotka–Volterra-systeem van hogere dimensie kunnen geven waarvoor het gedrag van de eerste coördinaten (evenveel als de dimensie van het dynamische perceptron) equivalent is met het gedrag van het neurale netwerk.

Nadat we deze transformatie voor dynamische perceptrons hebben uitgevoerd, kunnen we die ook uitvoeren voor dynamische perceptrons met zelfverbindingen. (Deze systemen produceren gedragingen die meer algemeen zijn dan die van de dynamische perceptrons.) Niet alleen tonen wij aan dat equivalente Lotka–Volterra-systemen bestaan, maar we geven ook de precieze parameters van deze equivalente systemen. We veralgemenen de transformatie ook verder tot het geval van dynamische meerlaagse perceptrons.

Hoofdstuk 5. Regeltheorie van neurale netwerken

In dit hoofdstuk, beschouwen we dynamische neurale netwerken opnieuw, maar deze keer voegen we er uitwendige ingangen aan toe. Daardoor verlaten we het gebied van dynamische systeemtheorie om het gebied van regeltheorie binnen te treden. Het eerste onderwerp in dit hoofdstuk is een probleem van praktisch belang : We geven een overzicht van de technieken voor de aanpassing van de parameters van een Hopfield netwerk om een gewenst ingangs-uitgangsgedrag te produceren. We stellen een algoritme voor dat deze taak uitvoert: het 'backpropagation-through-time'-algoritme. We leiden dit algoritme af met behulp van methoden uit de optimale regeltheorie [32]. Dit probleem is een eenvoudige vorm van systeemidentificatie en het is dus van belang vanuit het standpunt van de regeltheorie. De begrippen van observeerbaarheid en controleerbaarheid zijn nog belangrijker voor de regeltheorie. Het werk van Albertini en Sontag [7] antwoord op de vraag naar voorwaarden voor observeerbaarheid van neurale netwerken, aangezien het noodzakelijke en voldoende voorwaarden voor observeerbaarheid geeft. Het werk van Sontag en Sussman [133] beschouwt het probleem van controleerbaarheid van neurale netwerken. Een van hun voornaamste besluiten is dat de keuze van de activeringsfunctie van belang is voor de controleerbaarheid van het systeem. Ze geven het voorbeeld van een netwerk waarvoor het vervangen van de activeringsfunctie door een gelijkaardige activeringsfunctie het verlies van controleerbaarheid veroorzaakt. Zoals in het geval van statische neurale netwerken rijst de vraag of twee verschillende dynamische neurale netwerken hetzelfde ingangs-uitgangsgedrag kunnen hebben.

De voorwaarden, die Albertini en Sontag [6] voor ingangs-uitgangsequivalentie geven, lijken bijzonder sterk op de voorwaarden voor statische netwerken. Deze drie resultaten (observeerbaarheid, controleerbaarheid en equivalentie) leiden tot twee belangrijke begrippen : (1) het begrip van een activeringsfunctie met de onafhankelijkheidseigenschap (verzamelingen van translaties van de activeringsfunctie blijven altijd onafhankelijk), en (2) het begrip van een irreduceerbare matrix (een matrix waarvan twee rijen nooit gelijk zijn op een verandering van teken na). De irreduceerbaarheid van de matrix speelt een rol die gelijkaardig is aan de rol van een matrix met maximale rang in het geval van lineaire systemen.

De toepassing van neurale netwerken op praktische regelproblemen vergt het gebruikt van tijdsdiscrete modellen omdat de digitale regeltechniek de praktische toepassingen overheerst. We geven daarom een overzicht van verschillende regelschema's die tijsdiscrete neurale netwerken gebruiken. We beschouwen zowel ingangs-uitgangsmodellen als toestandsruimtemodellen, zoals voorgesteld door Narendra en zijn studenten [108, 109, 76, 77]. Deze tijdsdiscrete aanpak leidt tot 'inverse control', 'internal model control' en 'model-reference adaptive control' in het geval van ingangs-uitgangsmodellen. Deze aanpak leidt tot 'observer and controller design' in het geval van toestandsruimtemodellen.

Hoofdstuk 6. Benaderingsvermogen van dynamische neurale netwerken

We bestuderen hier de benadering van dynamische systemen door dynamische neurale netwerken. Enerzijds, zijn de resultaten over het benaderingsvermogen van statische neurale netwerken direct toepasbaar op dynamische netwerken met één verborgen laag, omwille van de benadering van het vectorveld van het dynamische systeem. Anderzijds, is het niet duidelijk of dynamische netwerken zonder verborgen laag willekeurige dynamische systemen kunnen benaderen. We tonen in dit hoofdstuk aan dat, als dynamische netwerken zonder verborgen laag een lineaire uitgangstransformatie hebben, ze willekeurige dynamische systemen kunnen benaderen omdat ze equivalent zijn met dynamische netwerken met een verborgen laag. De precieze bepaling van deze equivalentie en de uitdrukking ervan als een algemene stelling, die toepasbaar is op een breed gamma van neurale netwerken, is een originele bijdrage van dit werk, zie [96, 101].

Deze equivalenties tussen dynamische neurale netwerken met en zonder verborgen laag steunt op lineaire coördinatentransformaties tussen dynamische systemen van verschillende dimensie. Het centraal begrip is hier de notie van de 'immersibility' van een systeem in een systeem van hogere dimensie. Dit begrip houdt in dat, voor bepaalde initiële toestanden, het ingangs-uitgangsgedrag van het systeem van hogere dimensie gelijk is aan het gedrag van het systeem van lagere dimensie. Het voornaamste punt in de uiteenzetting is het bewijs van een

stelling over de equivalentie van veralgemeende neurale netwerken met en zonder verborgen laag. In deze stelling veronderstellen we niet dat de niet-lineaire transformatie component per component werkt. De transformatie kan om het even welke afleidbare transformatie zijn. Het bewijs is een veralgemening van het geval waar de matrix, die de verborgen dynamica bepaalt, inverteerbaar is en waar er een triviale coördinatentransformatie bestaat die het resultaat levert. In het algemene geval van een niet-vierkante matrix, moeten we het systeem analyseren met behulp van de singuliere waardenontbinding van de matrix van de verborgen dynamica. Eens we de algemene stelling hebben afgeleid, kunnen we die op neurale netwerken toepassen. We kunnen ook, onder aanvullende voorwaarden, de stelling veralgemenen tot neurale netwerken met zelf-verbindingen.

We kunnen deze equivalentie dan gebruiken om aan te tonen dat dynamische perceptrons willekeurige systemen kunnen benaderen. We volgen hier het werk van Sontag [131]. We geven ook een overzicht van een gelijkaardig resultaat van Funahashi en Nakamura [42] over het universele benaderingsvermogen van Hopfieldnetwerken. Dit resultaat gebruikt een coördinatentransformatie die duaal is met de onze, om de equivalentie tussen Hopfieldnetwerken en dynamische perceptrons met één verborgen laag vast te stellen.

Hoofdstuk 7. Lie algebratheorie van niet-lineaire regelsystemen

In dit hoofdstuk, voegen we de methoden van differentiaalmeetkundige controle toe aan de regeltheorie. Veel van de ontwikkelingen zijn beperkt tot regelsystemen die affien in de ingangen zijn. We voeren de begrippen Lie-afgeleide en Lie-haak in. De Lie-afgeleide meet de evolutie van een reële functie langs een baan van een regelsysteem. De Lie-haak is een veralgemening voor vectorvelden van het vectorprodukt. Deze objecten zijn nuttig voor de studie van de meetkunde van regelsystemen omdat ze vergelijkingen toelaten tussen banen, vectorvelden, en reële functies. Het doel van dit hoofdstuk is een overzicht te geven van de toepassing van de methoden van differentiaalmeetkundige controle op neurale netwerken.

We geven eerst een overzicht van de stelling van Frobenius. Deze stelling gebruikt Lie-haken om te controleren of het gedrag van een regelsysteem beperkt is tot een gegeven oppervlakte dan wel of het vrij is om ieder punt in de ruimte te bereiken. De stelling houdt een sterk verband met de notie van controleerbaarheid. Een gevolg van de stelling is de 'controllability rank condition' van Herman en Krener [56]. Deze voorwaarde verzekert een bepaalde vorm van controleerbaarheid door de rang te berekenen van een verzameling Lie-haken van de vectorvelden die het regelsysteem bepalen. Voor observeerbaarheid, verzekert de 'observability rank condition' (die duaal is met de 'controllability rank condition') een bepaalde vorm van observeerbaarheid.

Wij volgen dan het werk van Żbikowski [151] en passen de 'controllability rank condition' toe op neurale netwerken. Rekenmoeilijkheden maken het onmogelijk om de 'controllability rank condition' en 'observability rank condition' na te gaan voor neurale netwerken. Differentiaalmeetkundige methoden kunnen, spijtig genoeg, geen resultaten leveren die vergelijkbaar zijn met de resultaten van Hoofdstuk 5. We bestuderen dan de inversen van dynamische neurale netwerken. Voor deze analyse, moeten we het begrip relatieve graad van een affien regelsysteem invoeren. Deze relatieve graad is gelijk aan het aantal keer dat we de uitgang van het systeem moeten afleiden om de ingang expliciet te zien verschijnen. Een stelling van Hirschorn [60] levert dan een manier om de inverse van een affien regelsysteem te berekenen met behulp van Lie-afgeleiden. De toepassing van deze stelling op Hopfieldnetwerken, zoals voorgesteld door Khambapati [67], geeft voldoende voorwaarden die de parameters van het systeem bepalen waarvoor we het inverse systeem expliciet kunnen berekenen. De inverse is in feite hetzelfde netwerk maar het wordt door een ander ingangssignaal gedreven.

Omdat coördinatentransformaties een belangrijke rol spelen in Hoofdstukken 4 en 6, geven we een overzicht van een coördinatentransformatie (die dezelfde berekeningen gebruikt als diegene voor de relatieve graad) die een affien regelsysteem locaal in een canonieke vorm brengen. Deze canonieke vorm leidt tot het begrip van de nuldynamica van een systeem. Deze nuldynamica is de verzameling van ingangsignalen waarvoor de uitgang van het systeem altijd nul blijft. We kunnen deze nuldynamica voorstellen door een systeem afgeleid van de canonieke vorm en we kunnen deze nuldynamica dan gebruiken om een locale inverse van het systeem te berekenen. Ten slotte zien we ook hoe we de canonieke vorm kunnen gebruiken om het probleem van terugkoppelingslinearisatie op te lossen. Terugkoppelingslinearisatie is het probleem van het vinden van een terugkoppeling waarvoor het systeem lineair wordt. We besluiten door op te merken dat de toepassing van differentiaalmeetkundige methoden op neurale netwerken nog beperkt is en dat vooruitgang noodzakelijk is opdat deze aanpak een rijpe onderzoeksrichting kan worden.

Hoofdstuk 8. Integratie van gewone differentiaalvergelijkingen met behulp van compositiemethoden

Na de invoering van de werktuigen uit de differentiaalmeetkunde voor de analyse van regelsystemen in Hoofdstuk 7, keren we terug naar dynamische systemen. We voeren compositiemethoden in voor de integratie van gewone differentiaalvergelijkingen op de vakgebieden van neurale netwerken en regeltechniek. Hoewel deze methoden bekend zijn op het gebied van Hamiltoniaanse systemen, is hun gebruik voor de studie van neurale netwerken nieuw. Dit gebruik van compositiemethoden is nog een bijdrage van dit proefschrift die we in [90, 94, 98, 100, 102] reeds hebben voorgesteld.

Compositiemethoden steunen op de studie van de oplossingen van een gewone differentiaalvergelijking als eenparametergroepen van niet-lineaire transformaties, die exponentiële transformaties genoemd worden. De Lie-haak en de formule van Baker, Campbell en Haussdorff laten toe om de compositie van een aantal exponentiële transformaties te interpreteren als de oplossing van een nieuwe gewone differentiaalvergelijking. Het voornaamste idee achter compositiemethoden is dat, als we het vectorveld van de gewone differentiaalvergelijking in elementaire stukken kunnen splitsen, het mogelijk is een samenstelling van exponentiële transformaties (die oplossingen zijn van de elementaire stukken) op te bouwen die de oplossing van het meer complexe systeem benadert. We kunnen de berekeningen, die geassocieerd zijn aan zulke samenstellingen, meestal sneller uitvoeren dan de berekeningen voor standaard integratiemethoden.

We tonen eerst aan hoe we zulke compositiemethoden kunnen berekenen en passen deze methoden toe op de versnelling van een 'forwarding control law' voor de 'ball and beam' opstelling. Het gebruik van een compositiemethode is inderdaad eenvoudig en laat een beduidende versnelling van dit regelschema toe. We geven daarna een lijst van meer complexe integratoren die ook nauwkeuriger zijn. We geven ook een overzicht van een resultaat dat aantoont dat deze compositiemethoden een willekeurige nauwkeurigheid kunnen leveren, als ze maar complex genoeg zijn.

We beschouwen dan het probleem van de simulatie van grote roosters van circuits van Chua. Dit circuit is niet-lineair en heeft aanpasbare parameters. Door grote roosters van zulke circuits af te leiden, is het mogelijk om rekenenmechanismen op te bouwen voor signaalverwerking en voor de simulatie van niet-lineaire spatiotemporele fenomenen. Het hoge aantal variabelen (7500 in onze experimenten) maakt hun simulatie rekenintensief. Door de vergelijkingen op te splitsen in een lineair gedeelte en een (ontkoppeld) niet-lineair gedeelte, kunnen we een snelle compositie-integrator construeren die standaardintegratiemethoden verslaat met een factor van ten minste twee.

Hoofdstuk 9. Compositienetwerken voor systeemmodellering

In dit laatste hoofdstuk, stellen we een nieuwe aanpak voor de voorspelling van dynamische systemen voor : het compositienetwerk. Deze methode vindt haar oorsprong in de opmerking dat we de oplossing van een dynamisch systeem altijd kunnen uitdrukken als een inverteerbare transformatie (de exponentiële transformatie). Daarom is het belangrijk benaderingsschema's te ontwikkelen die de inverteerbaarheid behouden. Klassieke benaderingschema's, zoals veeltermen en statische neurale netwerken, kunnen inverteerbaarheid niet verzekeren. Aangezien het resultaat van een reeks composities van inverteerbare

transformaties altijd inverteerbaar is, beschouwen we de mogelijkheid om composities van transformaties te gebruiken voor de benadering en voorspelling van dynamische systemen. Deze aanpak blijkt compleet nieuw te zijn. We hebben ze in [91, 92, 93, 95, 97, 99] voorgesteld.

Aangezien we reeksen van composities reeds in Hoofdstuk 8 hebben onderzocht, zijn compositienetwerken een variatie op compositiemethoden. In ons geval, kennen we de parameters van de compositiemethode niet a priori. Onze methode is een 'grey-box'-aanpak waar we a priori kennis over de algemene structuur van het systeem eisen, maar waar we de parameters van het model uit beschikbare meetgegevens moeten schatten. We bespreken eerst hoe we de optimisatieproblemen kunnen oplossen die geassocieerd zijn met deze parameterschatting. We vergelijken dan compositienetwerken met meerlaagse perceptrons voor de voorspelling van de Lorenz attractor wanneer alle toestandsvariabelen meetbaar zijn. Compositienetwerken maken een efficiënt gebruik van de a priori kennis over de Lorenz attractor. Ze hebben minder meetgegevens nodig om dezelfde nauwkeurigheid van voorspelling te leveren. Ze zijn meer robuust tegen ruis. Ze kunnen sneller getrained worden. We kunnen onze methode tot slot ook met onregelmatige data gebruiken.

Nochtans is de veronderstelling dat de volledige toestand van het systeem meetbaar is, onrealistisch. We moeten daarom het geval beschouwen waar slechts één van de toestanden meetbaar is. In dit geval wordt ons probleem een eenvoudige vorm van systeemidentificatie. Opnieuw leveren compositienetwerken een goede performantie, dicht bij de performantie van een ideaal model. We kunnen onze methode verder uitbreiden om systemen met ingangen toe te laten. Het compositienetwerk is een 'grey-box'-model dat a priori kennis nodig heeft over het te voorspellen systeem. Voor 'black-box'-modellen, combineren we het compositienetwerk en het meerlaagse perceptron in wat we een 'MLP in dynamics space' noemen. We tonen aan dat zulke structuur dynamische systemen universeel kan benaderen.

Ten slotte tonen we aan dat er een sterk verband bestaat tussen compositiemethoden en het 'motion-planning'-probleem. Dit is het probleem waar we een regelwet proberen te berekenen die een systeem van een gegeven initiële toestand naar een gewenste eindtoestand kan brengen. Lafferiere en Sussman [72] hebben dit probleem bestudeerd in het geval van nilpotente systemen. We tonen aan dat, voor zulke systemen, compositiemethoden en het 'motion-planning'-probleem dezelfde formulering hebben.

Appendix A. Detectie van fraude in de mobiele telefonie met behulp van neurale netwerken

In de appendix stellen we werk voor dat de toepassing van neurale netwerken op de detectie van fraude in de mobiele telefonie betreft. Dit werk maakt deel uit van een Europees project over de veiligheid van mobiele communicatie.

Onze bijdrage bestaat uit de ontwikkeling van een prototype van een fraudedetectiesysteem, gebaseerd op een gesuperviseerd neuraal netwerk. We hebben dit prototype reeds in [1, 2, 22, 23, 89, 103, 104] voorgesteld. We geven ook een overzicht van twee andere prototypes — een regelgebaseerd systeem en een ongesuperviseerd neuraal netwerk — die door onze partners ontwikkeld werden.

De eerste stap in de ontwikkeling van een fraudedetectiesysteem is de analyse van de verschillende scenario's van fraude en de bepaling van de kenmerken in het gesprekspatroon van een gebruiker die op fraude wijzen. Alle drie prototypes steunen op de analyse van het gesprekspatroon van iedere gebruiker en proberen verdachte gedragsveranderingen op te sporen. De systemen gebruiken differentiaalanalyse. Dit betekent het volgende. Ze volgen het gedrag van een gebruiker op lange termijn en beschouwen dit gedrag als het normale referentieprofiel. Ze volgen ook het gedrag van de gebruiker op korte termijn om het huidige gedrag te schatten. De systemen ontdekken dan verdachte gebruikers wanneer er abnormale verschillen tussen de profielen op korte en lange termijn ontstaan. De systemen gebruiken een snelle databank om deze honderdduizenden profielen efficiënt te bewaren. Ze gebruiken ook een 'monitoring' systeem om de nodige informatie te verzamelen zodat een analist de verdachte gebruikers verder kan analyseren en een finale beslissing kan nemen.

Het regelgebaseerde systeem en het gesuperviseerde neurale netwerk hebben een gemeenschappelijke manier om de kenmerken die het gedrag van een gebruiker voorstellen te berekenen. Deze systemen gebruiken een eenvoudige filtreringsprocedure. Het gesuperviseerde neurale netwerk is een meerlaags perceptron dat we met het 'conjugate-gradient'-algoritme hebben getrained op reeksen van gesprekken van zowel fraudulente als niet-fraudulente gebruikers. Het geeft een goede performantie die we via het gebruik van het 'receiver-operating' kenmerk meten. Deze maatstaaf geeft een balans tussen detectie van fraude en valse alarmen. We gebruiken deze maatstaaf omdat valse alarmen een belangrijk probleem zijn voor een systeem dat zo veel gebruikers telt. Het regelgebaseerd systeem gebruikt een aantal regels die zowel differentiaalanalyse als absolute analyse (d.w.z. zonder referentie naar het profiel op lange termijn) uitvoeren. De alarmdrempels worden met de hand gezet. Het derde prototype is een ongesuperviseerd neuraal netwerk. In tegenstelling tot de andere prototypes zijn er hiervoor geen voorbeelden van fraude vereist. Het heeft alleen als doel abnormale gedragingen op te sporen. De reden hiervoor is dat de kenmerken van fraude snel veranderen eens fraudeurs zien dat hun strategieën niet meer werken en dat ze dan nieuwe vormen van fraude ontwikkelen. Het systeem schept profielen die de waarschijnlijkheidsverdeling van de gesprekken van iedere gebruiker voorstellen. Door deze profielen op korte en lange termijn op te bouwen, kan het systeem plotse veranderingen in de verdeling van de gesprekken van een gebruiker opsporen. Zulke veranderingen wijzen op een abnormale verandering van het gedrag van de gebruiker en vragen om een verdere analyse. Ieder systeem heeft zijn eigen voordelen en nadelen.

Chapter 1

Introduction

1.1 What neural networks are

As we wandered across different fields of research during our doctoral work—trying to gather knowledge that we thought helpful for the study of neural networks—we had the opportunity to interact with many people unfamiliar with neural networks. Their first interrogation was naturally to wonder what neural networks are. Answering this elementary question happens to be a real challenge as soon as we stop waving our hands and saying that "the human brain can perform complex pattern recognition, decision, and control tasks at low processing speed through its massively parallel architecture while the fastest computers cannot rival the brain for such tasks; so we are going to build massively parallel architectures that mimic the functioning of the brain". This idea did indeed give birth to the field of neural networks as the study of the adaptive behavior of assemblies of simple units interacting with each other. But it now dates back to the neuron model of McCullogh and Pitts in 1943 [84], the learning rule of Hebb in 1949 [55], and the perceptron of Rosenblatt in 1958 [120]. Such a brief description is not faithful anymore to the extent and diversity of neural networks. In the 1940s and 50's, neural networks were part of cybernetics [146] together with computer science, systems and control theory, artificial intelligence, signal processing, and information theory—all at an embryonic stage. Each of these disciplines went its own way in the 1960s with computer science, systems and control theory, and information theory and signal processing maturing rapidly. Neural networks and artificial intelligence were struggling against each other because they were competing towards the elusive goal of building intelligent devices. In 1969, Minsky—a major proponent of artificial intelligence—proved in his book Perceptrons [87] that Rosenblatt's model was severely limited. Neural networks could not deliver their promises and almost went into oblivion. In the 1980s, neural networks were reborn

with the discovery of backpropagation as a simple (albeit computer-intensive) method for the adaptation of networks more complex than Rosenblatt's perceptron. The possibility to overcome the limitations of the perceptron led to a boom of neural networks with herds of researchers moving into the field. The 1990s are a period of maturation with the development of the first real-world applications and the building of solid bridges to neighboring disciplines. The boom of the 1980s that brought researchers from other disciplines into neural networks, together with the already fuzzy agenda of parallel distributed processing, led to an heterogeneous community. The researchers, who repopulated the area of neural networks, came from a variety of fields and introduced their own techniques, thus making any brief synthesis of present neural network research impossible.

The notion of a community is then probably the most meaningful description of neural networks. By mentioning the fields with which neural networks communicate, we can give an idea of what people sharing this idea of parallel distributed processing do. But first, we have to say that the past rivalry between artificial intelligence and neural networks left them isolated from each other; symbolic logic is almost banned from neural networks. At the most biological end, neural networks interact with neuroscience through researchers, such as Sejnowski [127, 126], working on faithful models of real brains. This discipline is *computational neuroscience* and its object really is the study of the brain. Another discipline whose object is the study of the brain and that interacts with neural networks is cognitive science, with Anderson [9] as a representative. We can contrast these disciplines with approaches where researchers try to build machines, which perform functions similar to those of the brain, by using adaptive models of interconnected processing units; we call these approaches *neural computation*. Although there is a split between neural computation and computational neuroscience, the cohabitation between technologists and biologically-inclined scientists is smooth. The area where the overlap between the two sides is largest may be vision, where work is done by researchers from neurophysiology to statistics, such as Mumford [106] and Geman [16]. The development of methods for pattern recognition is certainly a primary goal of neural network research. A contribution of neural network theory is the representation of its methods as networks of elementary units. To achieve a given pattern recognition task, researchers then develop adaptation rules that "learn" the parameters of the network from available data. Obviously, learning brings neural networks within the realm of statistical modeling because the process we have described is a form of model estimation. As a consequence, the interaction between neural networks and statistics, although far from being smooth, is probably the most fruitful of all. Many talented statisticians—such as Friedman [39, 40], Breiman [19, 20], Ripley [118], and Tibshirani [141]—brought some much-needed rigor into the field of neural networks. This contribution was probably decisive for the maturation of the field.

In return, neural networks helped statistics to move towards more challenging problems than those it usually addressed. At the most mathematical end of neural networks, we connect to approximation theory, whose methods are necessary to study the capabilities of static neural networks [86]. In contrast, for neural networks whose state evolves over time, people such as Hirsch [58] and Kopell [34] exemplify the contributions of dynamical system theory. Of great interest to us is also the interaction between control theory and neural networks, with researchers from control theory such as Narendra [108, 77] and Sontag [133] who both made major contributions to the study of neural networks used for control. Moving further towards applications, we have circuit theory where Chua [24, 25] founded the domain of cellular neural networks; we also have signal processing with a researcher such as Kung [70]. Our list is by no means exhaustive. This interaction with other fields and the contributions it brought have given neural networks many faces. Of all the disciplines that find their origin in cybernetics, the field of neural networks is probably the most faithful to the original program of cybernetics, which was the understanding and synthesis of complex systems. Its imprecisely defined goal of parallel distributed processing gives it a refreshing enthusiasm and makes it one of the few spaces of research with a real interdisciplinarity.

1.2 Structure of this dissertation

There are many ways to contribute to the advancement of a field. The approach we prefer consists in the following. We try to enrich the body of knowledge about neural networks by bringing in theoretical elements from other fields. We then translate them into our technical language and apply them to our field. The influence of this philosophy is clear in Chapter 4, where we use a model from physics (the quasi-monomial form) to state an equivalence between neural networks and models from theoretical biology (the Lotka–Volterra systems). The same holds in Chapter 8, where we bring elements of Lie algebra theory and a numerical method for the integration of ordinary differential equations into the field of neural networks. It is again true in Chapter 9, where we develop new types of networks using these notions from Lie algebra theory and where we make a link between a problem in control, the approximation of dynamical systems, and the integration of ordinary differential equations. Furthermore, to stimulate interchanges between neural networks and its neighboring fields, we firmly anchor neural networks in approximation theory, dynamical system theory, and control theory in the review chapters.

Our drive when studying neural networks is a fascination for nonlinearity, in general, and the dynamics of these simple—yet general—systems, in particular. The idea that led to this dissertation is that invertible transformations play an important role in the study of continuous-time systems (through the expression

of their solutions as one-parameter groups of diffeomorphisms, through changes of coordinates, through the reconstruction of their state by an embedding of the output, and so on). Neural network researchers have made little use of invertible transformations so far, leaving room for original developments. We do not discuss this idea explicitly during our presentation because it is overly general, but we can note that the contributions in Chapters 4 and 6 use changes of coordinates (which are one-to-one transformations) and that Chapters 8 and 9 deal with compositions of invertible transformations for the approximation of dynamical systems.

Our thesis goes as follows: (1) dynamical neural networks and composition networks are adequate models of nonlinear systems, (2) the study of changes of coordinates gives us a deeper understanding of dynamical neural networks, and (3) Lie algebra theory is the right tool for the analysis of compositions of transformations used for the modeling of dynamical systems. We want this dissertation to link together topics from different fields, creating a broad and articulate picture of neural networks.

To defend our thesis effectively and to give the context necessary for an accurate interpretation, we organize this work as follows. In Chapter 2, we review the approximation properties of static neural networks because these properties guarantee that, even if neural networks have a simple structure, their behavior can be general. In Chapter 3, we introduce dynamical neural networks, which are dynamical systems with smooth saturations, and study their convergence and stability with the tools from dynamical system theory. In Chapter 4, we discuss a representation of dynamical neural networks as Lotka–Volterra systems, which are classical quadratic models from theoretical biology and dynamical system theory. In Chapter 5, we add inputs to our dynamical neural networks and obtain control systems; we then review the results from control theory applicable to neural networks. In Chapter 6, we study the approximation properties of dynamical neural networks with inputs through the development of a general linear change of coordinates for neural networks. In Chapter 7, we view these systems in the framework of differential-geometric control and introduce Lie algebra theory for the first time in this work. In Chapter 8, we present a successful use of Lie algebra theory through the integration of ordinary differential equations. In Chapter 9, we develop this application of Lie theory further and present composition networks for the prediction of dynamical systems. Finally, we present in an appendix some separate work on the detection of fraud in mobile communications using neural networks.

Our discussion follows a natural progression: from static networks to dynamical networks, then to control systems, followed by the extension to differential geometry and Lie algebra theory. Another important trend is the movement from abstract topics (approximation theory and dynamical system

theory) to applied topics (control theory, prediction of dynamical systems, and fraud detection), which is consistent with a sound engineering approach.

Because neural networks stretch across several disciplines and the literature has become vast in the past fifteen years, giving a review of neural network theory is impossible in a work of this size and scope. For a successful attempt, we therefore refer to the more than 1100-page and 300-author *Handbook of Brain Theory and Neural Networks* edited by Arbib [10]. Even if we were to limit ourselves to summarizing work at the intersection of the theories of neural networks, dynamical systems, and control; it would be difficult to be comprehensive and at the same time avoid complete dilution. In the parts of this work where we review existing literature, we therefore choose to explore on a limited number of key results in depth. We avoid the pitfall of eclecticism by concentrating on strong results that span our field of investigation and by insisting on what links them together. Our presentation of neural networks illustrates the methods that we can use to tackle problems involving neural networks.

As we go across several fields and want people from these fields to show interest in our work, we do not assume any particular background from the reader. We always give introductory material and precise definitions of the terms we use, so that everyone can follow our arguments. We discuss the proof of the results we review whenever we think that such a discussion helps to build the mathematical skills necessary to analyze neural networks. The precise extent to which we discuss these results is then the consequence of a tradeoff between completeness, relevance, the need for supplementary background material, and space considerations. As a minimum, we mention the main ingredients of the proof and stress the links with the results discussed before. To give proper credit to the authors whose work we review, we give the name of the author in the statement of theorems and main propositions. We also refer to these authors in the index. Our main goal was that the review chapters can serve as a roadmap of the literature—giving a bird's-eye view as well as precise directions. In the parts that contain original material, completeness was the primary objective and we discuss all results in detail.

1.3 Chapter-by-chapter overview

After this introductory sketch of the position of our work in relation to neural network theory and system theory, we move on to the body of this dissertation, which has two main parts. In the first part, we study the properties of different types of perceptron-type networks (static, dynamical, and dynamical with inputs) and, in the second part, we introduce the composition network as a new method for the modeling of dynamical systems. Chapters 2, 3, 5,

and 7 review the current literature while Chapters 4, 6, 8, 9, and the appendix contain original contributions. We outline the content of each chapter below.

Chapter 2. Neural networks for function approximation

We study neural networks used for the approximation of functions. More precisely, we review results about the universal approximation properties and rates of convergence of feedforward single–hidden-layer perceptrons.

Such a network computes a vector-valued function of its input in three stages. First, it computes an affine transformation of its vector of inputs into a space of different dimension. The resulting vector is the activation vector. Second, it computes a smooth nonlinear transformation of each component of the activation vector to produce what we call the hidden vector. The nonlinear, component-by-component transformation is often a saturation function (called a sigmoid)—that is, a function that is continuous, monotone, and has finite limits towards infinity. Third, it computes a linear transformation of the hidden vector to produce an output vector, which has a dimension that is possibly different from that of the input vector and the hidden vector. We can give a diagrammatic representation of such a computation as an acyclic graph. We call the components of the different vectors units or neurons. The input and output units are directly accessible while the intermediary units are hidden from direct measurements. The sigmoidal saturation plays a decisive role in giving this structure its nonlinear properties. Although many other types of neural networks do not use such sigmoidal units, the idea of a computational device built as a network of saturating units that interact with each other (either statically or dynamically) is certainly the founding idea of neural networks. The sigmoidal unit will recur throughout this dissertation, even if the neural networks that use them have different descriptions and different applications.

The objective of Chapter 2 is first to determine what kind of real-valued functions of several variables such networks can approximate. It turns out that single–hidden-layer perceptrons can approximate just about any function (for example, measurable functions) in more than one sense. We say that they have the universal approximation property. We begin by stating the now standard results of Cybenko [28] and Hornik, Stinchcombe, and White [62]. We then move to a more recent result of Leshno [75] that states that a single–hidden-layer perceptron can universally approximate continuous functions if and only if its activation function is not a polynomial. An important point is that we achieve this result essentially by using a classical tool from analysis (the Weierstrass theorem) which states that polynomials can approximate continuous functions universally. Such an approach is edifying in that it shows that the much-touted universal approximation property of neural networks is overrated. But, as neural networks have proved themselves more useful than polynomials for high-dimensional problems, we consider the possibility that

they provide more compact (hence, easier to estimate) representations than polynomials. We review Barron's work [13] on the rates of convergence of neural networks—that is, we study how the approximation error decreases with the number of neurons of the network. We see that, for some interesting classes of functions, single–hidden-layer perceptrons do indeed provide more compact representations than polynomials. We then discuss the proper context and necessary nuances for this result.

Finally, we consider the question of whether two neural networks can represent the same function because, if this is the case, we need to pay some extra care when analyzing them. It turns out that two different networks can indeed represent identical functions; but two such networks are related by a simple symmetry, which makes the problems related to a redundant representation less acute.

Chapter 3. Neural networks as dynamical systems

We review here the convergence and stability properties of dynamical neural networks. While the static networks of Chapter 2 gave at their output a nonlinear transformation of their input, dynamical neural networks evolve over time and we describe them by a set of ordinary differential equations. We study different types of dynamical neural networks from the point of view of dynamical system theory. Again, the sigmoidal activation plays a central role because the ordinary differential equations describing the neural networks contain sigmoidal transformations.

We introduce the basic definitions of convergence of the trajectories of a system to some fixed point, and of stability of a fixed point. We then introduce the tools we use to characterize these properties: Lyapunov's theorem and LaSalle's invariance principle.

First, we consider conditions that guarantee convergence of a neural network. A general result we address is the Cohen–Grossberg theorem [26]. It gives sufficient conditions for convergence for a class of competitive systems that is a generalization of neural network models. The theorem is applicable to a wide range of models and its proof relies on the construction of Lyapunov functions for the systems under consideration. However, its requirement that the interaction matrix be symmetric and positive restricts its practical application. The next results we consider are those of Hirsch [58] on the convergence of cascades of neural networks. These systems are made of a sequence of subnetworks that have their own dynamics and for which a subnetwork feeds its output only into the subnetworks that follow it in the cascade. These results rely also on the construction of Lyapunov functions for cascades of networks.

Second, we consider conditions that guarantee global asymptotic stability of a neural network. We briefly review a simple result that relies on multiple local linearizations to guarantee stability. Next, following Forti and Tesi [38], we

introduce Lur'e–Postnikov Lyapunov functions for neural networks. Again, we use Lyapunov's theorem to prove stability, but we use here a reverse-engineering approach. Indeed, we fix the form of the Lyapunov function a priori and then try to determine whether values of its parameters exist for which the function satisfies the conditions of Lyapunov's theorem.

Chapter 4. Representation of neural networks as Lotka–Volterra systems

Here we prove that we can represent different types of recurrent neural networks as Lotka–Volterra systems. This result is an original contribution of this dissertation and we presented it in [88]. Lotka–Volterra systems are a classical model from theoretical biology. Their main characteristic is their simplified quadratic form, which makes their analysis easier than the analysis of dynamical neural networks.

We start with the simple case of dynamical perceptrons. Their simple structure somewhat limits their range of behaviors, but their transformation clearly illustrates the general process. The first step is to transform the dynamical perceptron into what we call a quasi-monomial form. Such a form is a system of ordinary differential equations for which the dynamics of each variable is equal to the product of this variable with a linear combination of products of powers of all the variables. The second step then consists in transforming the quasi-monomial form into a Lotka–Volterra system, following the procedure proposed by Brenig [21]. We perform the first step by a simple change of variables where we posit the sigmoidal transformation of the old variables as our new variables. Because of particular properties of the derivatives of the two most common activation functions (the hyperbolic tangent and the logistic sigmoid), this change of variables directly produces a quasi-monomial form. In this quasi-monomial form, products of powers of the variables determine the differential equation of the system. We perform the second step by considering these products of powers of the variables as additional variables of the systems. Simplifying the resulting expression, we observe that the system has taken a Lotka–Volterra form. This transformation means that, for any dynamical perceptron, we can give a Lotka–Volterra system of higher dimension for which the behavior of its first coordinates (as many as the dimension of the dynamical perceptron) is equivalent to the behavior of the neural network, for an appropriately chosen initial condition.

Once we have performed this transformation for dynamical perceptrons, we can perform it for dynamical perceptrons with self-connections. (These systems exhibit more general behaviors than dynamical perceptrons.) We also take special care to give the exact parameters of the equivalent Lotka–Volterra system and not only state its existence. We also generalize this transformation further to the case of dynamical multilayer perceptrons.

Chapter 5. Control theory of neural networks

In this chapter, we consider dynamical neural networks again but this time we add external inputs to them. By doing so, we leave the field of dynamical system theory to enter the field of systems and control theory. Our first topic is a question of practical importance: We review how to adapt the parameters of a Hopfield-type network to produce a desired input–output behavior. We present an algorithm that performs this task, the backpropagation-through-time algorithm, and use methods from optimal control [32] to derive it. This problem is a simple form of system identification and is therefore of importance from the point of view of control theory. Even more important for control theory are the notions of observability and controllability. The work of Albertini and Sontag [7] settles the question of observability by giving necessary and sufficient conditions under which neural networks are observable. The work of Sontag and Sussman [133] addresses the question of controllability. The proof of their theorem relies on a result on convex hulls and a local controllability lemma. One of their main conclusions is that the choice of the activation function is important for controllability. They exhibit a network for which changing the sigmoidal activation to another activation, which looks qualitatively the same, causes the loss of controllability. As in the case of static neural networks, the question arises whether two dynamical networks can have the same input–output behavior. The conditions that Albertini and Sontag [6] give for input–output equivalence of neural networks are surprisingly similar to those in the static case. From these three results (observability, controllability, and equivalence) arise two major notions: (1) the concept of an activation having the independence property (which means that sets of translates of the activation always remain linearly independent), and (2) the concept of an irreducible matrix (for which two rows are never equal up to a change of sign). The irreducibility of the matrix plays a role analogue to that of a matrix being of maximal rank in the case of linear systems.

The application of neural networks in practical control problems requires the use of discrete-time models because of the preeminence of digital control. We therefore review different control schemes using neural networks in discrete-time. We consider both input–output models and state-space models, as presented by Narendra and his students [108, 109, 76, 77]. This discrete-time approach leads to inverse control, internal model control, and model-reference adaptive control in the case of input–output models. It leads to observer and controller design for state-space models.

Chapter 6. Approximation by dynamical neural networks

We study here the approximation of dynamical systems by dynamical neural networks. On the one hand, the approximation results for static neural

networks are directly applicable to dynamical single–hidden-layer networks through the approximation of the vector field of the dynamical system. On the other hand, it is not obvious that dynamical networks without hidden layer can approximate arbitrary dynamical systems. We show in this chapter that, as long as networks without hidden layer have an additional linear output map, they can approximate arbitrary dynamical systems because they are equivalent to dynamical networks with a hidden layer. The precise determination of this equivalence and its statement as a general theorem applicable to a wide range of neural networks is an original contribution of our work, see [96, 101].

These equivalences between dynamical neural networks with and without hidden layer rely on linear changes of coordinates between dynamical systems of different dimensions. The central notion here is the immersibility of a lower-dimensional system into a higher-dimensional one, which means that, for an appropriate subset of initial conditions, the input–output behavior of the higher-dimensional system is identical to that of the lower-dimensional one. The main point of the discussion is the proof of a theorem on the equivalence of neural-like dynamical systems with and without hidden layer. For the systems under consideration, we do not require that the nonlinear transformation act element by element: It can be any differentiable map. The proof is a generalization of the case where the matrix determining the hidden dynamics of the system is invertible and for which there is a trivial change of coordinates giving the desired result. In the more general case of nonsquare matrices, we need to analyze the system by using the singular value decomposition of the matrix under consideration. This analysis requires some care. Once we have obtained our main theorem, we can apply it to neural networks. We can also, under some additional conditions, extend it to neural networks with self-connections.

We can then use this equivalence to prove that dynamical perceptrons can approximate arbitrary dynamical systems, following the work of Sontag [131]. We also review a related result by Funahashi and Nakamura [42] on the universal approximation capabilities of Hopfield-type networks. This result uses a change of coordinates, which is the dual of ours, to state the equivalence between Hopfield-type networks and dynamical single–hidden-layer perceptrons.

Chapter 7. Lie algebra theory of nonlinear control systems

In this review chapter, we add the tools from differential-geometric control to control theory. Many of the developments are limited to control systems that are affine in the inputs. We introduce the concepts of a Lie derivative and of a Lie bracket. The Lie derivative quantifies the evolution of a real-valued function along a trajectory of a control system. The Lie bracket is a generalization to vector fields of the vector product. These objects are useful to study the geometry of control systems as they permit all sorts of comparisons between trajectories, vector fields, and real-valued functions. The purpose of

this chapter is to review the application of the methods of differential-geometric control to neural networks.

We first review Frobenius's theorem. This theorem uses the Lie bracket to check whether the behavior of a control system started at some initial condition is constrained to stay on some surface or if it is free to reach any point in space. The statement of the theorem is deeply linked to the notion of controllability. It gives rise to what Herman and Krener [56] call the controllability rank condition. This condition guarantees some form of controllability by checking the rank of a set of brackets of the vector fields determining the control system. For observability, the observability rank condition, which is dual to the controllability rank condition, guarantees some form of observability.

Following Żbikowski [151], we apply the controllability rank conditions to dynamical neural networks. Computational difficulties make the controllability and observability rank conditions impossible to check. Differential-geometric control cannot, unfortunately, deliver results comparable to those of Chapter 5. We then turn to the study of inverses of dynamical neural networks. For this analysis, we need to introduce the notion of the relative degree of a control-affine system, which is the number of times we have to differentiate the output of the system to recover an explicit dependence on the input. A theorem of Hirschorn [60] then provides a way to compute the inverse of a control-affine system by using iterated Lie derivatives. The application of this theorem to Hopfield-type networks, as proposed by Khambapati and his coworkers [67], gives necessary conditions on the parameters of the network for which we can explicitly compute the inverse system. The inverse is actually the same network but driven by a different input.

Because changes of coordinates play a central role in Chapters 4 and 6, we review a change of coordinates (based on the computations associated to the relative degree) that brings a control-affine system locally into a canonical form. This canonical form leads to the concept of the zero dynamics of a system, which is the set of inputs for which the output of the system remains identically zero. We can represent this zero dynamics by a system derived from the canonical form and use it to find a local inverse for the system. Finally, we see how we can also use this canonical form to tackle the problem of feedback linearization, which amounts to finding a feedback law under which the system becomes linear. We conclude by observing that the application of differential-geometric methods to neural networks is still fragmentary and that significant progress is necessary before this approach can become a mature direction of research.

Chapter 8. Integration of ODEs by composition methods

After the introduction of the tools of differential geometry for the analysis of control systems in Chapter 7, we return to dynamical systems. We introduce

the composition methods for the integration of ordinary differential equations into the fields of control and neural networks. Although these methods are well-known in the field of Hamiltonian systems, their use for the study of neural networks is new and this use is another contribution of this dissertation, which we presented in [90, 94, 98, 100, 102].

These methods rely on the study of the solution of an ordinary differential equation as a one-parameter group of nonlinear transformations, called exponential maps. The Lie bracket and a formula called the Baker–Campbell–Hausdorff formula offer the possibility of interpreting the composition of several exponential maps as the solution to some new ordinary differential equation. The main idea behind composition methods is that, if we can split an ordinary differential equation into elementary parts, it is possible to construct a sequence of exponential maps (solution of the elementary parts) that approaches the solution of the more complex system. We can often perform the computations related to such sequences of compositions faster than the computations for standard integration schemes.

We first show how we can compute such composition methods and then apply these methods to the acceleration of a forwarding control law for the ball and beam. The use of a composition method is indeed easy and allows a significant speed-up of this control scheme. We then present a list of more complex but more accurate integrators. We also review a result that shows that these methods can achieve any desired accuracy if they are sufficiently complex.

We then consider the simulation of large arrays of Chua's circuits. These circuits are nonlinear and have adjustable parameters. By constructing large arrays of such circuits it is possible to construct devices for signal processing or for the simulation of nonlinear spatiotemporal phenomena. The large number of variables (7500 in our experiments) makes their simulation computationally intensive. By splitting the equations between the linear part and the (decoupled) nonlinear part, we can construct a fast composition integrator that outperforms standard methods by a factor of at least two.

Chapter 9. Composition networks for system modeling

In this final chapter, we present a new approach for the prediction of dynamical systems: the composition network. This method starts with the observation that we can always express the solution of a dynamical system as an invertible transformation, which we call the exponential map. That is why it may be interesting to develop approximation schemes that preserve invertibility. Standard approximation schemes, such as polynomials or static neural networks, cannot guarantee invertibility. Because sequences of compositions of invertible transformations result in an invertible transformation, we consider the possibility of using compositions of transformations for the approximation and

prediction of dynamical systems. This approach appears to be completely new; we presented it in [91, 92, 93, 95, 97, 99].

As we have already investigated sequences of compositions in Chapter 8, composition networks turn out to be a variation on the composition methods for which we do not know the parameters a priori. Our method is a grey-box approach requiring knowledge of the general structure of the system while we can estimate the parameters of the model from available data. We first discuss how to solve the optimization problem associated with this parameter estimation procedure. We then compare composition networks with the multilayer perceptron for the prediction of the Lorenz attractor when all the state variables are accessible. The composition networks make efficient use of the a priori knowledge about the Lorenz attractor: They need significantly less samples than multilayer perceptrons for a prediction that is equally accurate. They are more robust to noise. They are also faster to train than multilayer perceptrons. We can furthermore use our method with data sampled at irregular intervals.

Yet the assumption that the full state is accessible is not realistic. We therefore consider the case where only one of the states is accessible. In this case, our problem becomes a system identification problem. Again, the composition network performs well, coming close to an ideal model. We can also modify our method to allow systems with inputs. The composition network is a grey-box model requiring a priori knowledge about the system we want to predict. For black-box modeling, we combine the composition network and the multilayer perceptron to give what we call an MLP in dynamics space. We show that such an architecture can universally approximate dynamical systems.

Finally, we show that a strong link exists between composition methods or approximation by compositions, and the motion-planning problem. The motion-planning problem is the situation where we try to compute an open-loop control law that steers a system from a given initial condition into a desired final condition. Lafferiere and Sussman [72] studied this problem for the case of nilpotent systems. We show that, for such systems, composition methods and the motion-planning problem share the same formalism.

Appendix A. Detection of mobile phone fraud using neural networks

In the appendix, we present work on the application of neural networks to the detection of mobile phone fraud. This work is part of a European project on security in mobile communications. Our contribution is the development of a fraud detection prototype based on a supervised neural network, as presented in [1, 2, 22, 23, 89, 103, 104]. We also review two other prototypes, a rule-based system and an unsupervised neural network, developed by the other partners of the project.

The first step in the design of a fraud detection tool is the analysis of the different scenarios of fraud and the determination of the characteristics indicative of fraud in the calling pattern of a user. All three prototypes rely on the analysis of the calling pattern of each user and try to detect suspicious changes of behavior. The systems use differential analysis. This expression means the following. We keep track of the behavior of the user on a long-term basis and consider this behavior as the normal reference profile. We also monitor the profile of the user on a short-term basis to estimate the current behavior. The systems then detect suspicious users when an abnormal difference between the short-term and the long-term profiles occurs. These systems use a fast database to manage the several hundreds of thousands of profiles efficiently. They also use a monitoring system to gather the necessary information so that a fraud analyst can review the suspicious users proposed by the systems and make a final decision.

The rule-based system and the supervised neural network share a common way of computing the characteristics (duration and frequency of calls on a short-term and long-term basis) representative of the behavior of the user. The systems can compute these profiles efficiently through a simple filtering strategy. The supervised neural network is a multilayer perceptron trained with the conjugate-gradient algorithm on labeled sequences of calls from fraudulent as well as normal users. It delivers good performance, which we assess via a receiver operating characteristic. This criteria measures the tradeoff between correct detections and false alarms. We use it because false alarms are a major problem for a system containing so many users. The rule-based system contains a set of rules performing differential analysis and absolute analysis (i.e., without reference to the long-term profile). The thresholds of the rules are optimized manually. The third prototype is the unsupervised neural network. In contrast with the other prototypes it does not require examples of fraud for its development. It aims only at the detection of abnormal behaviors to catch new forms of fraud as fraud tends to change its characteristics once fraudsters realize that their fraud schemes are being defeated. The system creates a profile representative of a probability distribution of the calls of each user. By profiling on a short-term and long-term basis, the system can detect when the distribution of the calls of the user changes suddenly. Such a change is indicative of an abnormal change in the behavior of the user and requires further analysis. All three systems have their own advantages and drawbacks.

1.4 A few remarks about style and notation

Throughout this dissertation, we strive towards a plain, dynamic style. To this end, we follow mostly the recommendations of Lyn Dupré [33] and Nicholas Higham [57]. One of the difficulties is that a rigorous treatment of neural

networks requires heavy notation, especially as we wander through the theories of approximation, dynamical systems, control, and Lie algebras. Developing a consistent and light notation was a challenge. In particular, trying to avoid a clutter of subscripts and superscripts—as Knuth [68] recommends—was not always easy.

We have adopted the following notational conventions. We denote real values by lowercase Greek letters; except for the activation function σ or ρ of a neural network, a projection π, and the one-to-one maps φ, ψ, Φ, and Ψ. We denote vectors, real-valued functions, and vector fields by lowercase letters; except for the time t and the bias vector a_0 for which we always use the subscript zero. We denote matrices, cost-type functions (the Lyapunov function V, the Hamiltonian H, the cost Q, and the error function E), and upper or lower bounds by uppercase letters, except for the function N implemented by a neural network. We denote sets, distributions, functional operators, and manifolds by calligraphic letters. Finally, we sometimes denote trajectories of a differential equation by uppercase Greek letters. We also denote a diagonal matrix by Λ; its diagonal elements then forming the vector λ. A major departure from these conventions occurs in Chapters 8 and 9. In these chapters, we denote vector fields by uppercase letters to follow the conventions of Lie algebra theory (fortunately, we do not need matrices there).

Furthermore, we use subscripts to index objects, including vectors and matrices. However, we also use subscripts to denote entries in a vector and double subscripts to denote entries in a matrix. We use these conflicting rules to avoid a clutter of notation. We resolve the conflicts by using a bracketed superscript to index vectors and matrices whose entries we need to access. We also use bracketed superscripts to denote multiple instances of the same object. An important exception is that we denote the kth derivative of the activation function σ by $\sigma^{(k)}$ to avoid the visual burden of writing $\frac{d\sigma^k(\xi)}{d\xi^k}$ often.

Another eccentric property of the activation function σ is that it can stand for a vector-valued function as well as for the real-valued function that is the building block of its vector-valued counterpart. We avoid confusion by noting that the vector-valued function itself acts on a vector (we always have an expression like $\sigma(x)$) while the real-valued function always acts on a real value (we always have an expression like $\sigma(\xi)$). We invite the reader to come back here whenever necessary to become familiar with the notation.

Chapter 2

Neural networks for function approximation

The advent of cheap computational power in the 1980s led to the revival of the field of artificial neural networks and saw some important applications in pattern recognition—for example in handwriting recognition [74], speech recognition [17], or text-to-speech [126]. As the dormancy of the field had been the result of the severe limitations [87] encountered by Rosenblatt's perceptron [120] (the precursor of feedforward networks [122]), the question arose as to what the capabilities of multilayer perceptrons exactly were. The main use of neural networks was then for classification tasks, so the first results on their capabilities appeared for classification problems solved by two–hidden-layer networks [78, 43]. However, classification is a particular type of regression problem and the question quickly became: What kind of functions can neural networks approximate? Trying to go beyond the wild claims that neural networks could "mimic" the functioning of the brain because they were "universal approximators" (as often heard at the turn of the 90's), we want to give here a clear picture of some important results on approximation using neural networks. We want to give an idea of which techniques we can use to prove these results and achieve this goal with a mostly self-contained presentation that requires only a passive background in analysis.

2.1 Early results

From 1989 on, mathematicians began publishing characterizations of the universal approximation capabilities of neural networks. By universal approximation, we mean that we have a family of functions $\mathcal{N}$ that is dense in a "large"

class of functions, such as the set of all continuous functions or of all measurable functions (recall that a function is measurable if it is possible to compute its integral, which may be infinite, with respect to some measure of volume, in our case the Lebesgue measure).

We need first to describe neural networks mathematically. What the neural networks we consider have in common is that they all use sigmoidal transformations, which we define now.

Definition 2.1. An activation function σ from $\mathbb{R}$ to $\mathbb{R}$ is *sigmoidal* if

$$\lim_{\xi \to -\infty} \sigma(\xi) = 0 \text{ and } \lim_{\xi \to \infty} \sigma(\xi) = 1.$$

Different types of neural networks use such activation functions. The main examples are

- Single–hidden-layer perceptrons with a single output:

  $$N(x) = b_0 + \sum_{i=1}^{n} b_i \sigma(a_i x + a_0),$$

  where $N(x) : \mathbb{R}^d \to \mathbb{R}$ and $x \in \mathbb{R}^d, a_0, \ldots, a_n, b_0, \ldots, b_n \in \mathbb{R}$.

- Single–hidden-layer perceptrons with multiple outputs:

  $$N(x) = B\sigma(Ax + a_0) + b_0,$$

  where $N(x) : \mathbb{R}^d \to \mathbb{R}^p$ and $x \in \mathbb{R}^d, A \in \mathbb{R}^{n \times d}, a_0 \in \mathbb{R}^n, B \in \mathbb{R}^{p \times n}, b_0 \in \mathbb{R}^p$.

- Multilayer perceptrons (MLP)

  $$N(x) = C(\sigma B \sigma(Ax + a_0) + b_0) + c_0,$$

  where $N(x) : \mathbb{R}^d \to \mathbb{R}^p$ and $x \in \mathbb{R}^d, A \in \mathbb{R}^{n \times d}, a_0 \in \mathbb{R}^n, B \in \mathbb{R}^{m \times n}, b_0 \in \mathbb{R}^m, C \in \mathbb{R}^{p \times m}, c_0 \in \mathbb{R}^p$.

For the multilayer perceptron, we organize the sigmoidal transformations in layers and each sigmoid acts on an affine transformation of the sigmoid at the previous layer. In the example given above, we present a two–hidden-layer perceptron as an illustration. We say that these networks are all feedforward networks because we organize the computations in layers and that there are no feedback loops (which we call recurrences also).

The best known results are those of Cybenko [28] and of Hornik, Stinchcombe, and White [62]. These results treat only the case of the approximation of functions from $\mathbb{R}^n$ to $\mathbb{R}$ by single–hidden-layer feedforward networks. We limit ourselves to the single-output case because we can approximate functions

from $\mathbb{R}^n$ to $\mathbb{R}^q (q > 1)$ by approximating each of its components separately. Furthermore, the results are valid for MLPs as we can take care that the activation functions in the hidden layers after the first one have no nonlinear effect by forcing them to work in their linear region (we make all their weights small enough). All the theorems involve the following elements in their assumptions. First, we consider some large class of functions, such as the continuous functions, the L_p functions, or functions that are finite with respect to some measure. Second, we consider some norm—that is, some notion of distance between two functions—like the L_2 or L_∞ norms, or some norm defined with respect to some measure. To make sure that the distance between two functions remains finite, we define our functions only on a compact set or, when our functions and our norm are defined with respect to some measure, we request that this measure be finite. Third, we choose some set of admissible activation functions, such as continuous sigmoidal functions, or bounded measurable functions. The statement of these theorems is that—for some appropriate choice of the class of functions, of the norm, and of the class of activation functions—the set of neural networks with any such activation function is dense under this norm in this class of functions.

Definition 2.2. Let $\mathcal{C}$ and $\mathcal{N}$ be two sets of functions belonging to some metric space $\mathcal{B}$. Given a norm $\rho(x)$ such that the distance $d(x, y) = \rho(x - y)$ is defined for all $x \in \mathcal{N}, y \in \mathcal{C}$; we say that $\mathcal{N}$ is *dense* under the ρ norm in $\mathcal{C}$ if, for any $f \in \mathcal{C}$ and any $\epsilon > 0$, there exists $g \in \mathcal{N}$ such that $d(f, g) < \epsilon$.

Definition 2.3. Let $\mathcal{C}$ be a set of functions from $\mathbb{R}^d$ to $\mathbb{R}$. Let $\mathcal{N}_\sigma^d$ be the set of neural networks from $\mathbb{R}^d$ to $\mathbb{R}$ with activation function σ. Given a norm $\rho(x)$; we say that the neural networks $\mathcal{N}_\sigma^d$ have the *universal approximation property* if $\mathcal{N}_\sigma^d$ is dense in $\mathcal{C}$ under the norm ρ.

An important family of norms are the L_p norms $(1 < p < \infty)$. We define these norms as

$$||f||_p = \left(\int_{\mathbb{R}^d} |f(x)|^p dx \right)^{1/p}.$$

The set of L_p functions is the set of functions that have a finite L_p norm. We define the L_∞ norm as

$$||f||_\infty = \inf_{M \in \mathbb{R}} \left\{ M : \int_{\{x \in \mathcal{R}^d : |f(x)| > M\}} dx = 0 \right\}.$$

Cybenko states that feedforward networks with continuous sigmoidal activation functions are dense in the set of continuous functions of the unit hypercube under the L_∞ norm and that feedforward networks with bounded L_1 activation functions are dense under the L_1 norm in the space of L_1 functions. We denote the unit hypercube $[0, 1]^d$ by I^d.

Proposition 2.1 (Cybenko). *Let σ be any continuous sigmoidal function. Consider the set $C(I^d)$ of all continuous functions on the unit hypercube. Consider the supremum norm L_∞. For this set of functions and this norm, $\mathcal{N}_\sigma^d$ has the universal approximation property.*

Proposition 2.2 (Cybenko). *Let σ be any bounded measurable sigmoidal function. Consider the set $L_1(I^d)$ of all absolutely integrable functions on the unit hypercube. Consider the absolute norm L_1. For this set of functions and this norm, $\mathcal{N}_\sigma^d$ has the universal approximation property.*

Cybenko based his proofs on the Hahn–Banach theorem [121]. This theorem is a result from functional analysis we do not wish to detail further here.

In contrast, Hornik, Stinchcombe, and White [62] state that feedforward networks with nondecreasing sigmoidal activation functions are dense under the L_∞ norm in the set of continuous functions on any compact set and that feedforward networks with nondecreasing sigmoidal activation functions are dense in the set of all measurable functions under the norm induced by a probability distribution. We first have to recall that a *compact set* is—in the particular case of $\mathbb{R}^d$—any closed bounded subset of $\mathbb{R}^d$, where closed means that the set contains all its limits points. We must also define the type of activation we allow for our networks.

Definition 2.4. A sigmoidal activation function σ from $\mathbb{R}$ to $[0,1]$ is a *squashing* function if it is nondecreasing.

Remark that we require the activation function to be sigmoidal but not to be continuous. We now can state the results.

Proposition 2.3 (Hornik et al.). *Let σ be any squashing function. Let $\mathcal{K}$ be any compact set in $\mathbb{R}^d$. Consider the set $C(\mathcal{K})$ of all continuous functions on $\mathcal{K}$. Consider the supremum norm L_∞ on $\mathcal{K}$. For this set of functions and this norm, $\mathcal{N}_\sigma^d$ has the universal approximation property.*

Proposition 2.4 (Hornik et al.). *Let σ be any squashing function. Consider the set M^d of all measurable functions from $\mathbb{R}^d$ to $\mathbb{R}$. Consider any probability distribution $p(x)$ on $\mathbb{R}^d$ and the following norm ρ_p:*

$$\textit{For all } f \in M^r : \rho_p(f) = \inf_{\epsilon>0} \left\{ \epsilon : \int_{\{x \in \mathbb{R}^d : |f(x)| > \epsilon\}} p(x)dx < \epsilon \right\}.$$

For this set of functions and this norm, $\mathcal{N}_\sigma^d$ has the universal approximation property.

The probability distribution is necessary to make sure that the norm is finite. Proposition 2.4 is a direct consequence of Proposition 2.3 by the fact that the set of continuous functions is dense in the set of measurable functions under a ρ_p norm. These results by Hornik are obtained not by studying

single–hidden-layer neural networks directly, but by studying what they call $\Sigma\Pi$ networks. These are networks where the output is equal to a linear combination of *products* of the hidden neurons instead of just a linear combination of the hidden neurons. They apply the Stone–Weierstrass theorem to this type of network and obtain results for single–hidden-layer networks as the special case where the products are restricted to a single neuron. We see that feedforward networks can indeed approximate large classes of functions. The main difficulty appears to come from the choice of the norm. Although it is not difficult to approximate discontinuous functions in the L_p norm ($1 < p < \infty$), approximating discontinuous functions in the L_∞ norm is a difficult problem in general as was shown by Sontag [131].

The question of the approximation capabilities of single–hidden-layer networks has been almost completely settled by Leshno and his coworkers [75] as they give *necessary and sufficient* conditions for universal approximation. All they require is that the activation be piecewise continuous and locally bounded. This condition is weak because it does require neither continuity nor boundedness. They state that—as long as the activation function is *not a polynomial*—single–hidden-layer feedforward networks can approximate continuous functions in the L_∞ norm and they can approximate weighted-L_p functions under the weighted-L_p norm. Their result is for the most part rather easy to follow and is instructive about the techniques used to study approximation capabilities of neural networks, we therefore discuss it in some more details after we have presented the Weierstrass theorem.

2.2 Weierstrass's theorem

The central tool for our study of the approximation capabilities of feedforward neural networks is the Weierstrass theorem. It states that polynomials of a real variable are dense in the set of continuous functions of a real variable on any closed interval. Recall first that we say that a sequence of functions $f_1, f_2, \ldots : \mathcal{K} \to \mathbb{R}$ converges to another function $f : \mathcal{K} \to \mathbb{R}$ *uniformly* if, for any $\epsilon > 0$, there is an M such that $|f_n(x) - f(x)| < \epsilon$ for any $k > M$ and $x \in \mathcal{K}$.

Theorem 2.5 (Weierstrass). *For any real continuous function f of a single real variable defined on the interval $[a, b]$, there exists a sequence of polynomials $P_1(\xi), P_2(\xi), \ldots$ that converges uniformly to f on $[a, b]$:*

$$\lim_{n \to \infty} P_n(\xi) = f(\xi) \quad \textit{uniformly.}$$

Proof. We consider the case where f is defined on $[0, 1]$; this assumption is without loss of generality by an appropriate translation and dilation of the x-axis. By definition, we suppose that f is zero outside of this interval, so that f

is uniformly continuous on $\mathbb{R}$. We suppose also that $f(0) = f(1) = 0$. If this is not the case and we are looking at some function g that does not satisfy this condition, then the function $f(\xi) = g(\xi) - g(0) - \xi(g(1) - g(0))$ does satisfy the condition. If we find a sequence of polynomials such that $\lim_{n\to\infty} P_n(\xi) = f(\xi)$, then $\lim_{n\to\infty} P_n(\xi) + g(0) + \xi(g(1) - g(0)) = g(\xi)$ and the sequence remains a sequence of polynomials.

We define the following sequence of polynomials $Q_1(\xi), Q_2(\xi), \ldots$:

$$Q_n(\xi) = c_n(1 - \xi^2)^n, \text{ where we choose } c_n \text{ such that } \int_{-1}^{1} Q_n(\xi)d\xi = 1.$$

Further, we define the sequence of polynomials $P_1(\xi), P_2(\xi), \ldots$ by

$$P_n(\xi) = (f * Q_n)(\xi) = \int_{-1}^{1} f(\xi - \zeta)Q_n(\zeta)d\zeta, \quad \text{for } 0 \leq \xi \leq 1.$$

Remark that the integral goes from -1 to 1 since ξ is restricted to the interval $[0, 1]$ and f is zero outside the same interval. The idea is the following. The terms $(1 - \xi^2)^n$ converge pointwise to zero except at the origin where they always stay equal to one. But we require that the integral of Q_n always stays equal to one. The only possibility is then that Q_n tends to a Dirac impulse. But we know that the convolution of a function with a Dirac impulse $\delta(\xi)$ is equal to the function itself. Formally at least, we have

$$(f * \delta)(\xi) = \int_{-1}^{1} f(\xi - \zeta)\delta(\zeta) = f(\xi).$$

Therefore, the sequence $P_n(\xi)$ converges pointwise to $f(\xi)$. Furthermore, the convolution is commutative and we have

$$P_n(\xi) = (f * Q_n)(\xi) = (Q_n * f)(\xi) = \int_{0}^{1} f(\zeta)Q_n(\xi - \zeta)d\zeta.$$

Remark that the integral goes from 0 to 1 since $f(\xi)$ is zero outside the $[0, 1]$ interval. This last expression shows that we can regard the integrand as a polynomial in ξ with coefficients dependent on ζ. Thus, after integration with respect to ζ, we have that $P_n(\xi)$ is a polynomial in ξ. The only technical problem we are left with is that we want the convergence of the sequence of polynomials to be uniform. We solve this problem by uniformly bounding the difference between the polynomials and the function itself. A computation shows that $c_n < \sqrt{n}$. For any $\delta > 0$, we have $Q_n(\xi) < \sqrt{n}(1 - \delta^2)^n$. This inequality means that Q_n converges to zero uniformly if we stay away from the origin (for all ξ such that $\delta \leq |\xi| \leq 1$). We now evaluate the difference between these polynomials and the function $f(\xi)$. For any $\epsilon > 0$, we take $\delta > 0$ such that $|f(\xi + \delta) - f(\xi)| < \epsilon/2$ for all ξ (this relation is possible because f

is uniformly continuous on the interval $[-1,1]$) and we take the supremum of f to be M. Using also the definition of the polynomials $P_n(\xi)$, the fact that $\int_{-1}^{1} Q_n(\xi)d\xi = 1$, and the triangle inequality; we have

$$\begin{aligned} |P_n(\xi) - f(\xi)| &= \left| \int_{-1}^{1} (f(\xi-\zeta) - f(\xi))Q_n(\zeta)d\zeta \right| \\ &\leq \int_{-1}^{1} |f(\xi-\zeta) - f(\xi)|Q_n(\zeta)d\zeta \\ &\leq 2M \int_{-1}^{-\delta} Q_n(\zeta)d\zeta + \frac{\epsilon}{2} \int_{-\delta}^{\delta} Q_n(\zeta)d\zeta + 2M \int_{\delta}^{1} Q_n(\zeta)d\zeta \\ &\leq 4M\sqrt{n}(1-\delta^2)^n + \frac{\epsilon}{2} < \epsilon, \end{aligned}$$

as long as we take n large enough, which shows that the sequence $P_n(\xi)$ converges uniformly to $f(\xi)$. □

It is possible to extend our version of the Weierstrass theorem to show that polynomials are uniformly dense in the set of continuous functions of *several* variables. It is also possible to extend the theorem further so that it applies to more general classes of functions than polynomials (the Stone–Weierstrass theorem). We shall see in the next section that the Weierstrass theorem is a major building block for the study of the approximation capabilities of neural networks.

2.3 Necessary and sufficient conditions for universal approximation

The result by Leshno and coworkers [75], which we present now, shows that—as long as the activation function is not a polynomial—feedforward neural networks can always approximate continuous functions in the L_∞ norm and L_p functions ($1 < p < \infty$) in the L_p norm. It is a powerful result because it imposes only weak constraints on the activation function of the neural network while giving *necessary and sufficient* conditions for universal approximation. Moreover, it deals with the problem of approximating continuous function in the L_∞ norm, which is stronger than the results of Cybenko and Hornik. The condition we impose on the activation function is that it must be piecewise continuous and locally essentially bounded. A *locally–essentially-bounded* function is bounded in the L_∞ norm on any compact subset of $\mathbb{R}$. We denote by $\mathcal{E}$ the set of all piecewise-continuous and locally–essentially-bounded functions of a real variable. Now, we are in a position to state Leshno's result.

Theorem 2.6 (Leshno et al.). *Let $\sigma \in \mathcal{E}$ and consider the L_∞ norm on any compact set $\mathcal{K}$ in $\mathbb{R}^d$. Then $\mathcal{N}_\sigma^d$ has the universal approximation property on $C(\mathcal{K})$ if and only if σ is not a polynomial almost everywhere.*

This theorem gives a result for the approximation of continuous functions; for discontinuous functions, we can consider the following proposition.

Proposition 2.7. *Let $\sigma \in \mathcal{E}$ and consider the L_p norm ($1 \leq p < \infty$) on any compact set $\mathcal{K}$ in $\mathbb{R}^d$. We have that $\mathcal{N}_\sigma^d$ has the universal approximation property on $L_p(\mathcal{K})$ if and only if σ is not a polynomial almost everywhere.*

We now prove Theorem 2.6. The proof is in two parts and seven steps. The first part (Lemmas 2.8, 2.9, and 2.10) shows that (1) neural networks with polynomial activation functions do not have the universal approximation property, (2) the approximation problem reduces to the purely one-dimensional case, and (3) neural networks with C^∞ activation functions can approximate any power of a single variable (hence, any polynomial) and can therefore approximate any continuous function by application of Weierstrass's theorem. The second part (Lemmas 2.11, 2.12, 2.13, and 2.14) shows that, in the case of neural networks whose activation function is piecewise continuous and locally bounded, (4) the neural network can approximate the convolution of its activation function with any smooth function; (5) the neural network can approximate any neural network with any such smooth convolution as activation function and if this convolution is not a polynomial, we have universal approximation by Lemma 2.10; (6) if universal approximation is impossible because any such convolution is always a polynomial, the degree of these polynomials is uniformly bounded; (7) if these convolutions are always polynomials of bounded degree, the activation function itself had to be a polynomial. All these steps together prove Theorem 2.6. Remember that the norm under consideration is the L_∞ norm and that we will always use it on some compact subset $\mathcal{K}$ of $\mathbb{R}^d$. We will always keep in mind that the convergence of a sequence of functions to a function $\lim_{n\to\infty} f_n = f$ means that $\lim_{n\to\infty} \sup_{x\in\mathcal{K}} |f_n(x) - f(x)| = 0$.

Lemma 2.8. *Single–hidden-layer networks with polynomial activation functions are not universal approximators of continuous real functions on $\mathbb{R}^d$.*

Proof. If the activation function is a polynomial of degree k, the function realized by the neural network is always a polynomial of degree k. Since the degree of these polynomials is limited to k, they cannot be dense in the set of continuous functions. □

Lemma 2.9. *If the single-input neural networks $\mathcal{N}_\sigma^1$ are universal approximators of continuous functions of a single variable, the multiple-input neural networks $\mathcal{N}_\sigma^d$ are universal approximators of continuous functions of several variables:*

$$\mathcal{N}_\sigma^1 \text{ is dense in } C(\mathbb{R}) \text{ implies that } \mathcal{N}_\sigma^d \text{ is dense in } C(\mathbb{R}^d).$$

We want to show that, if we can approximate any continuous function of a single variable, we can use these approximators as building blocks for the approximation of continuous real functions of several variables. We do this by looking at what we call ridges. Ridges are functions that change only along one direction and are constant along all others: $r(x) = g(ax)$ with $a \in \mathbb{R}^d$. Single–hidden-layer neural networks are a particular case of ridge, where the function is a translation and a dilation of a sigmoid.

Proof. We thus want to show that $\operatorname{span}\{g(ax) : a \in \mathbb{R}^d, g \in C(\mathbb{R})\}$ is dense in $C(\mathbb{R}^d)$. It is enough to prove that we can find a sequence of networks that converges to any monomial in d variables. Indeed, we can then find a sequence of networks converging to any polynomial by summation and we know that polynomials are dense in the set of continuous functions by the Weierstrass theorem. If we can show this property for any monomial in two variables, we can then inductively show it for any monomial in d variables. We can write these monomials as

$$Q_r^q(x_1, x_2) = x_1^{q-r} x_2^r.$$

We want to show that Q_r^q belongs to $\operatorname{span}\{f(a_1 x_1 + a_2 x_2) : a \in \mathbb{R}^2\}$. We want to show that we can write Q_r^q in the following particular form:

$$Q_r^q(x_1, x_2) = \sum_{i=0}^{q} c_i((i+1)x_1 + x_2)^q,$$

for some appropriate choice of c_i. After the change of variables $z = x_2/x_1$, we see that we want to find c_i such that

$$z^r = \sum_{i=0}^{q} c_i(i + 1 + z)^q.$$

We can check that the polynomials $(z + i + 1)^q$ are linearly independent by evaluating their derivatives at the origin. This independence implies that there must exist a unique solution to the previous equation. Hence, Q_r^q belongs to $\operatorname{span}\{f(a_1 x_1 + a_2 x_2) : a \in \mathbb{R}^2\}$. By induction, we can find a sequence of networks that converges to any monomial in d variables and by the Weierstrass theorem we prove our lemma. □

Lemma 2.10. *If $\sigma \in C^\infty(\mathbb{R})$ and it is not a polynomial, then $\mathcal{N}_\sigma^1$ has the universal approximation property in $C(\mathbb{R})$.*

First, recall that C^k functions are continuous functions whose k first derivatives are all continuous and that C^∞ functions have derivatives of all order that are all continuous. We say that C^∞ functions are smooth. This lemma

is an exciting step because its proof already gives us a simple universal approximation result. The proof is elementary and relies on showing that, for any monomial ξ^k, we can build a sequence of networks that converges to ξ^k. This independence implies that we can always find a sequence of networks converging to any polynomial. Since—by Weierstrass's theorem—the set of all polynomials is dense in the set of continuous functions of a single variable on any compact subset of $\mathbb{R}$, we have that there is a sequence of neural networks that converges to any continuous function on any compact set. Let us therefore show that these networks with smooth activation functions can approximate any monomial.

Proof. Let us consider a sequence $h_1, h_2, \ldots$ such that (1) for all j, $h_j \neq 0$, and (2) $\lim_{j\to\infty} h_j = 0$. For any h_j, we consider the following network with two hidden neurons:

$$N_j^{(1)}(\xi; a, a_0) = \frac{1}{h_j}\sigma((a + h_j)\xi + a_0) - \frac{1}{h_j}\sigma(a\xi + a_0).$$

We have that $\lim_{j\to\infty} N_j^{(1)}(\xi; a, a_0) = \frac{d}{da}\sigma(a\xi + a_0) = \xi\sigma'(a\xi + a_0)$. In particular, for $a = 0$, $\lim_{j\to\infty} N_j^{(1)}(\xi; a, a_0) = \xi\sigma'(a_0)$. Now, consider the following network with four hidden neurons:

$$N_j^{(2)}(\xi; a, a_0) = \frac{1}{h_j}(N_j^{(1)}(\xi; a + h_j, a_0 + +h_j) - N_j^{(1)}(\xi; a, a_0)).$$

We have that $\lim_{j\to\infty} N_j^{(2)}(\xi; a, a_0) = \frac{d}{da}\xi\sigma'(a\xi + a_0) = \xi^2\sigma''(a\xi + a_0)$. In particular, for $a = 0$, $\lim_{j\to\infty} N_j^{(2)}(\xi; a, a_0) = \xi^2\sigma''(a_0)$. Recursively, defining

$$N_j^{(k)}(\xi; a, a_0) = \frac{1}{h_j}(N_j^{(k-1)}(\xi; a + h_j, a_0) - N_j^{(k-1)}(\xi; a, a_0))$$

and setting $a = 0$, we have $\lim_{j\to\infty} N_j^{(k)}(\xi; a, a_0) = \xi^k\sigma^{(k)}(a_0)$ (where $\sigma^{(k)}$ denotes the kth derivative of σ). Since σ is C^∞, its derivatives of all order exist and since σ is not a polynomial, its derivatives never become identically zero. Hence, we can always find an a_0 such that $\sigma^{(k)}(a_0) \neq 0$. We therefore see that we can always find a sequence of single–hidden-layer neural networks that converges to the monomial ξ^k. As argued before, this convergence implies that we can always find a sequence of neural networks that converges to any polynomial, hence we can find a sequence that converges to any continuous function. Lemmas 2.8, 2.9, and 2.10 taken together show that neural networks with a smooth activation function that is nonpolynomial can universally approximate smooth functions of several variables. □

In this second part, we deal with activation function that are not smooth anymore but locally essentially bounded. In this lemma and in the next one,

we try to recover the smoothness properties necessary for the application of the previous lemma by using a convolution. Indeed, the operation of convolution by a smooth function s has the property of smoothing out discontinuities and, for this reason, the function $s(\xi)$ is called a mollifier [41].

Lemma 2.11. *For any $\sigma \in \mathcal{E}$ and any C^∞ function $s : \mathbb{R} \to \mathbb{R}$ with compact support ($s \in C_c^\infty$: there exists an interval $[-a, a]$ such that $s(\xi) = 0$ for all $\xi \notin [-a, a]$), there exists a sequence of networks that uniformly converges to the convolution $\sigma * s$.*

Proof. The convolution is defined by

$$(\sigma * s)(\xi) = \int_{-a}^{a} \sigma(\xi - \zeta) s(\zeta) d\zeta. \tag{2.1}$$

It is well-defined since s has compact support and σ is bounded on any interval, which guarantees that the integral is always finite. We look at the following networks:

$$M_m(\xi) = \sum_{i=1}^{m} s(\zeta_i) \Delta\zeta_i \sigma(\xi - \zeta_i) = \sum_{i=1}^{m} \sigma(\xi - \zeta_i) s(\zeta_i) \Delta\zeta_i,$$

where $\Delta\zeta_i = 2a/m$ and $\zeta_i = -a + 2ia/m$. What we do is approximate the integral (2.1) by the trapezoidal rule on the interval $[-a, a]$ (recall that s has compact support, hence there exists such an a) and divide the interval into an increasing number of subintervals. Now, this trapezoidal rule converges uniformly to the integral. Indeed, σ is locally essentially bounded and s is C^∞, hence both are essentially bounded on $[-a, a]$. Moreover, σ is piecewise continuous, hence it is uniformly continuous except on the union of a finite number of arbitrary small intervals. Thus, the trapezoidal rule converges uniformly on the complement of this union and we can make the contribution of the integral over the union arbitrarily small. Therefore, $\lim_{m\to\infty} M_m(\xi) = (\sigma * s)(\xi)$. □

Lemma 2.12. *For any $\sigma \in \mathcal{E}$, if we can find $s \in C^\infty$ such that $\sigma * s$ is not a polynomial, then $\mathcal{N}_\sigma^1$ has the universal approximation property in $C(\mathbb{R})$ under the L_∞ norm on any interval.*

Proof. We can always find a sequence of networks that converges uniformly to $(\sigma * s)(\xi)$; thus, we can always find a sequence of networks that converges to $(\sigma * s)(a\xi + a_0)$. But now, since s is C^∞ and has compact support, $\sigma * s$ is C^∞ because of the smoothing property of the convolution [3]. Therefore, we can use Lemma 2.10 and $\mathcal{N}_\sigma^1$ is dense in $C(\mathbb{R})$ if $\sigma * s$ is not a polynomial. □

We are thus left with studying the conditions on σ under which $\sigma * s$ is a polynomial for any s.

Lemma 2.13. *If, for all $s \in C_c^\infty$, $(\sigma * s(\xi))$ is a polynomial; then, the degree of these polynomials is uniformly bounded (there exists an M such that, for all s, $\text{degree}(\sigma * s) < M$).*

This lemma is technical and only preparatory for the last step. It requires the use of Fréchet spaces and Baire's category theorem, which would be cumbersome to introduce here. We refer the reader to [75] for its proof. Now, we are ready to complete the proof by showing that the convolution is always a polynomial (assuming that there is a uniform bound on the degree) only if the activation function itself is a polynomial.

Lemma 2.14. *Let $\sigma \in \mathcal{E}$. If $\sigma * s$ is a polynomial of degree inferior to M for all $s \in C_c^\infty$, then σ is a polynomial almost everywhere.*

Proof. Since $(\sigma * s)(\xi)$ is always a polynomial of degree at most m, the $(m+1)$th derivative of this convolution is identically zero:

$$\frac{d^{(m+1)}}{d\xi^{(m+1)}}(\sigma * s)(\xi) = \int_{\mathbb{R}} \sigma^{(m+1)}(\xi - \zeta) s(\zeta) d\zeta = 0, \quad \text{for all } s \in C_c^\infty.$$

Since this relation must hold for any $s \in C_c^\infty$, the $(m+1)$th derivative of σ must be identically zero. Therefore, σ is a polynomial. □

Proof of Theorem 2.6. Lemmas 2.8 to 2.14 prove together the theorem. □

An important conclusion is that the universal approximation properties of neural networks have little magical to them. What we have done is stated Weierstrass's theorem, which says that polynomials can approximate continuous functions, and shown that neural networks can approximate polynomials; hence, they can also approximate continuous functions. The Weierstrass theorem is really the main actor of this play and we have not shown anywhere that neural networks are more powerful than polynomials. This begs the question: What could neural networks be good for? Practice does give a real advantage to neural networks for problems in high dimensions, where polynomial series are unusable. The problem with polynomial series is that even for moderately high dimensions, the number of monomials needed for approximation blows up exponentially, which makes the estimation of the coefficients ill-conditioned. Although neural networks rely on gradient descent, which implies that the estimation of the parameters is still a difficult problem, they are a workable alternative. Neural networks seem to provide more compact representations than polynomials. To confirm this possibility, we have to study the rates at which the approximation error of neural networks decreases as we increase the number of neurons (or parameters) in the network.

2.4 Rates of convergence

We present now results of Barron and Jones [13, 65] that show that (for some large class of functions from $\mathbb{R}^d$ to $\mathbb{R}$ defined by a condition on the Fourier transform) single–hidden-layer networks with n neurons can approximate functions from this class with an L_2 error of the order of $1/\sqrt{n}$. The activation function of these networks must be measurable, bounded, and sigmoidal. The approximation rate is independent of the dimension of the input space; although we will have to qualify this statement carefully later. The performance of neural networks is here in contrast with the performance of series expansions, which cannot achieve an error smaller than $O(1/n^{1/d})$ for the same class of functions; although we will also have to qualify this statement later.

The functions under investigation are defined by properties of their Fourier transform. Let $f(x)$ defined on $\mathbb{R}^d$ be a function having a Fourier representation:

$$f(x) = \int_{\mathbb{R}^d} e^{i\omega x} F(\omega) d\omega.$$

This is a multidimensional Fourier transform, so ω is a vector in $\mathbb{R}^d$ and $F(\omega)$ is a d-dimensional *complex-valued* function from $\mathbb{R}^d$ to $\mathbb{C}$. If $\omega F(\omega)$ is integrable, we define the following quantity, which measures the oscillations of the function through the first moment of the absolute value of its Fourier representation:

$$C_f = \int_{\mathbb{R}^d} ||\omega|| \cdot |F(\omega)| d\omega.$$

Now, we define Γ_C to be the set of functions having a Fourier representation for which C_f exists and $C_f \leq C$. We can also reinterpret the finiteness condition on C_f as the requirement that the gradient of the function f admits a Fourier transform. We define the error between a network and a function as the L_2 norm restricted to a ball $\mathcal{B}_r$ of radius r and weighted according to a chosen probability distribution $p(x)$. Barron [13] gives the following bound on the error of the neural network.

Theorem 2.15 (Barron). *If f is such that C_f is finite and the activation function of the neural network is measurable, bounded, and sigmoidal. Then, for any positive integer n, there exists a single–hidden-layer network*

$$N_n(x) = \sum_{i=1}^{n} b_i \sigma(a_i x + a_0) \quad \textit{with } b_i, b_0, a_0 \in \mathbb{R}, \textit{ and } a_i \in \mathbb{R}^d,$$

such that

$$E(N_n, f) = \Big(\int_{\mathcal{B}_r} (f(x) - N_n(x))^2 p(x) dx \Big)^{1/2} \leq \frac{2rC_f}{\sqrt{n}}.$$

This bound is fairly tight as Barron showed [12] that (in dimension two or more) there is always a function g in the set Γ_C for which

$$E(N_n, g) \geq O(1/n^{(1/2+1/d)}).$$

We see that these two bounds converge to each other for n large.

The central tool for the proof of the preceding proposition is the following lemma that studies how fast convex combinations of n elements of a set approach points in the closure of the convex hull of this set. We first recall that a *Hilbert space* is a (possibly infinite-dimensional) vector space endowed with a scalar product and where we choose the norm as the square root of the scalar product of a vector with itself. We also recall that the *closure* of a set is the set of the limits of all the sequences of elements of the original set.

Definition 2.5. The *convex hull* $\mathcal{H}$ (see Fig. 2.1) of a set $\mathcal{G}$ is the set of all convex combinations of elements of $\mathcal{G}$ (a convex combination is a combination for which the sum of the coefficients equals one):

$$\mathcal{H} = \left\{ h_n(x) : \begin{array}{l} \exists n > 0, g_1, \ldots, g_n \in \mathcal{G} \text{ such that } h_n(x) = \sum_{i=1}^n \alpha_i g_i(x) \\ \text{for some } \alpha_1, \ldots, \alpha_n > 0 \text{ such that } \sum_{i=1}^n \alpha_i = 1 \end{array} \right\}.$$

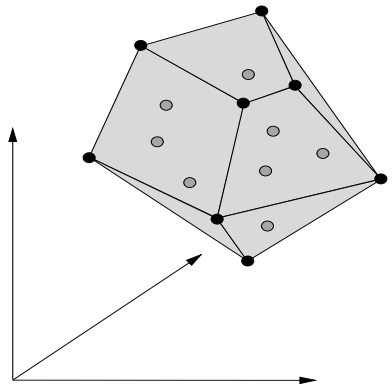


Figure 2.1: Convex hull of a set. The convex hull is the smallest set (here, the shaded-grey volume), containing all the points of the original set that is closed under convex combination. (Black dots determine the surface of the convex hull while grey dots are those inside the hull.)

Lemma 2.16. *Let $\mathcal{G}$ be a set of functions in a Hilbert space, such that $||g|| < b$ for all $g \in \mathcal{G}$. Let $\bar{h}$ be a point in $\bar{\mathcal{H}}$ the closure of the convex hull of $\mathcal{G}$. Then, for every $c > b^2 - ||\bar{h}||^2$, there is a convex combination h_n of elements of $\mathcal{G}$ such that $||\bar{h} - h_n|| \leq c/n$.*

We refer the reader to Barron [13] for the proofs of the lemma and of the theorem. The proof of the lemma relies on taking a random sequence of points in $\mathcal{G}$ and showing that we can optimize the coefficients of the convex combination so that the error always remain under the upper bound c/n. The proof of the theorem relies on applying the lemma to a set $\mathcal{G}_\sigma$ of sigmoidal functions:

$$\mathcal{G}_\sigma = \left\{ b\sigma(ax + a_0) \text{ with } |b| < 2C; b \in \mathbb{R}, a \in \mathbb{R}^d, a_0 \in \mathbb{R} \right\}$$

by showing that the *convex* closure of this set contains the set of functions Γ_C that we want to approximate. Barron proves this fact by showing that (1) the convex closure of the set of sigmoidal networks (with output weights of magnitude bounded by $2C$) contains the set of networks with step activation functions (with a similar restriction on the weights), (2) the convex closure of this set of step networks contains the set of networks with cosine activation function (with a similar restriction on the weights), (3) the convex closure of the set of cosine networks contains the functions from Γ_C for which $f(0) = 0$, and (4) the convex closure of the set of sigmoidal networks contains the functions from Γ_C for which $f(0) = 0$ (this restriction to $f(0) = 0$ does not cause any loss of generality since we can solve the problem by setting the output threshold to the true value of the function at zero). The proofs of universal approximation by neural networks are of no help to prove the previous relations since we limit ourselves to convex combinations. The need for cosine networks stems from the fact that the condition defining the Γ_C is a condition on the Fourier representation and that cosines are easy to use for approximation in Fourier space.

2.5 Optimal nonlinear approximation

DeVore [31] has given an important result on rates of approximation that are achievable by nonlinear approximation. Nonlinear approximation has a specific meaning in this case: The set of all approximating functions with n terms forms a nonlinear surface in function space; for example, the set of all single–hidden-layer perceptrons with n hidden neurons does not form a linear space while the set of all polynomials of degree less than or equal to n does. If we want to approximate functions from some spaces of functions defined by a condition on the partial derivatives of these functions, the result of DeVore states the following: If the approximation method is such that it approximates two functions that are close to each other (in some quasi-norm) by approximators whose parameters (weights) are close to each other, then it cannot achieve a rate of convergence better than that of splines with free knots.

The functions we consider are functions in some Sobolev space W_p^r [3]. Because the apparatus from functional analysis needed to define Sobolev spaces

is heavy, we have to remain sketchy about the definition of the objects under consideration. This Sobolev space is the subset of L_p containing the functions that have r weak derivatives, which are themselves in L_p. There exists a canonical seminorm $|\cdot|$ associated with this Sobolev space W_p^r (a seminorm is always positive but can be zero even for a nonzero vector). We define $\mathcal{K}$ to be the set of functions from W_p^r whose seminorm is less than or equal to one: $\mathcal{K} = \{f \in W_p^r : |f| \leq 1\}$.

We require that our approximation method be a continuous selection. Let $\mathcal{M}_n$ be a mapping from $\mathbb{R}^n$ into L_p—this mapping is our approximator, which associates the *function* $\mathcal{M}_n(a)$ to the set of parameters a. Now, when we take a function f we want to approximate, the map $\mathcal{A}_n : L_p \to \mathbb{R}^n$ gives us the parameters of the approximator with n parameters we associate to this function. The function that this approximator realizes is then given by $\mathcal{M}_n(\mathcal{A}_n(f))$. We say that $\mathcal{A}_n$ is a *continuous selection* if it is a continuous map from L_p to $\mathbb{R}^n$ (with respect to the L_p norm). Therefore, the set of parameters, which the continuous selection associates to two functions that are close to each other, are themselves close to each other.

Theorem 2.17 (DeVore et al.). *Let $\mathcal{K} = \{f \in W_p^r : |f| \leq 1\} \subset L_p : \mathbb{R}^d \to \mathbb{R}$ be the set of functions of d variables we want to approximate. For any approximator $\mathcal{M}_n$ and any continuous selection $\mathcal{A}_n$, we have for any $q \geq p$ that*

$$\max_{f \in \mathcal{K}} \|f - \mathcal{M}_n(\mathcal{A}_n(f))\|_q \geq C(r) n^{-r/d},$$

where $||.||_q$ is the L_q norm and $C(r)$ is a multiplicative constant dependent only on r.

This theorem means that whatever class of approximators we use, as long as our method of approximation is a continuous selection, we cannot achieve a rate of approximation faster than $n^{-r/d}$ for d-dimensional functions in L_p whose partial derivatives of rth order are all in L_p. DeVore shows that approximation by piecewise polynomials (splines with free knots) can actually achieve such a rate of convergence (they find an upper bound on the error with the same power of n).

The ratio r/d is an important indicator of smoothness as it shows that we can balance dimension and smoothness. If we restrict the choice of functions we want to approximate to the functions for which the number of L_p partial derivatives is equal to half of the number of variables, we can achieve a rate of convergence of $n^{-1/2}$ in the same way as in Barron's result.

2.6 The curse of dimensionality

It is important to discuss the exact meaning of these results. The main claim we want to examine is summarized in the following headline from the Neural Times of May 1, 1993: "Neural Networks Beat The Curse of Dimensionality"! As a reminder, the curse of dimensionality implies that, for many estimation problems, the amount of data needed to achieve a given precision in the estimation grows exponentially with the number of dimensions of the problem. Indeed, the rate of approximation from Theorem 2.15 appears to be independent of the dimension d of the input space. But this statement is somewhat misleading because the constant C_f in the approximation rate $E(N_n, f) \leq \frac{2rC_f}{\sqrt{n}}$ is a d-dimensional integral, which means that C_f can depend on d. Or, put the other way round, if we look at the set Γ_C of functions for which $C_f \leq C$, it can in some sense get smaller and smaller as d increases. The condition that $C_f \leq C$ may require that the functions gets smoother and smoother as the dimension increases. Moreover, the statement depends on limiting ourselves to a given class of functions Γ_C. Indeed, if we look at other classes of functions, the independence of the rate of convergence on the dimension d of the output space disappears as showed by the approximation rates developed in [83, 86, 71]. But the class of functions Γ_C is relatively large and does contain interesting functions [13]. Another thing to remark is that we have achieved this result for all the networks that have any bounded measurable function $\sigma(x)$ with $\lim_{x\to-\infty} \sigma(x) = 0$ and $\lim_{x\to\infty} \sigma(x) = 1$ for activation function (the exact value of the limits is irrelevant as long as they exist and are distinct). The generality of this result means that we have in no way used any continuity assumption for the activation function, not to mention monotonicity. The networks for which this result are valid are therefore much more general than just the networks with the hyperbolic tangent or logistic sigmoid activation function that are most common in the field of neural networks. There are even similar results for radial basis function networks where the centers and the radii of the basis functions are adaptive [46]. These results seem therefore to hold for large classes of neural networks.

DeVore's result reinforces our argument that, to obtain dimension independent rates of approximation, we have to require more and more smoothness as the dimension of the input space increases. For functions with partial derivatives of order r, the ratio r/d seems to be a reliable indicator of the difficulty of approximating the function. Indeed, Mhaskar [86] showed that, for spaces of functions similar to these considered by DeVore, single–hidden-layer sigmoidal networks also achieve the lower bound (just as piecewise polynomials did). The difficulty with DeVore's result is that it is not clear that it holds for sigmoidal networks. For one thing, the result applies to functions in L_p approximated by approximators in L_p while the sigmoid is never in L_p except for $p = \infty$. An even more important point is that it is not clear whether approximation with

sigmoidal networks gives rise to a continuous selection. Indeed, algorithms for the estimation of the parameters of a sigmoidal network are based on gradient descent. We can represent these algorithms by a nonlinear dynamical system depending on the function that we want to approximate. It is not guaranteed that such dynamical systems will be structurally stable, meaning that two systems that are close would not necessarily have equilibria that are close. Even if we try to get rid of the algorithm by considering the best approximation possible for each function, the problem remains. Kainen and Kürková [66] have just shown that the best approximation does not provide a continuous selection for sigmoidal networks. It seems, however, that the general rule of trading smoothness for dimension still applies to neural networks, even if an alternative to DeVore's result is necessary.

To contrast these results for single–hidden-layer neural networks, we now look at series expansion. For expansions with respect to a fixed basis, we obtain a significantly worse bound for the approximation of functions in Γ_C. Suppose that $H = \{h_1, h_2, \dots\}$ is a basis of Γ_C ($\text{span}(H) \supset \Gamma_C$) and that we look at series expansions $S_n(x) = \sum_{i=1}^{n} b_i h_i(x)$. We call H_n the subset containing the first n basis functions $H_n = \{h_1, \dots, h_n\}$. For the error, we take the L_2 norm again, but we restrict ourselves to the case where we define the functions on the unit hypercube and the probability distribution is uniform ($p(x) = 1$). We thus compute the error for the series expansion as follows:

$$E(S_n, f) = \min_{b_1,\dots,b_n} \left(\int_{[0,1]^d} \left(f(x) - S_n(x)\right)^2 dx \right)^{1/2}.$$

We need to look at the worst case for this error to find a lower bound for the rate of approximation of series expansions. Barron gives the following theorem.

Theorem 2.18. *For any basis H of Γ_C, we have that*

$$\sup_{f \in \Gamma_C} E(S_n, f) \geq \kappa \frac{C}{d} \left(\frac{1}{n}\right)^{1/d},$$

where κ is a constant independent of the choice H of basis functions and we know that $\kappa > 1/(8\pi e^{\pi - 1})$.

Examples of series expansion (all possibly multidimensional) include polynomial series, Chebycheff polynomials, Fourier series, and so on. When comparing the rates of convergence of Theorems 2.15 and 2.18, we see that fixed basis expansions perform much worse than single–hidden-layer networks. It seems that the possibility of adapting the basis functions themselves (here by an affine transformation of the inputs) allows a more compact form for the approximation of functions in a fairly general setting according to Barron's and DeVore's result.

A last remark is that there might be better ways to use fixed basis functions than series expansions. In series expansion, the choice of basis functions is not only fixed but we are forced to use them in an order given prior to examining our function. If we perform badly on one function, we could decide to change the order of the basis functions to suit the needs of this function, but then some other function will become difficult to approximate for the basis in this new order (this is a certainty given the bound presented previously). What we are not allowed to do is to look first at our function, then reorder our basis functions, then do approximation with this new basis. This procedure might be much more efficient. To convince ourselves of this fact, we could take the case of radial basis functions positioned at the nodes of a finite grid. We could do series expansion by using the nodes starting at the center of the grid and including more and more nodes by spiraling away from the center of the grid, but we would get a method of approximation for which the situation described before holds. In contrast, if we were to use a clustering method to select only the nodes that are relevant to the particular function we want to approximate, we would be in the situation of Barron's bound. Techniques where we do approximation by selection of a subset of a fixed basis are called sparse approximation. Their use had been limited so far because of the lack of efficient algorithms for the selection of the basis functions. However, recent work [45, 143] indicates that it is possible to formulate this selection problem as a quadratic optimization problem. It seems that such methods could achieve the same rates of convergence as sigmoidal networks or that we could use them to estimate the parameters of sigmoidal networks. We expect that these methods will cause a major shake-up in the field of neural networks.

Our main conclusion in this discussion is that methods for which we can adaptively change the basis for approximation provide in general more compact representations than those where the basis for approximation is fixed a priori.

2.7 Uniqueness of the weights of static neural networks

We have been looking at the approximation capabilities of single–hidden-layer neural networks. When we want to find a particular solution to an approximation problem, the following important question arises: Can two different single–hidden-layer networks produce the same function? If this situation occurs, we will have to take some precautions when trying to solve approximation problems. In fact, it is easy to show that this situation is possible, as we will see shortly.

Again, we consider only the case of networks with a single output. (You can follow the reasoning at the hand of Figure 2.2.) This case generalizes to networks with multiple outputs directly. If we take a neural network and

permute two of its hidden nodes, we get another network that realizes the same mapping. If the activation function is odd and we change the sign of all the weights coming into a neuron and change the sign of all the weights going out from this neuron, we also do not change anything. More trivially, if the output weight of a neuron is zero, it does not have any contribution and we could remove it. If the input weights (but not necessarily the bias) of a neuron are zero, we could replace the neuron by an output bias. If the input weights of two neurons are equal or opposite (and the function is odd), we could replace them by a single neuron.

The question is, once we have simplified the networks to take these trivial cases into account: Are two equivalent neural networks necessarily related by one of these simple symmetries (permutation of the nodes and change of the sign of all the weights of one neuron)? The answer is affirmative. Sussman [135] showed this for neural networks with the hyperbolic tangent as activation function. This result was later extended byAlbertini, Sontag, and Maillot [8], who gave a general condition on the activation function, which arises from the method of residues.

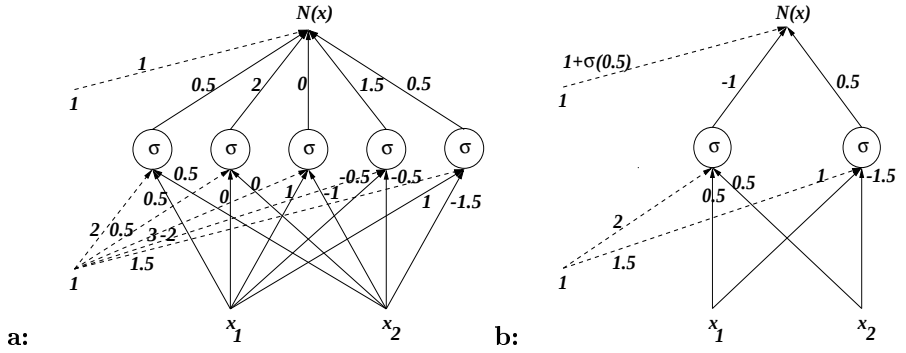


Figure 2.2: Reducibility of neural networks. Solid arrows denote the weights a_{ij} and b_i while dashed arrows denote the biases a_0 and b_0. **a:** A reducible five-neuron network. The first and the fourth neurons have opposite weights and biases; we can sum them up after a sign reversal. The second neuron has zero weights; we can replace it by a contribution to the output bias. The third neuron has a zero output weight; we can remove it because it does not make any contribution. **b:** An irreducible two-neuron network. By performing the reduction of the five-neuron network, we obtain this irreducible network.

2.7.1 The independence property

The idea of the proof is based on the linear independence of a set of functions. If two networks are equivalent, then their difference is a network for which the

output is always zero. So, we have a linear combination of functions of the form $\sigma(ax + a_0)$, which is identically zero. If we can show that any finite set of such functions (with different weights) is linearly independent, the only way to make this linear combination equal to zero is to set all the coefficients equal to zero. This property means that the neurons of the two networks are identical or that they have neurons that cancel each other out within themselves. This property of independence does not hold for arbitrary activation functions in general. For example, if the activation function is periodic (there exists a $T \in \mathbb{R}$ such that $\sigma(\xi + T) = \sigma(\xi)$ for all $\xi \in \mathbb{R}$), the difference of two neurons can be made identically zero by a translation of T. Similarly, if $\sigma(\xi) = e^{\xi}$, we have $e^{a_0}\sigma(a\xi) - \sigma(a\xi + a_0) \equiv 0$.

Definition 2.6. The activation σ satisfies the *independence property* if, for any number n of single-input neurons $\sigma(a_i\xi + a_{0i})$, $(\xi \in \mathbb{R})$ with weights $\{(a_1, a_{01}), (a_2, a_{02}), \ldots, (a_n, a_{0n})\}$ such that $a_i, a_{0i} \in \mathbb{R}$ for all i, $a_i \neq 0$ for all i, and

$$(a_i, a_{0i}) \neq \pm(a_j, a_{0j}), \quad \text{for all } i \neq j,$$

we have

$$b_0 + \sum_{i=1}^{n} b_i\sigma(a_i\xi + a_{0i}) = 0 \text{ for all } \xi \in \mathbb{R} \text{ implies } b_0 = b_1 = \cdots = b_n = 0$$

—that is, the functions $\sigma(a_1\xi + a_{01}), \sigma(a_2\xi + a_{02}), \ldots, \sigma(a_n\xi + a_{0n})$, and 1 are linearly independent.

We now return to networks with multiple inputs, which we write as

$$N(x) = b_0 + \textstyle\sum_{i=1}^{n} b_i\sigma(a_i x + a_{0i}),$$

with $x \in \mathbb{R}^d, b_0 \in R$, and $a_i \in \mathbb{R}^d, a_{0i}, b_i \in \mathbb{R}$ for all $i = 1, \ldots, n$. We need to deal with the case where some neurons add up or it is possible to remove them.

Definition 2.7. A single–hidden-layer network is *irreducible* if none of the following applies to the weights of the network:

1. There is an $i \in \{1, \ldots, n\}$ such that $b_i = 0$. We can then remove the corresponding neuron.
2. There is an $i \in \{1, \ldots, n\}$ such that $a_i = 0, a_i \in \mathbb{R}^d$. The corresponding neuron makes a constant contribution and we can then replace the output bias b_0 by $b_0 + \sigma(a_{0i})$ and remove neuron i.
3. There are $i, j \in \{1, \ldots, n\}$ such that $a_i = a_j$ and $a_{0i} = a_{0j}$. The two neurons have the same weights. We can therefore remove neuron j and replace the output weight b_i of neuron i by $b_i + b_j$.

4. There are $i, j \in \{1, \ldots, n\}$ such that $a_i = -a_j$ and $a_{0i} = -a_{0j}$. The two neurons have opposite weights. Since we have assumed that σ is odd, we can change the sign of the input and output weights of neuron j without changing anything. We can therefore remove neuron j and replace the output weight b_i of neuron i by $b_i - b_j$.

When a network is reducible, we do reduce it to get an irreducible network.

2.7.2 Irreducibility and sign–permutation equivalence

We can now take the symmetries of the network into account. If we can map two networks onto each other by a permutation of the nodes or a change of sign of the input and output weights of some of the nodes, the two networks will realize the same function. We consider the set $\mathcal{P}$ of the matrices representing all permutations of n elements and the set $\mathcal{S}$ of the matrices with 1 or -1 on the diagonal. To denote matrices built up from vectors, we use the following convention: $[v_1; v_2]$ denotes the matrix formed by stacking up the vectors v_1 and v_2 (we have $[v_1; v_2] = [v_1^T v_2^T]^T$ if the notation $[w_1 w_2]$ denotes the matrix created by putting the vectors side by side). We shall later use the same notation for matrices: $\left[\begin{array}{c} A; \\ B \end{array}\right] = \left[\begin{array}{cc} A^T & B^T \end{array}\right]^T$. Using this convention, we define the matrices of the weight of a network as $A = [(a_1, a_{01}); \ldots; (a_n, a_{0n})]$ and $B = (b_1, \ldots, b_n)$. The main result is the following theorem, after we define an equivalence relation between networks.

Definition 2.8. Two irreducible networks with weights A, B, b_0 and A', B', b_0' are *sign–permutation equivalent* if $b_0 = b_0'$ and there exist $P \in \mathcal{P}$ and $S \in \mathcal{S}$ such that $A = SPA'$ and $B = SPB'$.

Theorem 2.19 (Albertini et al.). *Let the activation function σ be odd and satisfy the independence property. If two networks $N(x)$ and $N'(x)$ are irreducible and their outputs are equal everywhere ($N(x) = N'(x), \forall x \in \mathbb{R}^d$), then they are sign–permutation equivalent.*

Proof. Suppose $N(x)$ and $N'(x)$ are equal everywhere, thus

$$b_0 + \sum_{i=1}^{n} b_i \sigma(a_i x + a_{0i}) = b_0' + \sum_{i=1}^{n} b_i' \sigma(a_i' x + a_{0i}') \text{ for all } x \in \mathbb{R}^d.$$

We want to reduce the problem to the one-dimensional case since the independence property applies to single-input sigmoids. We achieve this reduction by restricting our attention to the behavior of the network along the line $x = \lambda v$, where v is a vector in $\mathbb{R}^d$. If we now set $\alpha_i = a_i v \in \mathbb{R}$ and $\alpha_i' = a_i' v \in \mathbb{R}$, we have

$$(b_0 - b_0') + \sum_{i=1}^{n} b_i \sigma(\alpha_i \lambda + a_{0i}) - \sum_{i=1}^{n} b_i' \sigma(\alpha_i' \lambda + a_{0i}') = 0 \text{ for all } \lambda \in \mathbb{R}. \quad (2.2)$$

Note that we now have again a one-dimensional network, so we can use the independence property. However, this property requires that the network be irreducible and we have to make sure that the weights α_i and α'_i are never zero and never equal up to a change of sign. We therefore choose v such that

1. For all $i = 1, \dots, n$, we have $\alpha_i \neq 0$ and $\alpha'_i \neq 0$.
2. For all $i \neq j$, we have $(\alpha_i, a_{0i}) \neq \pm(\alpha_j, a_{0j})$, $(\alpha'_i, a'_{0i}) \neq \pm(\alpha'_j, a'_{0j})$.

It is always possible to find such a vector since the aforementioned constraints form a linear system of equations and we look for a vector that does not make this system equal to zero. The only possible way for the application of the independence property to fail now is if there exist i and j such that $(\alpha_i, a_{0i}) = \pm(\alpha'_j, a'_{0j})$. We cannot avoid this situation by the choice of v since the input weights a_i of a neuron of the first network could be equal or opposite to the weights a'_j of a neuron of the second network (this situation cannot occur within the network itself because of its irreducibility). If such a situation occurs, we reduce the one-dimensional network represented by Equation 2.2 by flipping the signs to make neurons with opposite weights have equal weights and by adding the neurons with equal weights. Once we have reduced the equation, we can apply the independence property. Thus, the coefficients of the reduced networks need to be zero, which means that the weights of the two networks must be equal, except maybe for a change of sign of all the weights of a neuron. This equality proves our theorem. □

Now, we examine the possible choices of activation function σ if we want to satisfy the independence property. For this, we need to use elementary tools from the theory of functions of a complex variable. Albertini showed the following.

Proposition 2.20. *Let the activation function σ be real and analytic. If we can find a complex function $\rho : \mathcal{D} \subset \mathbb{C} \to \mathbb{C}$ defined on $\mathcal{D}$ such that*

$$\mathcal{D} = \{z : -\beta \leq |\Im\mathrm{m}(z)| \leq \beta\} \setminus \{\alpha + j\beta, \alpha - j\beta\},$$

where $\alpha + j\beta$ and $\alpha - j\beta$ are singularities ($\lim_{z \to \alpha \pm j\beta} |\rho(z)| = \infty$), and such that $\rho(x) = \sigma(x)$ for all $x \in \mathbb{R}$; then σ has the independence property.

Proof. Suppose we do not have linear independence, then there exist n and $b_0, \dots, b_n$ all different from zero such that

$$b_0 + b_1\rho(a_1 z + a_{01}) + \cdots + b_n\rho(a_n z + a_{0n}) = 0 \text{ for all } z \in \mathcal{D} \in \mathbb{C},$$

where n is as small as possible. This is in particular the case on the real axis. Since we have linear dependence, we have $b_i \neq 0$ for all i while $a_i \neq 0$ for all i and $(a_i, a_{0i}) \neq \pm(a_j, a_{0j})$ for all i, j. Suppose, without loss of generality by a

reordering of the terms, that $|a_1| \geq |a_i|$ for all $i \neq 1$ and define $y = a_1 z + a_{01}$. We have

$$b_0 + b_1\rho(y) + b_2\rho(a_2' y + a_{02}') + \cdots + b_n\rho(a_n' y + a_{0n}') = 0.$$

We know that $(a_i', a_{0i}') \neq \pm(1, 0)$ since $(a_i, a_{0i}) \neq \pm(a_j, a_{0j})$ for all i, j. We further know that $\rho(a_i'(\alpha + j\beta) + a_{0i}')$ is not a singularity since $\alpha + j\beta$ and $\alpha - j\beta$ are the only two singularities in the domain $\mathcal{D}$. Therefore,

$$\frac{b_0}{\rho(y)} + b_1 + b_2\frac{\rho(a_2' y + a_{02}')}{\rho(y)} + \cdots + b_n\frac{\rho(a_n' y + a_{0n}')}{\rho(y)} = 0$$

when $\rho(y) \neq 0$. But, in a neighborhood of $y_0 = \alpha + j\beta$, $\rho(y)$ is different from zero and if we take the limit when y tends to y_0, the quotients $\rho(a_i' y + a_{0i}')/\rho(y)$ all tend to zero (since y_0 is a singularity); we get

$$\lim_{y \to y_0} \frac{b_0}{\rho(y)} + b_1 + b_2\frac{\rho(a_2' y + a_{02}')}{\rho(y)} + \cdots + b_n\frac{\rho(a_n' y + a_{0n}')}{\rho(y)} = 0 = b_1.$$

This equality means that $b_1 = 0$, which is in contradiction with our assumption of linear dependence; hence σ must satisfy the independence property, which proves Proposition 2.20. □

2.8 Conclusion

We conclude here this review of the properties of feedforward sigmoidal networks as function approximators. We started by recalling the first main results on universal approximation by neural networks, which were proved by Cybenko [28] and Hornik, Stinchcombe, and White [62] among others. These results created much interest for neural networks. The versatility of neural networks was often put in contrast with the difficulty of using polynomial expansions for high-dimensional problems. Yet, from a theoretical point of view, we saw via the classical Weierstrass theorem that polynomials could already universally approximate continuous functions. Furthermore, we studied the approximation properties of neural networks by showing that we could use neural networks to approximate polynomials, which meant that they had at least the same approximating power as polynomials.

This approach was powerful and led to the condition that neural networks that do not have a polynomial activation function could universally approximate continuous function [75]; moreover, this condition was both necessary and sufficient. This result did have the negative side that it treated both neural networks and polynomials on an equal footing. We therefore went on to study whether neural networks could provide more compact representations than polynomial series. The answer was positive—at least for some interesting

classes of functions—as Barron's result [13] showed. However, this result was limited by the class of functions to which it applied and we saw that it was much harder to give general conditions under which neural networks are more powerful than polynomials. Yet the main conclusion remains that we gain a decisive advantage by choosing *adaptively* the basis used for approximation.

Finally, we also covered another fundamental result about whether there exist neural networks that represent an identical function. The answer was unfortunately positive, which means that we will have to treat neural networks with some care because many mathematical techniques rely on unique representations. However, the equivalence class was a simple finite group of transformations, which implies that the necessary precautions are often minimal. The results from this chapter will serve as a foundation to the work presented in the following chapters because they show that neural networks are appropriate models for static functions, which is a basic fact we shall use to show that dynamical neural networks are appropriate models of dynamical systems.

Chapter 3

Neural networks as dynamical systems

When studying systems that evolve over time, many methodologies are available. Perhaps the most fundamental approach is the body of work in dynamical system theory that studies the *qualitative* properties of the solutions of ordinary differential equations (ODEs) [59, 53]. This work is in contrast with the work done in numerical integration where researchers try to provide *quantitative* estimates for the solutions of ODEs. With the advent of powerful computers however, numerical techniques have become an increasingly important tool for more theoretical work [52]; the emblem of this trend being the study of chaotic dynamics [80]. In this chapter, we study the qualitative properties of neural network models of dynamical systems while we will put more emphasis on numerical methods in the last chapters.

3.1 Dynamical system theory

ODEs live on n-dimensional manifolds. A manifold $\mathcal{M}$ is, simply put, a smooth n-dimensional surface; for example a two-dimensional sphere. We often embed manifolds into some higher-dimensional linear space, of which they form a subset (like a two-dimensional sphere in a three-dimensional Euclidean space). The ODE gives us the rule for the evolution over time of the position of a point on this manifold by associating a velocity to each point on the surface. This velocity will always be tangent to the surface (we say that the velocity belongs to the tangent space of the surface, which we denote by $T_{\mathcal{M}}$), so the point never

leaves the surface. We call a particular solution to this problem a trajectory $x(t)$:

$$f : \mathcal{M} \to T_{\mathcal{M}}$$

$$\frac{dx(t)}{dt} = f(x(t))$$

The importance of this mathematical object for engineering stems from its power to model large classes of physical systems adequately (such as mechanical systems and electrical systems). A differential equation models the evolution of the state of a system over time (roughly said, the state of a physical system is a set of variables which, together with any exogenous variables, suffices to determine uniquely its future evolution). For example, we can use the ODE

$$\left\{ \begin{array}{lcl} \dot{x}_1 & = & x_2, \\ \dot{x}_2 & = & -(g/l)\sin x_1 - (\upsilon/(ml^2))x_2, \end{array} \right.$$

to model the decrease of the amplitude x_1 of the motion of a simple pendulum over time. Although it is easier to study differential equations on $\mathbb{R}^n$ (the manifold is then flat, hence linear), manifolds that are not flat are frequently necessary, especially when modeling mechanical systems. In the case of the simple pendulum, the state of the system is the angle $x_1 = \theta$ of the pendulum with a vertical line and the velocity of the pendulum $x_2 = \omega$. Since the angle is comprised between 0 and 2Π (a circle $\mathcal{S}^1$) and the velocity is real-valued, the appropriate manifold for modeling the pendulum is a two-dimensional cylinder $\mathcal{S}^1 \times \mathbb{R}$. But both dynamical system theory and numerical integration are difficult topics in the case of nonflat manifolds, so we will most often shy away from them and restrict our attention to ODEs on $\mathbb{R}^n$.

A powerful battery of tools is available to analyze nonlinear systems, although their use often requires some expertise. The most prominent methods from the qualitative theory of differential equations [59, 144, 53] rely on the study of

1. The behavior of the system around its fixed points (linearization; stable, unstable, and center manifolds)

2. The behavior of trajectories having remarkable properties (periodic orbits, homoclinic orbits, heteroclinic orbits)

3. The asymptotic behavior of trajectories (Lyapunov theory, LaSalle's invariance principle)

4. The qualitative change of behavior of the system for a variation of its parameters (perturbation analysis, bifurcation theory)

5. The decomposition of the dynamics into qualitatively different components (averaging methods, fast and slow moving manifolds)

The use of these methods in the field of neural networks is still much of an open topic. A complete review of their application to neural networks would be beyond the scope of this work. We therefore restrict ourselves to the particular problem of the asymptotic behavior of the trajectories of a neural network. Behind the choice to survey this problem lies its intimate relationship with control theory.

What does dynamical system theory have to say about the limit behavior of neural networks? The main contributions to the study of neural networks from the point of view of dynamical system theory are those of Stephen Grossberg [50], Michael Cohen [26], and Morris Hirsch [58]. Continuous-time neural networks here consist of an ODE on $\mathbb{R}^n$, where each states represents the activity of what we call a neuron and the activity of each neuron evolves as a function of the activity of the other neurons (usually, some nonlinear transformation of a linear combination of these activities, or some linear combination of a nonlinear transformation of these activities). Some examples, which recur throughout this dissertation, are the following systems:

- Hopfield-type networks:

$$\dot{x} = -\Lambda x + B\sigma(x),$$

  where $x \in \mathbb{R}^n$, Λ is a diagonal matrix, and $B \in \mathbb{R}^{n\times n}$.

- Dynamical perceptrons:

$$\dot{x} = \sigma(Ax + a_0),$$

  where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, and $a_0 \in \mathbb{R}^n$.

- Dynamical perceptrons with self-connections:

$$\dot{x} = -\Lambda x + \sigma(Ax + a_0),$$

  where $x \in \mathbb{R}^n$, Λ is a diagonal matrix, $A \in \mathbb{R}^{n\times n}$, and $a_0 \in \mathbb{R}^n$.

- Dynamical single–hidden-layer perceptrons:

$$\dot{x} = B\sigma(Ax + a_0),$$

  where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $a_0 \in \mathbb{R}^m$, and $B \in \mathbb{R}^{n\times m}$.

- Dynamical multilayer perceptrons:

$$\dot{x} = \sigma(C\sigma(B\sigma(Ax + a_0) + b_0) + c_0),$$

  where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{p\times m}, C \in \mathbb{R}^{n\times p}$ and $a_0 \in \mathbb{R}^m, b_0 \in \mathbb{R}^p, c_0 \in \mathbb{R}^n$.

There exists no clear nomenclature for dynamical neural networks as their architecture is often tailored to the specific needs of the different methods we use to analyze them. We say that a network is of Hopfield type if the sigmoid acts directly on the state (we have terms of the form $\sigma(x)$) and we say that a network is of the perceptron type if we have terms of the form $\sigma(Ax + a_0)$. We call these kinds of systems dynamical neural networks or continuous-time recurrent neural networks. We use the terms *dynamical*, *continuous-time*, and *recurrent* interchangeably in most of this dissertation.

3.2 Convergence and stability

There are many behaviors of interest in dynamical systems: stable equilibria, saddles, heteroclinic orbits (which are trajectories that leave an equilibrium point to converge to another), homoclinic orbits (which are trajectories that leave an equilibrium point to converge back to it), periodic solutions, quasi-periodic solutions, strange attractors, bifurcations between these behaviors, and so on. While nonlinear systems exhibit a wealth of phenomena absent from linear systems, it is not yet clear from an engineering point of view how we can use many of these. Nor is it clear how to synthesize systems producing a desired behavior. Even their analysis remains a delicate business. The typical engineering task is to guarantee that the behavior of a system is stable around some equilibrium. But, in contrast with linear systems, nonlinear systems can have more than one fixed point. We will therefore consider the two following fundamental problems for neural networks: When does a neural network possess a unique stable equilibrium? When does a neural network possess multiple stable equilibria? These questions are modest, yet they are already difficult to answer because of the typically high dimension of dynamical neural networks. The answer to these questions is also directly relevant to engineering because they generalize the corresponding questions about linear systems.

We first remark that, if we want to study the behavior of some solutions of an ODE, it must make sense to speak about such solutions. We assume the usual condition of Lipschitz continuity [59] of the vector fields to guarantee that there exists a unique trajectory of the dynamical system going through each point. Moreover, when we define systems on the whole of $\mathbb{R}^n$, they satisfy a global Lipschitz condition that guarantees that the solutions are defined for all times. We now define what is meant by stability and then present the results from Lyapunov and LaSalle [73] that we use to prove most of the results about neural networks. We make a semantic distinction between stability and convergence. Convergence refers here to the fact that all trajectories converge to some subset of isolated fixed points, while stability refers to the fact that all trajectories in a neighborhood of some fixed point converge to this fixed point. We first consider the convergence and quasi-convergence of a system.

Definition 3.1. A dynamical system is *convergent* if, for any initial condition x_0, the trajectory passing through this initial condition converges to some fixed point p:

$$\text{For all } x_0, \text{ there exists } p \text{ with } f(p) = 0 \text{ such that } \lim_{t\to\infty} x(t) = p.$$

Definition 3.2. A dynamical system is *quasi-convergent* if, defining $\mathcal{E}$ to be the set of equilibria ($\mathcal{E} = \{p \in \mathbb{R}^n : f(p) = 0\}$), any trajectory $x(t)$ of the dynamical system converges to the set of equilibria:

$$\lim_{t\to\infty} \min_{p\in\mathcal{E}} \|x(t) - p\| = 0.$$

Quasi-convergence is not equivalent to convergence if the equilibria are not isolated. But the trajectories of a quasi-convergent system effectively slow down in a fashion similar to those of a convergent system, although they might not exactly converge. As a consequence, quasi-convergent system cannot have periodic orbits.

We now turn to the definition of stability, first in the sense of Lyapunov and then asymptotic.

Definition 3.3. Suppose the system $\dot{x}(t) = f(x(t))$ has an equilibrium at $x = p$ ($f(p) = 0$). We say that this equilibrium is *stable* in the sense of Lyapunov if, for any $\epsilon > 0$, there exists a $\delta > 0$ such that for any initial condition x_0 within a distance δ of the equilibrium ($||x_0 - p|| < \delta$), we have that the trajectory remains within ϵ of the equilibrium for all positive times: $||x(t) - p|| < \epsilon, \forall t \geq 0$.

The definition requires that trajectories in the neighborhood of the equilibrium never wander far off from the equilibrium.

Definition 3.4. An equilibrium $x = p$ of the system $\dot{x}(t) = f(x(t))$ is *asymptotically stable* if it is stable in the sense of Lyapunov and there exists a $\delta > 0$ such that

$$\lim_{t\to\infty} ||x(t) - p|| = 0 \text{ for all } x_0 \text{ such that } ||x_0 - p|| < \delta.$$

We say that p is *globally asymptotically stable* if $\delta = \infty$.

We can obtain local stability results by studying the linearization of the system around its fixed points. Another way to study stability is through the use of Lyapunov functions. They have the advantage that they can deliver some global results while linearization techniques always retain their local character. Lyapunov functions are useful both for the study of stability and of convergence. For stability, we have the following result.

Theorem 3.1 (Lyapunov). *Let $x = p$ be an equilibrium of $\dot{x}(t) = f(x(t))$. Suppose that we can find a function $V(x)$ such that $V(x)$ is differentiable, $V(x) > 0$ for all $x \neq p$, and $\lim_{x\to\infty} V(x) = \infty$. If the derivative $dV(x)/dt$ of V along any trajectory is negative:*

$$\frac{dV(x)}{dt} = \frac{dV(x)}{dx}\frac{dx}{dt} = \nabla V(x) \cdot f(x) \leq 0, \tag{3.1}$$

then the equilibrium p is stable. If the derivative of V along any trajectory is strictly negative ($\nabla V(x) f(x) < 0$), then p is globally asymptotically stable.

We call V a *Lyapunov function* of the system $\dot{x} = f(x)$. Here we denote the gradient of the function V by $\nabla V(x) = \left(\frac{\partial V}{\partial x_1}, \ldots, \frac{\partial V}{\partial x_n}\right)$. Later, we will also use this notation for the Jacobian of a vector-valued function. For convergence, we have an important generalization of this result: The LaSalle invariance principle. To state it, we need to define the concept of an invariant set.

Definition 3.5. A set $\mathcal{D}$ is *positively invariant* if, for all $x_0 \in \mathcal{D}$, we have that the trajectory $x(t)$ through x_0 remains in $\mathcal{D}$ for all positive times: $x(t) \in \mathcal{D}, \forall t > 0$. A set $\mathcal{D}$ is *negatively invariant* if, for all $x_0 \in \mathcal{D}$, we have $x(t) \in \mathcal{D}, \forall t < 0$. A set $\mathcal{D}$ is *invariant* if it is both positively and negatively invariant.

Theorem 3.2 (LaSalle invariance principle). *Consider the system $\dot{x}(t) = f(x(t))$. Suppose that we can find a function $V(x)$ such that $V(x)$ is differentiable, $V(x) \geq 0$ for all x, and $\lim_{x\to\infty} V(x) = \infty$. If the derivative $\frac{dV(x)}{dt}$ of V along any trajectory is negative (3.1), then all trajectories converge to the set $\mathcal{E}$ where the derivative of the Lyapunov function is zero:*

$$\mathcal{E} = \left\{ x \in \mathbb{R}^n : \frac{dV(x)}{dt} = \nabla V(x) \cdot f(x) = 0 \right\}.$$

Suppose that $\mathcal{M}$ is the largest invariant set contained in $\mathcal{E}$. Then, every trajectory—which remains bounded—converges to $\mathcal{M}$.

If the derivative along the trajectory is strictly negative, we have that all trajectories converge to the set $\mathcal{E}$ of equilibria: The system is quasi-convergent. If the set of equilibria is disconnected, the system must then be convergent.

This theorem is truly general because it does not limit itself to the study of the stability of a unique fixed point. It is just as valid in the case of multiple equilibria, or of periodic orbits, and so forth. Its application proceeds in two steps. We first have to find a Lyapunov function. Second, we study the set $\mathcal{E}$ where the derivative of the Lyapunov function along a trajectory is zero and check which points are in the invariant set $\mathcal{M}$. The set $\mathcal{M}$ will be the attractor of the system.

3.3 The Cohen–Grossberg theorem

Cohen and Grossberg [26] studied a general class of competitive dynamical systems, as described in Equation 3.2. We say that these systems are competitive because the dynamics of each variable x_i is inhibited by a negative contribution $-\sum_{i=1}^{n} c_{ik} d_k(x_k)$ from the other variables. This contribution is always positively weighted ($c_{ik} \geq 0$) and increases monotonously when the other variables increase ($d'_k(x_k) \geq 0$). Cohen and Grossberg showed that there exists a Lyapunov function for this class of systems when the matrix $C = [c_{ik}]$ of the coefficients of interaction between the variables is symmetric. The equations for this competitive system are

$$\frac{dx_i}{dt} = a_i(x_i) \left[b_i(x_i) - \sum_{i=1}^{n} c_{ik} d_k(x_k) \right], \quad (i = 1, \ldots, n). \tag{3.2}$$

The system described by Cohen and Grossberg is a generalization of a model of the activity of groups of neural cells. These cells are activated by some constant input. One of the groups of cells will maximally correlate with this input and see its activity increase while the activity of this "winning" group will inhibit the activity of the other groups.

Competitive models find their origins in population biology and we can describe them easily as follows. The state variables of a competitive system represent the population of a given "species"; the state space of the dynamical system is thus limited to positive values: $x \in \mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x_i \geq 0, (i = 1, \ldots, n)\}$. The dynamics of the system is the following:

$$\frac{dx_i}{dt} = x_i c_i(x), \quad (i = 1, \ldots, n), \tag{3.3}$$

where $c_i(x) : \mathbb{R}^n_+ \to \mathbb{R}$ has the following properties:

1. The competition functions $c_i(x)$ are C^∞.
2. For all $i = 1, \ldots, n$ and $j = 1, \ldots, n$, we have $\dfrac{\partial c_i(x)}{\partial x_j} < 0$ for all x such that $x_k > 0$ for all $k = 1, \ldots, n$.
3. There is a $K > 0$ such that, for all $i = 1, \ldots, n$, if $||x|| > K$, we have $c_i(x) < 0$.

The second condition is the most significant and states that the growth of a population tends to inhibit the growth of the other population and of itself (by competition for the available resources). The third condition states that, beyond some population level, all populations have to decrease because there are not enough resources available anymore. The factor x_i in front of the interaction functions $c_i(x)$ comes from the fact that the quantities that we

study are not the population levels directly, but the rates of growth of the populations. In the case of a population growing without bound at a constant growth rate α, we have $\dot{x} = \alpha x$ and thus $x(t) = e^{\alpha t} x_0$, which we can rewrite as $\ln(x(t)) - \ln x_0 = \alpha$. The quantity we want to study is therefore $\ln(x(t))$ and the equations for our competitive system are

$$\frac{d \ln x_i}{dt} = c_i(x),$$

which is equivalent to (3.3). The Lotka–Volterra systems that we describe in Chapter 4 are the result of the linearization of $c_i(x)$ around zero. However, competitive systems are actually extremely general since Smale [129] showed that we can represent any dynamical system by a higher-dimensional competitive system and since, as we show in the next chapter, we can approximate any dynamical system arbitrarily closely for a finite time by using Lotka–Volterra systems.

The Cohen–Grossberg network (3.2) has the following neural network as a particular case:

$$\begin{aligned} \dot{x}_i = \; & -a_i x_i + (b_i - c_i x_i)(u_i + f_i(x_i)) \\ & -(d_i x_i + e_i)\left(v_i + \textstyle\sum_{k=1}^{n} F_{ik} g_k(x_k)\right), \end{aligned} \tag{3.4}$$

with $i = 1, \ldots, n$. The state variables represent the activity of a population of cells. The first term forces the activity of the cell population to decay naturally as $-a_i x_i$. In the second term, which is excitatory, the state x_i is fed back into itself together with the input u_i and excites the cell population proportionally to $b_i - c_i x_i$. In contrast, the third term is inhibitory: The input v_i together with the contribution $\sum_{k=1}^{n} F_{ik} g_k(x_k)$ from all the cell populations inhibit the activity of the cell population i proportionally to $d_i x_i + e_i$. The initial condition are chosen such that $b_i - c_i x_i$ and $d_i x_i + e_i$ are both positive; the equations further guarantee that these quantities will remain positive for all times, which means that excitatory and inhibitory contributions always remain that way and the value of the signals always remain in a finite interval. By setting the inputs u_i and v_i to some constant value $(u_1, u_2, \ldots, u_n; v_1, v_2, \ldots, v_n)$ our system might converge to some stable equilibrium $(p_1, p_2, \ldots, p_n)$. We may then view our network as processing the input pattern $(u_1, u_2, \ldots, u_n; v_1, v_2, \ldots, v_n)$ to give the pattern $(p_1, p_2, \ldots, p_n)$ as output. Such processing is similar to what happens in associative memories, where we feed the system with an initial pattern and then let it evolve until it stabilizes at a recalled pattern. Indeed, by choosing the feedbacks $f_i(x_i)$ and $g_i(x_i)$ to be sigmoidal, we get a generalization of the following Hopfield-type network, which has been used as an associative memory [61]:

$$\dot{x}_i = -a_i x_i + \sum_{k=1}^{n} A_{ik} \sigma(x_k) + u_i.$$

Theorem 3.3 (Cohen and Grossberg). *Let us consider a system*

$$\dot{x}_i = a_i(x_i)\left[b_i(x_i) - \sum_{i=1}^{n} c_{ik} d_k(x_k)\right], \quad (i = 1, \ldots, n)$$

such that (for all $i = 1, \ldots, n$)

1. *(Symmetry) The interaction matrix C is symmetric and all its entries are positive ($c_{ik} \geq 0$ for all $i = 1, \ldots, n$ and $k = 1, \ldots, n$).*
2. *(Continuity) The functions $a_i(\xi)$ are continuous for $\xi \geq 0$ and the functions $b_i(\xi)$ are continuous for $\xi > 0$.*
3. *(Positivity) For $\xi > 0$, we have $a_i(\xi) > 0$ and, for $\xi \in \mathbb{R}$, $d_i(\xi) \geq 0$.*
4. *(Monotonicity) The interaction functions $d_i(\xi)$ are all differentiable and strictly increasing: $d_i'(\xi) > 0$ for $\xi \geq 0$.*
5. *(Boundedness of trajectories) We have $\lim_{\xi \to \infty} \sup(b_i(\xi) - c_{ii} d_i(\xi)) < 0$.*
6. *(Positivity of trajectories) We have either $\lim_{\xi \to 0^+} b_i(\xi) = \infty$, or we have $\lim_{\xi \to 0^+} b_i(\xi) = M < \infty$ and $\int_0^\epsilon \frac{1}{a_i(\xi)} d\xi = \infty$ for some $\epsilon > 0$.*

Then the function

$$V(x) = -\sum_{i=1}^{n} \int_0^{x_i} b_i(\xi_i) d_i'(\xi_i) d\xi_i + \frac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{jk} d_j(x_j) d_k(x_k) \tag{3.5}$$

is a Lyapunov function for this system and any trajectory of this system converges to an isolated equilibrium.

To establish that (3.5) is a Lyapunov function for the system (3.2), we first have to establish that it satisfies the requirements for a Lyapunov function. We have to check that the function $V(x)$ is always positive and that it is continuous and differentiable for every x. We limit ourselves to the positive orthant: $x_i \geq 0$ for all i since we are studying a competitive system and the activity of a population never goes below zero. We will show that $V(x)$ satisfies the necessary conditions by showing first that every trajectory starting from an initial condition in the positive orthant always remain in the positive orthant. Conditions 5 and 6 of the above theorem insure respectively the boundedness and the positivity of the trajectories.

Lemma 3.4. *Let system (3.2) be such that it satisfies the conditions of Theorem 3.3. If $x_i(0) > 0$ for all i, then there exists a bound M and a time T such that $x_i(t) < M$ for all $t < T$; furthermore, $x_i(t) > 0$ for all t.*

Proof. We prove that the trajectories are bounded using the following inequality:

$$b_i(x_i) - \sum_{k=1}^{n} c_{ik} d_k(x_k) \leq b_i(x_i) - c_{ii} d_i(x_i),$$

which is true since all c_{ik} are positive and all the functions $d_k(x_k)$ are positive also. We have that $\lim_{\xi \to \infty} \sup(b_i(\xi) - c_{ii} d_i(\xi)) < 0$, which implies that there exists a bound $M > 0$ such that $x_i > M$ for all i implies $b_i(x_i) - c_{ii} d_i(x_i) < 0$. This means that, for any point such that $x_i > M$ for all i, we have

$$\dot{x}_i = a_i(x_i) \left[b_i(x_i) - \sum_{i=1}^{n} c_{ik} d_k(x_k) \right] < 0$$

since $a_i(x_i)$ is always positive. Thus, the vector field always points toward the origin if we go far enough towards infinity. This property implies that, for any positive initial condition, there exists a time T after which $x_i(t) < M$ for all i and for all $t > T$: The trajectories are indeed bounded.

Now, we want to prove that the trajectories always remain positive. The idea is that a trajectory can never cross the $x_i = 0$ hyperplane. Suppose, in a first case, that $\lim_{\xi \to 0^+} b_i(\xi) = \infty$. Since $\sum_{k=1}^{n} c_{ik} d_k(x_k)$ is bounded if $x_k \leq M$ for all k, as x_i tends to zero, $b_i(x_i) - \sum_{i=1}^{n} c_{ik} d_k(x_k)$ always become positive and $\dot{x}_i > 0$ since $a_i(x_i) > 0$ if $x_i > 0$. Thus, x_i can never become negative and all trajectories must always remain positive. Suppose, in a second case, that $\lim_{\xi \to 0^+} b_i(\xi) = M < \infty$ and $\int_0^{\epsilon} (1/a_i(\xi)) d\xi = \infty$ for some $\epsilon > 0$. If we suppose that there is a first time T such that $x_i(T) = 0$, we have $\int_{x_i(0)}^{0} \frac{1}{a_i(\xi)} d\xi = -\infty$. Because of the definition of system (3.2), the previous expression is equal to $\int_0^T [b_i(x_i(t)) - \sum_{i=1}^{n} c_{ik} d_k(x_k(t))]\, dt$. Since the trajectories are bounded, the integrand remains bounded: $\int_0^T [b_i(x_i(t)) - \sum_{i=1}^{n} c_{ik} d_k(x_k(t))]\, dt > -\infty$, which is a contradiction. So, under the hypothesis of Theorem 3.3, the trajectories of the system always remain positive and bounded for any positive initial condition. □

To show that all the trajectories of this system converge to an equilibrium point, we apply the LaSalle invariance principle. For this, we need to guarantee that $V(x)$ is continuous and bounded along any positive trajectory.

Proposition 3.5. *Let system (3.2) be such that it satisfies the conditions of Theorem 3.3, then $V(x)$ is continuous and bounded along any positive bounded trajectory.*

Proof. We look at the function $V(x)$ and consider Condition 6 in Theorem 3.3. Suppose $\lim_{\xi \to 0^+} b_i(\xi) = M < \infty$, then the terms $\int_0^{x_i} b_i(\xi_i) d_i'(\xi_i) d\xi_i$ are all finite because all trajectories remain bounded. Furthermore, the product terms

$\frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{jk}d_j(x_j)d_k(x_k)$ are also finite because all variables x_k remain bounded and the functions $d_k(x_k)$ are continuous. Since all terms are continuous, $V(x)$ is also continuous. If $\lim_{\xi\to 0^+} b_i(\xi) = \infty$, it is necessary to modify the function $V(x)$ slightly for the proposition to hold, but this modification is possible, see [26]. □

We now need to show that $V(x)$ is decreasing along the trajectories.

Proposition 3.6. *The derivative of $V(x)$ along trajectories is negative:*

$$\frac{d}{dt}V(x(t)) \le 0.$$

Proof. It is interesting that we can write our system (3.2) in gradient form. If we define the diagonal matrix G such that

$$G(x) = \operatorname{diag}(a_1(x_1)/d_1'(x_1), \dots, a_n(x_n)/d_n'(x_n))$$

(with the notation "diag" meaning that the n given elements come on the diagonal of a $n \times n$ square matrix, all nondiagonal elements being zero) and set the function $H(x) = -V(x)$, we have $\dot{x}(t) = G(x(t))\nabla H(x(t))$. Indeed, the gradient ∇H is given by

$$\begin{aligned}\frac{\partial H}{\partial x_i} &= \frac{\partial}{\partial x_i}\int_0^{x_i} b_i(\xi_i)d_i'(\xi_i)d\xi_i \\ &\quad + \frac{1}{2}\frac{\partial}{\partial x_i}\sum_{k=1}^{n} c_{ik}d_i(x_i)d_k(x_k) \\ &\quad + \frac{1}{2}\frac{\partial}{\partial x_i}\sum_{j=1}^{n} c_{ji}d_j(x_j)d_i(x_i) \\ &= b_i(x_i)d_i'(x_i) + \frac{1}{2}\sum_{k=1}^{n}(c_{ik}+c_{ki})d_i'(x_i)d_k(x_k)\end{aligned}$$

Since, the matrix C is symmetric, we indeed have

$$\dot{x} = G(x)\nabla H(x) = \sum_{i=1}^{n}\frac{a_i(x_i)}{d_i'(x_i)}\left(b_i(x_i)d_i'(x_i) + \sum_{k=1}^{n} c_{ik}d_i'(x_i)d_k(x_k)\right),$$

which is our original system (3.2). To prove that $V(x)$ decreases along the trajectories, we can either compute the derivative explicitly,

$$\frac{dV(x)}{dt} = -\sum_{i=1}^{n} a_i(x_i)d_i'(x_i)\left[b_i(x_i) - \sum_{k=1}^{n} c_{ik}d_k(x_k)\right]^2 \le 0; \tag{3.6}$$

or use our gradient representation,

$$\frac{dV(x)}{dt} = \frac{dV(x)}{dx}\frac{dx}{dt} = \nabla V \cdot (-G\nabla V) = -G(\nabla V)^2 \le 0.$$

The inequality is true because $a_i(x_i) > 0$ and $d_i'(x_i) > 0$ for all i. □

Proof of Theorem 3.3. Now, by the LaSalle invariance principle, all trajectories converge to the largest invariant set $\mathcal{M}$ contained in the set of points $\mathcal{E}$ where the Lyapunov function remains constant:

$$\mathcal{E} = \left\{ x \in \mathbb{R}^n : \dot{V}(x) = \nabla V(x) \cdot f(x) = 0 \text{ and } x_i \geq 0 \text{ for all } i \right\}.$$

We have $\dot{V}$ as in (3.6). Since we have assumed that the functions $a_i(x_i)$ and $d_i'(x_i)$ are strictly positive for all i and that the third factor is a square, the only possibility for $\dot{V} = 0$ is if $b_i(x_i) - \sum_{k=1}^{n} c_{ik} d_k(x_k) = 0$. The points satisfying this condition are the equilibrium points of our original system (3.2). In this situation where the directional derivative of the Lyapunov function is zero only where the vector field is zero, we say that the Lyapunov function is strict. We have therefore shown that, under the conditions of Theorem 3.3—in particular, the symmetry of the interaction matrix C—all the fixed points of (3.2) are stable and all trajectories converge to one of these fixed points. □

Cohen and Grossberg relax the monotonicity assumption further by requiring only that the interaction functions $d_i(\xi)$ are all nondecreasing: $d_i'(\xi) \geq 0$ for $\xi \geq 0$. However, the proof of the theorem then requires some machinery from differential topology and the theory of several complex variables to study the equilibrium set of the system. The difficulty is that we need to make sure that the equilibria are isolated fixed points. The requirement that the activation function be nondecreasing instead of strictly increasing is of importance in the case where the activation function has a threshold below which its output is zero—such as the linear threshold activation function:

$$\sigma(\xi) = \begin{cases} 0, & \text{if } \xi < 0, \\ \xi, & \text{if } 0 \leq \xi \leq 1, \\ 1, & \text{if } \xi > 1. \end{cases}$$

We do not discuss this case here. Theorem 3.3 is general because it gives much freedom to choose the functions a_i, b_i, and d_i. However, some of the assumptions are rather strong: (1) we study the system only in the positive orthant, (2) the matrix C must be symmetric, and (3) all its entries must be positive. These assumptions may be too restrictive in many practical applications.

3.4 Cascades

Hirsch [58] considered cascades of neural networks and gave conditions under which they converge. A cascade is a chain of dynamical neural networks where the output of a network is fed into the input of any of the following ones. The results hold for any kind of system with inputs and is in no way limited to dynamical neural networks. It is, however, especially relevant to neural

networks because the cascade structure where the different networks feed their outputs into each other is typical of neural networks.

We consider systems with n states and k groups of m_k inputs of the form

$$\dot{x} = N(x, u_1, \dots, u_k),$$

where $x \in \mathbb{R}^n$ and N is a Lipschitz continuous function from $\mathbb{R}^n$ to $\mathbb{R}^n$ for any value of $(u_1, \dots, u_k) \in \mathbb{R}^{m_1} \times \cdots \times \mathbb{R}^{m_k}$. A cascade will be of the form

$$\begin{array}{rcl} \dot{x}^{(1)} & = & N_1(x^{(1)}), \\ \dot{x}^{(2)} & = & N_2(x^{(2)}, x^{(1)}), \\ & \vdots & \\ \dot{x}^{(c)} & = & N_c(x^{(k)}, \dots, x^{(1)}), \end{array} \tag{3.7}$$

and the different variables do not need to have the same size. The states of the preceding systems play the role of inputs in the following ones. It is interesting to study the stability and convergence properties of this type of systems. We have the following results.

It is often enough to analyze the case of a cascade of two systems and generalize from there to cascades of multiple systems. We write this simple system down:

$$\begin{array}{rcl} \dot{x} & = & N(x) \\ \dot{y} & = & M(y, x) \end{array} \tag{3.8}$$

Theorem 3.7 (Hirsch). *Let a family of systems $\dot{x}^{(i)} = N_i(x^{(i)}, u^{(i)})$ be globally asymptotically stable for any value of $u^{(i)}$, for all i; then the cascade (3.7) is globally asymptotically stable.*

Proof. We look at a cascade of two networks (3.8). If the statement is true for a cascade of two networks, it will be true for a cascade of arbitrary length. Indeed, we view a cascade of three networks as the cascade of a cascade of two networks with a single network; and we repeat this argument inductively. The first network in (3.8) converges to a unique equilibrium p since we have supposed that the system was globally asymptotically stable. Thus, any trajectory $(x(t), y(t))$ of the cascade will converge to the hyperplane $\{(x, y) : x = p\}$. But this hyperplane is positively (and negatively) invariant. This means that we can apply the LaSalle invariance principle. We have assumed that the system $\dot{y} = M(y, u)$ had a globally asymptotically stable equilibrium for any value of u. In particular, assume that $\dot{y} = M(y, p)$ converges to $y = q$. In our hyperplane, the single point (p, q) is the maximum invariant set. Since all trajectories $(x(t), y(t))$ converge to the hyperplane and that only the equilibrium (p, q) is invariant in this hyperplane, this implies that all trajectories converge to (p, q); hence, this equilibrium is globally asymptotically stable. □

Theorem 3.8 (Hirsch). *Consider a two-network cascade (3.8) and suppose that the equilibria of $\dot{x} = N(x)$ (the zeros of N) are isolated and that, for each equilibrium p of $\dot{x} = N(x)$, the linearization around any equilibrium of the system $\dot{y} = M(y,p)$ never has zero eigenvalues. Assume that there exists a C^1 strict Lyapunov function $V(x)$ for $\dot{x} = N(x)$ and there exists a C^2 strict Lyapunov function $U_p(y)$ for each system $\dot{y} = M(y,p)$. Then, there exists a C^1 strict Lyapunov function for the whole system (3.8).*

We refer the reader to [58] for the proof. The main idea is to take a combination of the Lyapunov functions $V(x)$ and $U_p(y)$, making sure that the derivative along the trajectories always remain negative. Moreover, we can apply this theorem inductively to cascades of Cohen–Grossberg networks (3.2) with symmetric weights and slightly stronger conditions than those of the Cohen–Grossberg theorem (3.3), as shown hereafter.

Theorem 3.9 (Hirsch). *Consider the competitive network (3.2) where the functions $a_i(x_i), b_i(x_i)$, and $d_i(x_i)$ are C^3 for all i. Suppose $a_i(x_i) > 0$ and $d_i'(x_i) > 0$ for all i. Suppose that the linearization of the system around any of the equilibria never has zero eigenvalues. Assume further that the interaction matrix is in upper block-triangular form and the diagonal blocks are symmetric. Then the system has a strict C^1 Lyapunov function.*

Proof. The cascade structure comes from the upper block-diagonal form. Every diagonal block determines a competitive network that accepts inputs from all the previous networks (blocks) and only from them. We want to apply Theorem 3.8, but we first have to show that we can apply it to a cascade of more than two networks. This application is possible in the case under consideration, but not in general for cascades of networks having a Lyapunov function. Suppose the last block of the matrix is from row and column r to n. This block is then of the form

$$\dot{x}_i = N_i((x_r, \ldots, x_n)) = a_i(x_i)\left[b_i(x_i) - \sum_{k=r}^{n} c_{ik} d_k(x_k)\right], \quad (i = r, \ldots, n).$$

We can apply the Cohen–Grossberg theorem to it and get a strict C^1 Lyapunov function since $d_i'(x_i) > 0$. Suppose the penultimate block is between rows and columns s and $r-1$. This block gives then a network of the form:

$$\begin{aligned} \dot{x}_i &= a_i(x_i)\left[b_i(x_i) - \sum_{k=s}^{r-1} c_{ik} d_k(y_k)\right] - a_i(x_i) \sum_{k=r}^{n} c_{ik} d_k(x_k) \\ &= M_i((x_s, \ldots, x_{r-1}), (x_r, \ldots, x_n)), \quad (i = s, \ldots, r-1). \end{aligned}$$

If the value of x_r to x_n are fixed, the last term is equal to $a_i(x_i)$ times a constant and we can move it into the contribution of $b_i(x_i)$. We are then in the hypotheses of the Cohen–Grossberg theorem and the system is globally

asymptotically stable and possesses a strict Lyapunov function. The two blocks are in cascade form; the first block has a Lyapunov function and the second block has a Lyapunov function for any input equal to some equilibrium of the first block. However, to apply Theorem 3.8, we need the Lyapunov function of the second block to be C^2 while the Cohen–Grossberg theorem only gives us a C^1 Lyapunov function. To guarantee that the Lyapunov function can be C^2, we have to require that $a_i(x_i), b_i(x_i)$, and $d_i(x_i)$ are C^3 for all i. With this assumption, we can apply Theorem 3.8, which gives us a C^1 Lyapunov function for the system made out of the first two blocks. We are then in a position to repeat the application of this method to the third block, since we have a C^1 Lyapunov function for the first two blocks and can find a C^2 Lyapunov function for the third block. This argument proves Theorem 3.9 inductively. □

3.5 Global asymptotic stability

We have so far discussed conditions for the convergence of all the trajectories of a dynamical neural network to some equilibrium (any two trajectories will converge, but not necessarily to the same point). A more limited question is to see when we can guarantee that all trajectories will converge to a unique equilibrium. Conditions under which the equilibrium of a neural network is globally asymptotically stable are probably the most necessary tools for engineering applications of neural networks in the field of control. Most control designs try to guarantee the asymptotic stability of the system resulting of the feedback interconnection of a plant with its controller. Many types of neural networks have the following desirable property. If we use a neural network as a model of a nonlinear plant and we use a linear system or a similar neural network as controller, the resulting dynamical system is a neural network *without inputs* (except maybe for a reference signal). We therefore want to guarantee asymptotic stability by an appropriate choice of the parameters of the controller (as these are the free parameters). This procedure justifies the coming discussion of global asymptotic stability from the point of view of applications. This is not to say that globally asymptotically stable designs should be our sole goal. Many other behaviors occur in dynamical systems and they should be used whenever appropriate. However, there is still little knowledge on how to use them for controller design.

We can obtain, as usual, local stability conditions by linearizing the system around the origin. There is an extension of this technique that guarantees global asymptotic stability [58]. It consists in showing that the distance between the trajectories starting from two different initial conditions must decrease over time. We prove this fact by linearizing the system at all points (not just at the fixed point) and making sure that the system locally forces trajectories to come together. We denote the linearization of the vector field f of the ODE $\dot{x} = f(x)$ by the Jacobian matrix $A(x) = \nabla f(x) = \left[\frac{\partial f_i(x)}{\partial x_j}\right]$.

Theorem 3.10 (Hirsch). *Consider the dynamical system $\dot{x}(t) = f(x(t))$ with linearization $A(q)$ around the point q. Suppose there exists a constant $\mu > 0$ such that $q^T A(q) \leq -\mu q^T q$ for all $q \in \mathbb{R}^n$, then the system possesses a unique equilibrium, which is globally asymptotically stable.*

Proof. We take two initial conditions $y(0)$ and $z(0)$. We look at the evolution over time of the distance $||y(t) - z(t)||^2$ between the trajectories going through these two points. We estimate this distance by taking the Taylor series of the vector field around $y(t)$: The series is, up to first order, $f(q) = f(y(t)) + A(y(t))(q-y(t)) + R(y(t), q-y(t))$ with $\lim_{q \to y(t)} R(y(t), q-y(t))/(q-y(t)) = 0$. The remainder R takes the higher-order terms into account but depends on the point $y(t)$ where we take the Taylor series. This means that, at any given point $y(t)$ and for any $\epsilon > 0$, there is a $\delta > 0$ such that, if $||q - y(t)|| < \delta$, we have $||R(y(t), q - y(t))|| < \epsilon ||q - y(t)||$. So, we take our initial conditions such that $||y(0) - z(0)|| < \epsilon$. We have, for the derivative of the distance between the trajectories,

$$\begin{aligned} \frac{1}{2}\frac{d}{dt}||y(t) - z(t)||^2 &= \frac{1}{2}\frac{d}{dt}(y(t) - z(t))^T (y(t) - z(t)) \\ &= [f(y(t)) - f(z(t))]^T (y(t) - z(t)) \\ &= [A(y)(z - y) + R(y, z - y)]^T (y - z) \\ &\leq A(y(t))||z(t) - y(t)||^2 + \epsilon ||z(t) - y(t)||^2. \end{aligned}$$

Suppose now that $\epsilon < (1/2)\mu$. We know that $q^T A(q) q \leq -\mu q^T q$ for any $q \in \mathbb{R}^n$. We therefore have

$$\frac{1}{2}\frac{d}{dt}||y(t) - z(t)||^2 \leq -(\mu/2)||y(t) - z(t)||^2.$$

This inequality implies that the distance between two trajectory decreases exponentially. This fact justifies our assumption about the bound on the remainder of the Taylor series:

$$R(y(t), z(t) - y(t)) \leq \epsilon ||z(t) - y(t)||$$

since the trajectories are always getting closer. Further, we do not need to assume that the distance between two initial conditions is small since, between two arbitrary initial conditions, there is always a *finite* sequence of points that will be close enough to guarantee the convergence of the trajectories passing through two successive points in the sequence. By transitivity, this relation guarantees that the two initial conditions have trajectories that converge to each other. This convergence implies that all the trajectories of the system converge to each other. This global convergence in turn implies that there exists a unique equilibrium for the system, which is globally asymptotically stable. Remark that we only know that the equilibrium exists and not where

it is. Now, the condition for convergence is that the Jacobian $A(q)$ of the vector field is such that there exists a $\mu > 0$ for which $A(q)^T q \leq -\mu q^T q$. This condition is equivalent to requiring that the largest eigenvalue of $1/2(A^T + A)$ be smaller than $-\mu$. This relation is true if

$$A_{ii} + \frac{1}{2} \sum_{j \neq i}^{n} |A_{ij} + A_{ji}| \leq -\mu, \quad (i = 1, \ldots, n), \tag{3.9}$$

according to Gerschgorin's circle theorem [147]. □

We can directly apply this result to neural networks, for example to Hopfield-type networks. We have the following corollary.

Corollary 3.11. *Consider the neural network*

$$\dot{x}_i = -\lambda_i x_i + \sum_{j=1}^{n} B_{ij} \sigma(x_j),$$

where $\lambda_i > 0$, the activation function σ is differentiable, monotone increasing, and its derivative is bounded: $0 \leq \sigma'(q) \leq \rho$ for all $q \in \mathbb{R}$. If

$$B_{ii} + \frac{1}{2} \sum_{j \neq i}^{n} (|B_{ij}| + |B_{ji}|) < \frac{\lambda_i}{\rho}, \quad (i = 1, \ldots, n), \tag{3.10}$$

then the neural network possesses a unique globally asymptotically stable equilibrium.

This corollary is a direct application of Theorem 3.10. For the elements of the Jacobian matrix, we have $A_{ii} = \partial f_i(x)/\partial x_i = -\lambda_i + B_{ii}\sigma'(x_i) \leq -\lambda_i + \rho B_{ii}$ and $|A_{ij}| = \left|\frac{\partial f_i(x)}{\partial x_j}\right| = |B_{ij}\sigma'(x_j)| \leq \rho |B_{ij}|$. Noting that $|A_{ij} + A_{ji}| \leq |A_{ij}| + |A_{ji}| \leq \rho(|B_{ij}| + |B_{ji}|)$ and substituting these quantities in (3.9), we get the condition (3.10) of the corollary. We can drop the term in $-\mu$ by making the inequality strict since there is a finite number of equations.

3.6 Lur'e–Postnikov Lyapunov functions

Another important source of techniques for checking global asymptotic stability of dynamical neural networks arises from their interpretation as Lur'e systems (see Figure 3.1 for an illustration). A Lur'e system results of the feedback interconnection of a linear system together with a static nonlinearity:

$$\begin{aligned} \dot{x} &= Cx + Bu, \\ y &= Ax, \end{aligned}$$

$$u = \psi(y),$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}^m, C \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}, A \in \mathbb{R}^{m\times n}$, and ψ is a map from $\mathbb{R}^m$ to $\mathbb{R}^m$. We use a notation slightly different from the usual notation for Lur'e systems to ensure compatibility with our notation for neural networks.

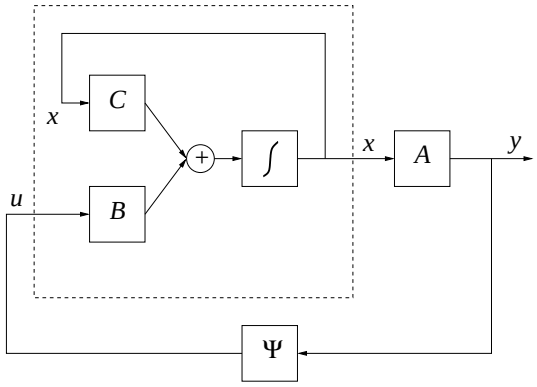


Figure 3.1: Block diagram of a Lur'e system. The dashed box represents the linear dynamical system $\dot{x} = Bx + Cu$. The output $y = Ax$ is fed back at the input through the static nonlinearity, $u = \Psi y$.

If we assume that the map ψ is a diagonal sigmoidal nonlinearity σ and we restrict our attention to the simple case where $A = I^{n\times n}$ (we use the notation $I^{n\times n}$ to represent an $n \times n$ unit matrix) and C is a diagonal matrix ($C = -\Lambda$, where all the diagonal elements of Λ are strictly positive), we get the following system:

$$\begin{aligned} \dot{x} &= -\Lambda x + Bu, \\ u &= \sigma(x). \end{aligned}$$

Substituting the equation for u in the equation for $\dot{x}$, we get the neural network

$$\dot{x} = -\Lambda x + B\sigma(x). \tag{3.11}$$

This equation represents a Hopfield-type network.

In the usual application of Lyapunov's second method, we have a system we want to analyze and we believe to be stable. We then try to find a Lyapunov function that proves stability. There is no systematic method for finding a Lyapunov function and the search proceeds by intuition and trial-and-error. In the Lur'e–Postnikov problem, we try to circumvent this problem by adopting a "reverse-engineering" philosophy. We build a parametrized class of candidate Lyapunov functions, trying to make it as large as possible while keeping

it as easy to manipulate as possible. Then, when given a system, we check whether some function in this class is a Lyapunov function for the system under consideration. If this is the case, the system is stable; otherwise we cannot decide since we have considered only a limited set of candidates. The method gives us only sufficient conditions for stability and is therefore conservative. However, we now have a fairly systematic procedure for checking stability: (1) take a family of candidate Lyapunov functions—which we know is easy to manipulate—, (2) impose that the function is a Lyapunov function for the system under consideration, (3) check under which conditions it is possible to satisfy the previous requirement. Since our original system (whether a Lur'e system or a neural network) itself depends on a number of parameters and since the candidate Lyapunov function is fairly easy to manipulate, we then get algebraic conditions that the parameters of the original system must satisfy to make it possible to find a Lyapunov function. We thus have algebraic conditions guaranteeing global asymptotic stability of our Lur'e system or neural network. The typical illustration of this philosophy would be to use a quadratic form as candidate Lyapunov function $V(x) = x^T P x$ where P is a symmetric positive definite function. In the case of linear systems, the existence of a Lyapunov function within this class is both a necessary and sufficient condition for global asymptotic stability. In the case of nonlinear systems it will only provide sufficient conditions. These conditions are conservative; but there is no practical generalization available that gives necessary and sufficient conditions for the stability of nonlinear systems. But, the following type of candidate Lyapunov function, called Lur'e–Postnikov Lyapunov function, has proved its usefulness:

$$V(x) = x^T P x + 2\alpha \sum_{i=1}^{n} \delta_i \int_0^{x_i} \psi_i(\xi) d\xi.$$

According to Narendra and Taylor [110]: "The precise reason for choosing this form is not known .... The great merit of this form when applied to the Lur'e problem is that $\dot{V}$ is a quadratic form, and thus the problem of manipulating $\dot{V}$ into a negative semidefinite form is greatly simplified".

In the case of neural networks, Forti and Tesi [38] considered a network slightly more general than (3.11):

$$\dot{x} = -\Lambda x + B\sigma(x) + b_0 = f(x), \tag{3.12}$$

where b_0 is a column vector. The main difference between (3.12) and (3.11) caused by the bias b_0 is that, while $x = 0$ is a fixed point of (3.11), it is not clear where the fixed points of (3.12) are, or even whether they exist. We need to define some properties of the elements of (3.12).

Definition 3.6. A square matrix B is a *P matrix* if all its principal minors are positive. It is a *P_0 matrix* if all its principal minors are nonnegative.

Definition 3.7. A square matrix B is *Lyapunov diagonally stable* if there is a diagonal matrix Δ with all positive diagonal elements such that the symmetric part $((\Delta B)^T + \Delta B)/2$ of ΔB is positive definite. We say that B is *Lyapunov diagonally semistable* if $((\Delta B)^T + \Delta B)/2$ is positive semidefinite.

Definition 3.8. We say that the activation function $\sigma(\xi) : \mathbb{R} \to \mathbb{R}$ satisfies the *sector* $[0, K]$ *condition* if $\sigma(0) = 0$ and all cords have slopes between 0 and K: $0 \leq \frac{\sigma(\xi_2)-\sigma(\xi_1)}{\xi_2-\xi_1} \leq K$ for all $\xi_1, \xi_2 \in \mathbb{R}$. If the activation function σ is differentiable, the condition on the slopes is equivalent to requiring $0 \leq \sigma'(\xi) \leq K$ for all $\xi \in \mathbb{R}$, by Rolle's theorem.

Forti and Tesi study the existence and uniqueness of equilibrium points by looking at the map $h(x) = \Lambda^{-1}B\sigma(x) + \Lambda^{-1}b_0$. The fixed points p of this map ($h(p) = p$) are the zeros of the vector field: $f(p) = -\Lambda p + \Lambda h(p) = 0$ and thus the equilibria of the Hopfield network. Forti and Tesi study conditions under which the map h is continuous and one-to-one with a continuous inverse (h is a homeomorphism: It is bijective). Brouwer's fixed point theorem together with the bijectivity of h guarantee existence and uniqueness of the equilibrium of a Hopfield network. They give the following proposition.

Proposition 3.12. *Let σ satisfy a sector $[0, K]$ condition. If $-B + (1/K)\Lambda$ is Lyapunov diagonally stable, then the network $\dot{x} = -\Lambda x + B\sigma(x) + b_0$ has a unique equilibrium.*

Their results also cover the more general cases where the activation function of each neuron might be different and might not satisfy the sector condition.

Lemma 3.13. *Let the $B \in \mathbb{R}^{n\times n}$, Λ be a diagonal matrix with all positive diagonal elements, and $K > 0$. Suppose that $-B + (1/K)\Lambda$ is Lyapunov diagonally stable (there exists a diagonal matrix Δ with all positive diagonal elements such that $-((\Delta B)^T + \Delta B)/2 + (1/K)\Delta\Lambda$ is positive definite). Then, we can find $\alpha > 0, \gamma > 0$, and the following $n \times n$ matrices: Q invertible, R arbitrary; P, S symmetric and positive definite such that*

$$\begin{aligned} &P(-\Lambda) + (-\Lambda)P = -QQ^T, \\ &P(-B) = -QR + \alpha\Delta(-\Lambda) + \gamma\Delta\Lambda, \\ &R^TR + S = -\alpha((\Delta B)^T + (\Delta B)) + 2(1/K)\gamma\Delta\Lambda. \end{aligned} \tag{3.13}$$

This system gives the algebraic relation we were talking about in our Lur'e–Postnikov procedure. If a solution exists to the above problem, then this solution gives us a set of parameters that we can use to prove that our candidate is indeed a Lyapunov function for our neural network. The lemma gives a sufficient condition for the existence of a solution, we refer the reader to [38] for its proof.

Theorem 3.14 (Forti and Tesi). *Suppose that the activation function σ satisfies a sector $[0, K]$ condition and that $-B + (1/K)\Lambda$ is Lyapunov diagonally stable. Then the system $\dot{x} = -\Lambda x + B\sigma(x) + b_0$ has a globally asymptotically stable equilibrium.*

Proof. By Proposition 3.12, we know that there exists a unique equilibrium p. We consider as candidate the generalized Lur'e–Postnikov function parameterized by a symmetric positive definite matrix P, a diagonal matrix Δ, and $\alpha > 0$ (we write $\Delta = \mathrm{diag}(\delta_1, \ldots, \delta_n)$):

$$V(x) = x^T P x + 2\alpha \sum_{i=1}^{n} \delta_i \int_0^{x_i} (\rho_i(\xi))\, d\xi,$$

where we define $\rho(x) = (\rho_1(x_1), \ldots, \rho_n(x_n))^T = \sigma(x + p) - \sigma(p)$. If we can find P, Δ, and γ such that the derivative of V along the trajectories of (3.12) is always strictly negative, then $V(x)$ is a Lyapunov function for this Hopfield-type network. We will look for conditions on the weights matrices Λ and B of (3.12) that guarantee that we can find such matrices P, Δ, and a scalar α. Evaluating the derivative of the candidate Lyapunov function along the trajectories, we get

$$\begin{aligned}
\frac{dV(x(t))}{dt} &= (\nabla V(x))^T f(x) \\
&= (2Px + 2\alpha\Delta\rho(x))^T (-\Lambda x + B\rho(x)) \\
&= 2x^T P(-\Lambda)x + 2x^T PB\rho(x) \\
&\quad +2\alpha\rho^T(x)\Delta(-\Lambda x) + 2\alpha\rho^T(x)\Delta B\rho(x) \\
&= x^T(P(-\Lambda) + (-\Lambda)P)x \\
&\quad -2x^T(P(-B) - \alpha\Delta(-\Lambda))\rho(x) \\
&\quad -2\alpha\rho^T(x)\tfrac{((\Delta(-B))^T + \Delta(-B))}{2}\rho(x).
\end{aligned} \tag{3.14}$$

But we have that $|\rho_i(x_i)| = |\sigma(x_i + p_i) - \sigma(p_i)| \leq K|x_i|$ since

$$0 \leq ((\sigma(\xi_2) - \sigma(\xi_1))/(\xi_2 - \xi_1)) \leq K,$$

for all $\xi_1, \xi_2 \in \mathbb{R}$, and $x_i\rho_i(x_i) \geq 0$ for the same reason. Thus, $\rho_i^2(x_i) \leq Kx_i\rho(x_i)$ and, therefore, $(1/K)\rho^T(x)\rho(x) \leq x^T\rho(x)$ or

$$x^T\rho(x) - (1/K)\rho^T(x)\rho(x) \leq 0. \tag{3.15}$$

If we add and subtract the quantity $2\gamma\Delta\Lambda(x^T\rho(x) - (1/K)\rho^T(x)\rho(x))$ to (3.14), we have

$$\begin{aligned}
\frac{dV(x(t))}{dt} &= x^T(P(-\Lambda) + (-\Lambda)P)x \\
&\quad -2x^T(P(-B) - \alpha\Delta(-\Lambda) - \gamma)\rho(x) \\
&\quad -\rho^T(x)\left[\alpha((\Delta(-B))^T + \Delta(-B)) + 2(1/K)\gamma\Delta\Lambda\right]\rho(x) \\
&\quad -2\gamma\Delta\Lambda(x^T\rho(x) - (1/K)\rho^T(x)\rho(x)).
\end{aligned} \tag{3.16}$$

Now, we use Lemma 3.13 to show that the previous expression is always negative definite. Indeed, the first three terms form a square plus a positive semidefinite quadratic form, and the last term is always negative. We have that there exist $\alpha > 0, \gamma > 0$, and the following $n \times n$ matrices: Q invertible; R arbitrary; P, S symmetric and positive definite that satisfy (3.13). Equation 3.16 then becomes

$$\begin{aligned} \frac{dV(x(t))}{dt} &= -x^T QQ^T x + 2x^T QR\rho(x) - \rho^T(x)R^T R\rho(x) - \rho^T(x)S\rho(x) \\ &\quad -2\gamma(x^T\rho(x) - (1/K)\rho^T(x)\rho(x)), \\ &= -(x^T Q - 2R\rho(x))^T(x^T Q - 2R\rho(x)) - \rho^T(x)S\rho(x) \\ &\quad -2\gamma\Delta\Lambda(x^T\rho(x) - (1/K)\rho^T(x)\rho(x)). \end{aligned} \tag{3.17}$$

We want to show that the sum of these three terms is negative definite. The first term is always less than or equal to zero since it is the opposite of a square, the second term also since S is positive definite, and the third term also by (3.15). The sum of the three terms is therefore always negative. But we want the sum to be negative definite ($\dot{V}(x) \leq 0$ for all $x; \dot{V}(x) = 0 \iff x = 0$). We proceed by contradiction. Suppose $\dot{V}(x) = 0$, then the three terms have to be zero. Thus, $\rho^T(x)S\rho(x) = 0$, which implies $\rho(x) = 0$ since S is positive definite. The first term then degenerates to $-x^T QQ^T x$. Since QQ^T is also positive definite, this term is zero only if $x = 0$. Thus, the derivative of the Lyapunov function is negative definite. It can only be zero at $x = 0$. This reasoning implies that p is the only equilibrium point of the neural network if $-B + (1/K)\Lambda$ is Lyapunov diagonally stable and that it is globally asymptotically stable. □

An alternative to the use of Lemma 3.13 is to set up a matrix problem whose solution gives a set of parameters for which our Lur'e–Postnikov candidate is indeed a Lyapunov function. Instead of trying to find a condition that guarantees the existence of a solution to the problem presented in the Lemma, we use the following procedure. Each time we want to check the stability of a particular network, we evaluate the derivative of the Lyapunov function as a function of the parameters P, Δ, α depending on the weights Λ, B of the neural network. Bounding the behavior of the derivative of the previous Lyapunov function to get rid of the dependence on $\rho(x)$, we get a linear matrix inequality [18]. If this linear matrix inequality admits a solution, we get a set of parameters of the Lur'e–Postnikov function that guarantees stability. There are powerful computational technique for solving linear matrix inequalities, which makes this approach a practical tool for checking stability of neural networks. Suykens [137] has developped this approach further to apply it to dynamical neural networks of the form

$$\dot{x} = Gx + C\sigma(Fx + B\sigma(Ax)),$$

which we can write as a Lur'e system with a nonlinearity $\psi(y) = \sigma(B\sigma(y))$. This network does not have any bias, so the origin is always a fixed point of the

system (we therefore do not have to discuss the existence of fixed points), but it is a continuous-time multilayer perceptron. The Lur'e–Postnikov procedure gives then a linear matrix inequality whose solution guarantees stability of this neural network.

3.7 Conclusion

This chapter gave a review of the convergence and stability of dynamical neural networks. We first presented different type of neural networks described by systems of ODEs. Such representations meant that we could analyze these networks with the methods of dynamical system theory. Because dynamical system theory is such a large field, we limited our scope by considering only the convergence and global asymptotic stability of neural networks. It was essential to study the stability properties of neural networks because stability is of utmost importance if we use neural networks for control applications, which is a major application of neural networks in engineering. Convergence (which we defined as the fact that every trajectory converges to some stable state, albeit there may be more than one such state) was a natural generalization of stability and we therefore chose it as our second theme. Convergence may also become more important as control theory moves away from the practices inherited from linear control theory and reaches towards new nonlinear designs.

We began by defining the concepts of convergence and stability, and by introducing the two major tools for their study: The second method of Lyapunov and the LaSalle invariance principle. We then used these tools to study a class of competitive systems (which are systems for which the growth of one variable inhibits the growth of the other variables) described by Cohen and Grossberg [26]. Their theorem is general and is applicable to different types of neural networks, although the required symmetry and positivity assumptions restrict its practical usefulness somewhat. We then reviewed Hirsch's study of cascaded systems [58] and found conditions under which the interconnection of a number of stable neural networks remains itself stable. Such criteria are important because many neural network architectures are built through interconnections.

From general convergence problems, we specialized to the problem of global asymptotic stability. We first considered a result based on linearizing the system at all points (not just at the equilibrium) and making sure that trajectories always get closer to each other. We then considered Lyapunov functions again. But, while Lyapunov functions are often developed on a case-per-case basis, we adopted the strategy of Lur'e–Postnikov Lyapunov functions [38], which is a reverse-engineering approach. Indeed, we started by giving ourselves an apparently arbitrary candidate Lyapunov function and then checked under what conditions it became a Lyapunov function for the system under consideration.

We saw that such a method was both practical and successful in the case of neural networks. The main message of this chapter is certainly that the methods from dynamical system theory for convergence and stability analysis are efficient tools for the study of neural networks.

Chapter 4

Representation of neural networks as Lotka–Volterra systems

In this chapter, we consider neural networks again as dynamical systems. But, instead of studying their asymptotic behavior (stability and convergence), we consider a series of nonlinear coordinate transformations that allow us to reinterpret their dynamics as equivalent to the dynamics of systems from a well-known class: the Lotka–Volterra systems. In Chapter 6, in contrast, we will investigate other types of changes of coordinates: linear transformations that map some continuous-time neural network with a hidden-layer into a neural network without hidden-layer and with an extra output map. The theme of change of coordinates in neural networks is central in this dissertation and we shall come back to it repeatedly.

4.1 Lotka–Volterra systems

We show here how to represent different types of neural networks (having the hyperbolic tangent or the logistic sigmoid as activation function) into Lotka–Volterra systems (also called predator-prey models) of the form:

$$\dot{z}_i = \lambda_i z_i + z_i \sum_{j=1}^{m} M_{ij} z_j, \qquad \text{with } j = 1, \ldots, m.$$

The corresponding Lotka–Volterra system will be of higher dimension than the original system but its n first variables have a behavior equivalent or identical

to that of the original system (if n is the number of variables of the original system).

Such a Lotka–Volterra representation of neural networks is of interest for several reasons. Lotka–Volterra systems are an important tool in mathematical biology for the modeling of competition between species and a classical subject of dynamical system theory [81, 82]. They have therefore been the object of intense scrutiny [140]. These systems have simple quadratic nonlinearities (not all quadratic terms are possible in each equation), which might be easier to analyze than the sigmoidal saturations of the neural networks. We thus expect that this representation will allow us to apply methods and results from the study of Lotka–Volterra systems to neural networks. We think, in particular, of the application of conditions for the global asymptotic stability of an equilibrium developed for quasi-monomial systems and Lotka–Volterra systems [49, 117] to neural networks. Also, early work by Grossberg [50] exploited one of the changes of variables that we present here to study convergence and oscillations in generalized competitive networks and generalized Lotka–Volterra systems. Furthermore, as we will show in Chapter 6 that dynamical neural networks can approximate arbitrary dynamical systems [131, 42] for any finite time because of their equivalence with dynamical single–hidden-layer perceptrons [101], our result also serves as a simple proof that Lotka–Volterra systems enjoy the same approximation property; and we might want to investigate them further from that point of view.

We have structured this chapter as follows. We consider first the simple case of a dynamical perceptron. We transform it into what we call a quasi-monomial form. This form allows us to transform the system further into a Lotka–Volterra system. Second, we consider the case of the dynamical perceptron with self-connections—that is, where each neuron has a linear connection to itself. We perform the same transformations first into a quasi-monomial form and then into a Lotka–Volterra system; we also discuss how to treat possible singularities of these transformations. Third, we show that the same transformations apply to much more general types of continuous-time recurrent neural networks with hyperbolic tangent or logistic activation functions. In the case of the logistic activation function, we see that the transformation does not present any singularity. Finally, we clarify what the exact constraints on the activation function are and we show that both the hyperbolic tangent and the logistic sigmoid satisfy these constraints.

4.2 Dynamical perceptrons

We consider first n-dimensional dynamical perceptrons:

$$\dot{x} = \sigma(Ax + a_0), \tag{4.1}$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, $a_0 \in \mathbb{R}^n$, and σ is a continuous sigmoidal function. We consider this type of system because its simplicity makes our calculations easier to follow, yet it permits to illustrate the essential points of our method. We do all developments for the special case of $\sigma(x) = \tanh(x)$. Another possibility is the use of the logistic sigmoid $\sigma(x) = 1/(1+e^{-x})$. We discuss in Section 4.6 what constrains the choice of the activation function. For the dynamical perceptron (4.1), choosing the logistic sigmoid would be meaningless since the vector field would then have positive components everywhere and the topology of the flow would be trivial (whereas the hyperbolic tangent guarantees the existence of an equilibrium). For the types of networks we describe in the following sections, this restriction to the hyperbolic tangent does not apply. But since the developments depend on the choice of the activation function, we limit ourselves to the hyperbolic tangent; all developments can be done similarly for the logistic sigmoid.

4.2.1 Sigmoidal change of coordinates

We first apply the change of coordinates $y = Ax + a_0$ (often called Sigma–Pi transformation [51]) and get the following equation:

$$\dot{y} = A\sigma(y).$$

As long as A is invertible (A is $n \times n$), we can directly revert the change of coordinates. Moreover, if A is singular, we in fact have an s-dimensional dynamics, where s is the rank of the matrix A. Indeed, if we look at two different trajectories $\bar{x}(t)$ and $\tilde{x}(t)$ of the system $\dot{x} = \sigma(Ax + a_0)$ having the property that their initial conditions $\bar{x}_0$ and $\tilde{x}_0$ are such that $A\bar{x}_0 = A\tilde{x}_0$, the initial conditions $\bar{y}_0 = A\bar{x}_0 + a_0$ and $\tilde{y}_0 = A\tilde{x}_0 + a_0$ for the system $\dot{y} = A\sigma(y)$ are equal. This means that $\bar{y}(t) = \tilde{y}(t)$ for all times. But the vector field of the initial system is $\dot{x} = \sigma(Ax + a_0) = \sigma(y)$, thus $\dot{\bar{x}}(t) = \dot{\tilde{x}}(t)$ for all times. This means that the trajectories $\bar{x}(t)$ and $\tilde{x}(t)$ through $\bar{x}_0$ and $\tilde{x}_0$ always remain parallel. Since the kernel of A is of dimension $n - s$, this means that the dynamics is in fact s-dimensional. If we define $\gamma(x_0) = x_0 - A^\dagger A x_0$ where $A^\dagger$ is the Moore–Penrose pseudoinverse of A, then $x(t) = A^\dagger(y(t) - a_0) + \gamma(x_0)$ with the initial condition $y_0 = Ax_0 + a_0$.

From here on, it is necessary to look at the equations coordinate by coordinate, so we have

$$\dot{y}_i = \sum_{j=1}^{n} A_{ij}\sigma(y_j), \qquad \text{with } i = 1, \ldots, n. \tag{4.2}$$

Now, we decide to apply the change of coordinates $z_i = \sigma(y_i)$; we get

$$\dot{z}_i = \sigma'(y_i)\dot{y}_i. \tag{4.3}$$

It is here that we make use of our special choice of activation function, $\sigma(y) = \tanh(y)$. The essential property of the activation function we need is that we can express its derivative as a polynomial of the activation function:

$$\sigma'(y) = \tanh'(y) = 1/\cosh^2(y) = 1 - \tanh^2(y) = 1 - \sigma^2(y). \tag{4.4}$$

Substituting (4.4) and (4.2) in Equation 4.3, we have

$$\dot{z}_i = (1 - z_i^2) \sum_{j=1}^{n} A_{ij} z_j. \tag{4.5}$$

We see that this form begins to resemble the Lotka–Volterra model except for the factor $(1 - z_i^2)$ instead of z_i. This means that the system is polynomial of degree three. However, the vector field of the system has now become a linear combination of products of powers of the variables, which is what we call a quasi-monomial form.

Example

We follow our developments at the hand of an example to clarify the procedure. We look at the case $n = 3$. The equations of the recurrent neural network are

$$\begin{cases} \dot{x}_1 &= \tanh(A_{11}x_1 + A_{12}x_2 + A_{13}x_3 + a_{01}), \\ \dot{x}_2 &= \tanh(A_{21}x_1 + A_{22}x_2 + A_{23}x_3 + a_{02}), \\ \dot{x}_3 &= \tanh(A_{31}x_1 + A_{32}x_2 + A_{33}x_3 + a_{03}). \end{cases}$$

We can check that the equations become, after the affine and sigmoidal changes of coordinates,

$$\begin{cases} \dot{z}_1 &= (1 - z_1^2)(A_{11}z_1 + A_{12}z_2 + A_{13}z_3), \\ \dot{z}_2 &= (1 - z_2^2)(A_{21}z_1 + A_{22}z_2 + A_{23}z_3), \\ \dot{z}_3 &= (1 - z_3^2)(A_{31}z_1 + A_{32}z_2 + A_{33}z_3). \end{cases} \tag{4.6}$$

4.2.2 Transformation into quasi-monomial forms

By quasi-monomial form, we mean a dynamical system for which we can express the component of the vector field for each variable as the product of this variable by a linear combination of products of powers of all variables.

Definition 4.1. A *quasi-monomial form* is a dynamical system of the form

$$\dot{z}_i = z_i \Big(\sum_{j=1}^{r} \bar{A}_{ij} \prod_{k=1}^{n} z_k^{B_{jk}}\Big), \quad \text{with } i = 1, \ldots, n. \tag{4.7}$$

The number of different products of powers of the variables is r and we define $\bar{A} = \left[\bar{A}_{ij}\right] \in \mathbb{R}^{n\times r}$ and $B = \left[B_{jk}\right] \in \mathbb{R}^{r\times n}$. It is important to note that the exponents B_{jk} can be real values (they are not restricted to natural numbers), hence the name of quasi-monomial form. Now, to reach this form, we write Equation 4.5 as

$$\dot{z}_i = z_i \left[\sum_{j=1}^{n} A_{ij} (z_i^{-1} z_j - z_i z_j) \right]. \tag{4.8}$$

We then need to collect all possible monomials and use their exponents to construct the matrix B and collect all the coefficients of the monomials correspondingly to form the matrix $\bar{A}$. We illustrate it first on the following example and then do it systematically in the next section. Once we have the quasi-monomial form, Brenig [21] has shown how to transform such a quasi-monomial form into a Lotka–Volterra system. Other authors [115, 49] have discovered similar transformations independently.

Example (cont'd)

To illustrate this somewhat obtuse notation (4.7), we take again the case $n = 3$. Expanding the expression (4.6), we get

$$\left\{ \begin{array}{lcl} \dot{z}_1 &=& z_1(A_{11} - A_{11}z_1^2 + A_{12}z_1^{-1}z_2 - A_{12}z_1z_2 + A_{13}z_1^{-1}z_3 + A_{13}z_1z_3) \\ \dot{z}_2 &=& z_2(A_{21}z_1z_2^{-1} - A_{21}z_1z_2 + A_{22} - A_{22}z_2^2 + A_{23}z_2^{-1}z_3 - A_{23}z_2z_3) \\ \dot{z}_3 &=& z_3(A_{31}z_1z_3^{-1} - A_{31}z_1z_3 + A_{32}z_2z_3^{-1} - A_{32}z_2z_3 + A_{33} - A_{33}z_3^2) \end{array} \right.$$

We see that there are $2n^2 = 18$ terms, but only $r = (3n^2 - n + 2)/2 = 13$ different monomials. The matrix B is then

$$B = \begin{pmatrix} 2 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 2 & 0 & 1 & 0 & 1 & -1 & -1 & 0 \end{pmatrix}^T;$$

while the matrix $\bar{A}$ is

$$\bar{A} = \begin{pmatrix} -A_{11} & -A_{12} & -A_{13} & 0 & 0 & 0 & A_{12} & A_{13} & 0 & 0 & 0 & 0 & A_{11} \\ 0 & -A_{21} & 0 & -A_{22} & -A_{23} & 0 & 0 & 0 & A_{21} & A_{23} & 0 & 0 & A_{22} \\ 0 & 0 & -A_{31} & 0 & -A_{32} & -A_{33} & 0 & 0 & 0 & 0 & A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

How should we interpret these matrices? If we look at the first row of B, we have a row vector $(2, 0, 0)$ (note that the notation above uses a transposition); and if we look at the first column of $\bar{A}$, we have a column vector $(-A_{11}, 0, 0)^T$. These vectors mean that the expression for the system contains a monomial of the form $z_1^2 z_2^0 z_3^0 = z_1^2$; and that it appears in the first

equation with a coefficient of $-A_{11}$ and is absent from the second and third equations. Checking this mechanism for the whole matrices, we indeed have $\dot{z}_i = z_i\Big(\sum_{j=1}^{13} \bar{A}_{ij} \prod_{k=1}^{3} z_k^{B_{jk}}\Big)$ with $i = 1, 2, 3$.

4.2.3 Computation of the matrices of the quasi-monomial forms

We derive here the structure of the matrices $\bar{A}$ and B of the quasi-monomial form for the dynamical perceptron in a systematic fashion. Again, we write the equations as

$$\dot{z}_i = z_i \left[\sum_{j=1}^{n} A_{ij} (z_i^{-1} z_j - z_i z_j) \right] . \tag{4.9}$$

We see that the expression between brackets contains $2n$ monomials; for the n differential equations this gives a total of $2n^2$ terms. There are two types of monomials: $z_i z_j$ and $z_i^{-1} z_j$. The first type of monomial can take two different forms: z_i^2 (when $i = j$) or $z_i z_j$ with $j > i$ (when $i \neq j$). There are n monomials of the first form and $n(n-1)/2$ monomials of the second form. The second type of monomials can also take two different forms: 1 (when $i = j$) or $z_i^{-1} z_j$ (when $i \neq j$). There is one monomial of the first form and $n(n-1)$ monomials of the second form. These different contributions give a total of $(3n^2 - n + 2)/2$ different monomials to share among $2n^2$ terms. Carefully collecting all these terms into row vectors of exponents to form the matrix B and all their corresponding coefficients into column vectors to form the matrix $\bar{A}$, we can write these matrices as

$$B = \left[C^{(1)}; C^{(2)}; \ldots; C^{(n)}; D^{(1)}; D^{(2)}; \ldots D^{(n)}; O^{1\times n} \right], \tag{4.10}$$

$$\bar{A} = \left[P^{(1)} P^{(2)} \ldots P^{(n)} R^{(1)} R^{(2)} \ldots R^{(n)} V \right], \tag{4.11}$$

(recall that we use the semicolon to stack up submatrices) where we define the submatrices $C^{(i)}, D^{(i)}, O^{1\times n}$, and $P^{(i)}, R^{(i)}, V$ as follows. The matrix $O^{n\times m}$ is a matrix of zeros of dimension $n \times m$; later, we will also use $I^{n\times m}$ to denote a diagonal matrix of ones of dimension $n \times m$. The matrix $C^{(i)}$ represents all the monomials of the form z_i^2 and $z_i z_j$ (with $j > i$) and thus the dimension of the matrix depends on the index i with the matrix having $n - i + 1$ rows and n columns. We can write it as

$$C^{(i)} = \left(\begin{array}{c|c|c} 0 \quad \cdots \quad 0 & 2 & 0 \quad \cdots \quad 0 \\ & 1 & \\ O^{(n-i)\times(i-1)} & 1 & I^{(n-i)\times(n-i)} \\ & \vdots & \\ & 1 & \end{array} \right) .$$

Each of its rows is associated with the corresponding column of the matrix $P^{(i)}$, which collects all the coefficients of the monomials z_i^2 and $z_i z_j$ (with $j > i$). Such a matrix has again variable dimension with n rows and $n-i+1$ columns. We can write it as

$$P^{(i)} = \left(\begin{array}{c|cccc} O^{(i-1)\times(n-i+1)} & & & & \\ \hline -A_{ii} & -A_{i,(i+1)} & -A_{i,(i+2)} & \cdots & -A_{in} \\ 0 & -A_{(i+1),i} & 0 & \cdots & 0 \\ \vdots & 0 & -A_{(i+2),i} & & \vdots \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & \cdots & -A_{ni} \end{array} \right).$$

Turning back to the matrix of exponents B, we collect all the exponents of the monomials of the form $z_i^{-1} z_j$ (with $j \neq i$) in the matrix $D^{(i)}$ which has always $n-1$ rows and n columns:

$$D^{(i)} = \left(\begin{array}{c|c|c} I^{(i-1)\times(i-1)} & -1 & O^{(i-1)\times(n-i)} \\ & \vdots & \\ \hline O^{(n-i)\times(i-1)} & \vdots & I^{(n-i)\times(n-i)} \\ & -1 & \end{array} \right).$$

We collect the coefficients of these monomials in the corresponding columns of $R^{(i)}$, which also has $n-1$ rows and n columns:

$$R^{(i)} = \left(\begin{array}{c} O^{(i-1)\times(n-1)} \\ \hline A_{i1} \quad \cdots \quad A_{i,(i-1)} \quad A_{i,(i+1)} \quad \cdots \quad A_{in} \\ \hline O^{(n-i)\times(n-1)} \end{array} \right).$$

We are now only left with the constant monomial (which we represent by the row vector $O^{1\times n}$) and its coefficients, which we collect in the column vector

$$V = (A_{11}, \ldots, A_{nn})^T.$$

Taking all this notation into consideration, we can represent the matrices A and B according to Equations 4.10 and 4.11.

Example

We now present the different matrices for the case of a four-dimensional system. This example clarifies how we have collected the different terms in the matrices

$\bar{A}$ and B. The submatrices are $C^{(1)}, C^{(2)}, C^{(3)}, C^{(4)}$ and $D^{(1)}, D^{(2)}, D^{(3)}, D^{(4)}$ for B:

$$C^{(1)} = \left(\begin{array}{c|ccc} 2 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right), \quad C^{(2)} = \left(\begin{array}{c|c|cc} 0 & 2 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 1 \end{array}\right),$$

$$C^{(3)} = \left(\begin{array}{cc|c|c} 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right), \quad C^{(4)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 2 \end{array}\right);$$

$$D^{(1)} = \left(\begin{array}{c|ccc} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{array}\right), \quad D^{(2)} = \left(\begin{array}{c|c|cc} 1 & -1 & 0 & 0 \\ \hline 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{array}\right),$$

$$D^{(3)} = \left(\begin{array}{cc|c|c} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ \hline 0 & 0 & -1 & 1 \end{array}\right), \quad D^{(4)} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{array}\right).$$

We have decomposed the matrix A into the submatrices $P^{(1)}, P^{(2)}, P^{(3)}, P^{(4)}$, $R^{(1)}, R^{(2)}, R^{(3)}, R^{(4)}$, and V:

$$P^{(1)} = \left(\begin{array}{c|ccc} -A_{11} & -A_{12} & -A_{13} & -A_{14} \\ \hline 0 & -A_{21} & 0 & 0 \\ 0 & 0 & -A_{31} & 0 \\ 0 & 0 & 0 & -A_{41} \end{array}\right), \quad P^{(3)} = \left(\begin{array}{cc} 0 & 0 \\ 0 & 0 \\ -A_{33} & -A_{34} \\ \hline 0 & -A_{43} \end{array}\right),$$

$$P^{(2)} = \left(\begin{array}{c|cc} 0 & 0 & 0 \\ -A_{22} & -A_{23} & -A_{24} \\ \hline 0 & -A_{32} & 0 \\ 0 & 0 & -A_{42} \end{array}\right), \qquad P^{(4)} = \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ -A_{44} \end{array}\right);$$

$$R^{(1)} = \left(\begin{array}{ccc} A_{12} & A_{13} & A_{14} \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right), \quad R^{(2)} = \left(\begin{array}{ccc} 0 & 0 & 0 \\ \hline A_{21} & A_{23} & A_{24} \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right),$$

$$R^{(3)} = \left(\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline A_{31} & A_{32} & A_{34} \\ \hline 0 & 0 & 0 \end{array}\right), \quad R^{(4)} = \left(\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline A_{41} & A_{42} & A_{43} \end{array}\right);$$

$$V = (A_{11}, A_{22}, A_{33}, A_{44})^T.$$

4.3 Transformation of the quasi-monomial forms into Lotka–Volterra systems

From this quasi-monomial form, we can get a Lotka–Volterra system by a single change of coordinates. The whole transformation relies now on a trick: We look at each product of powers of the variables that appear in the quasi-monomial form as a new variable of our dynamical system. As mentioned in the introduction, we define Lotka–Volterra systems as follows (note that we abandon the notation for the linear term; we can always replace this term by making sure that one of the variables remains constant, so that one of the terms produces the necessary linear contribution).

Definition 4.2. A *Lotka–Volterra system* is a dynamical system of the form

$$\dot{z}_i = z_i \sum_{j=1}^{m} M_{ij} z_j, \qquad \text{with } j = 1, \dots, m. \tag{4.12}$$

We have the following theorem [21].

Theorem 4.1 (Brenig). *For any quasi-monomial form (4.7) of dimension n and with r terms, there exists a Lotka–Volterra system (4.12) of dimension $n+r$ for which the dynamics of the first n variables is identical to the dynamics of the quasi-monomial form.*

Proof. With r the number of different monomials in the quasi-monomial form, we define the supplementary variables $z_{n+1}, \dots, z_{n+r}$ as

$$z_{n+k} = \prod_{l=1}^{n} z_l^{B_{kl}}, \qquad \text{with } k = 1, \dots, r.$$

If we now look at the system of $n + r$ variables $\dot{z}_i$, taking into account these new definitions, we have for the first n variables

$$\dot{z}_i = z_i \sum_{j=1}^{r} \bar{A}_{ij} z_{n+j}, \qquad \text{with } i = 1, \dots, n.$$

Because of our special definition of the new variables, many simplifications occur in the expression of their time derivative, as seen in Equation 4.13. We first use the formula for the derivative of a product (4.14). Then we try to reconstruct the original variables since they are almost preserved by the derivation (4.15); the necessary correction leads to the appearance of a logarithmic derivative (4.16), which in turn transforms the exponent of the monomials into multiplicative coefficients (4.17). Lastly, the remaining correction produces the

new expression of the vector fields of the first n variables, except for the disappearance of the multiplicative variable at the front (4.18). Writing these computations down, we have, for the r supplementary variables,

$$\begin{aligned}
\dot{z}_{n+j} &= (\prod_{k=1}^{n} z_k^{B_{jk}})' \qquad \text{with } j = 1, \dots, r && (4.13)\\
&= \sum_{l=1}^{n} (\prod_{\substack{k=1\\k\neq l}}^{n} z_k^{B_{jk}})(z_l^{B_{jl}})' && (4.14)\\
&= \sum_{l=1}^{n} (z_{n+j}/z_l^{B_{jl}})(z_l^{B_{jl}})' && (4.15)\\
&= z_{n+j} \sum_{l=1}^{n} (\ln(z_l^{B_{jl}}))' && (4.16)\\
&= z_{n+j} \sum_{l=1}^{n} B_{jl}\dot{z}_l/z_l && (4.17)\\
&= z_{n+j} \sum_{l=1}^{n} B_{jl} \sum_{m=1}^{r} \bar{A}_{lm} z_{n+m} && (4.18)\\
&= z_{n+j} \sum_{m=1}^{r} ((\sum_{l=1}^{n} B_{jl}\bar{A}_{lm}) z_{n+m}). && (4.19)
\end{aligned}$$

Thus, we can regroup all these terms by defining the $(n+r)\times(n+r)$ matrix M as

$$M = \begin{pmatrix} O^{n\times n} & \bar{A} \\ O^{r\times n} & B\bar{A} \end{pmatrix},$$

since $\sum_{l=1}^{n} B_{jl}\bar{A}_{lm} = (B\bar{A})_{jm}$; and the total system becomes then of the form

$$\dot{z}_i = z_i \sum_{j=1}^{n+r} M_{ij} z_j, \qquad \text{with } i = 1, \dots, n+r,$$

which is precisely an $n+r$-dimensional Lotka–Volterra system! □

In the case of the dynamical perceptron, we have now the following corollary.

Corollary 4.2. *For any dynamical perceptron (4.1) of dimension n, there exists a Lotka–Volterra system (4.12) of dimension $(3n^2+n+2)/2$ for which the dynamics of the first n variables is equivalent to the dynamics of the dynamical perceptron.*

Proof. This corollary is a direct consequence of our transformation of the dynamical perceptron into a quasi-monomial form and of Theorem 4.1. The developments of Section 4.2.3 transform the dynamical perceptron into a quasi-monomial form with matrices (4.10) and (4.11) whose respective dimensions are $r \times n$ and $n \times r$ (with $r = (3n^2 - n + 2)/2$). After this first transformation, Theorem 4.1 associates a Lotka–Volterra system of dimension $n+r$ to the quasi-monomial form. Supposing we had a solution $z(t) = (z_1(t), z_2(t), \ldots, z_{n+r}(t))^T$ of the equivalent Lotka–Volterra system, we can then retrieve the behavior of the original system by transforming back the n first variables (if the matrix A is invertible):

$$x(t) = A^{-1}(\sigma^{-1}((z_1(t), \ldots, z_n(t))^T) - a_0).$$

If the matrix is not invertible, we can apply the argument from Section 4.2 to retrieve the solution. □

4.4 Dynamical perceptrons with self-connections

One criticism that we can formulate about the dynamical perceptron is that its global behavior is often similar to the behavior around its fixed point. Although we can obtain complex dynamics by a careful choice of parameters, this is a delicate operation. The dynamics of such systems appear to be essentially limited. One way to obtain complex oscillatory behaviors more easily is to add a proportional self-connection to each variable:

$$\dot{x}_i = -\lambda_i x_i + \sigma(\sum_{j=1}^{n} A_{ij} x_j + a_{0i}). \tag{4.20}$$

The reason why we can obtain oscillatory behaviors more easily is that we can choose A to make one of the fixed points unstable and then choose the λ_i sufficiently large so that the behavior of the system always remain bounded. A further remark is that this kind of system can have multiple isolated equilibria, while the previous system either had to have a single equilibrium or a connected set of equilibria. The dynamics of the systems with inhibitory connections is thus richer. Also, the activation function for this system is not limited to the hyperbolic tangent anymore; in Section 4.6, we discuss other possible choices and conclude that we could use the logistic sigmoid as well.

4.4.1 Setup of the quasi-monomial forms

We represent this system by a Lotka–Volterra system. The first step is to define the following auxiliary variables:

$$w_i = \sigma(\sum_{j=1}^{n} A_{ij} x_j + a_{0i}).$$

Looking at the derivative of w, we then have the following dynamics for the system of x and w variables:

$$\begin{array}{rcl} \dot{x}_i & = & w_i - \lambda_i x_i, \\ \dot{w}_i & = & (1 - w_i^2) \sum_{j=1}^{n} A_{ij} (w_j - \lambda_j x_j). \end{array} \tag{4.21}$$

This dynamics is essentially in quasi-monomial form and we can rewrite it as

$$\begin{array}{rcl} \dot{x}_i & = & x_i(-\lambda_i + x_i^{-1} w_i), \\ \dot{w}_i & = & w_i \sum_{j=1}^{n} (-\lambda_j A_{ij} x_j w_i^{-1} - \lambda_j A_{ij} x_j w_i + A_{ij} w_i^{-1} w_j + A_{ij} w_i w_j). \end{array}$$

In the next subsection, we show how to write the matrices $\bar{A}$ and B of the quasi-monomial form of this system. Again, we just have to carefully collect all the exponents of the monomials in the matrix B and all their coefficients in the matrix $\bar{A}$. After this transformation into quasi-monomial form, the transformation of Section 4.3 directly applies in the same fashion except that the variable x is directly part of the Lotka–Volterra system and does not need to be retrieved by a change of coordinates (we pay this advantage by having a larger Lotka–Volterra system than for the dynamical perceptron).

4.4.2 Matrices of the quasi-monomial forms for dynamical perceptrons with self-connections

We recall that, for the system (4.20), the quasi-monomial form is

$$\begin{array}{rcl} \dot{x}_i & = & x_i(-\lambda_i + x_i^{-1} w_i), \\ \dot{w}_i & = & w_i(\sum_{j=1}^{n} (-\lambda_j A_{ij} x_j w_i^{-1} - \lambda_j A_{ij} x_j w_i + A_{ij} w_i^{-1} w_j + A_{ij} w_i w_j)). \end{array} \tag{4.22}$$

To use the quasi-monomial formalism in the same way as in the previous section, we need to unify our formalism and use a single set of variables. We therefore define the following variable $u = (x; w)$:

$$\begin{array}{rcll} u_i & = & x_i, & \text{with } i = 1, \ldots, n; \\ u_{n+i} & = & w_i, & \text{with } i = 1, \ldots, n. \end{array}$$

If the matrices $\bar{A}^{(\lambda)}$ and $B^{(\lambda)}$ contain respectively all the coefficients and all the exponents of the different monomials, we can write the dynamics of u in quasi-monomial form:

$$\dot{u}_i = u_i\Big(\sum_{j=1}^{m} \bar{A}_{ij}^{(\lambda)} \prod_{k=1}^{2n} u_k^{B_{jk}^{(\lambda)}}\Big).$$

To build the matrices $\bar{A}^{(\lambda)}$ and $B^{(\lambda)}$, we have to look at all the monomials in (4.22). If we first focus on the terms $w_i^{-1}w_j$ and w_iw_j, which already appeared in Section 4.2.2, we see that we have exactly the same contribution in the matrix $B^{(\lambda)}$ as in the case of the dynamical perceptron. We have the same cancellation of the factors of the terms in $w_i^{-1}w_j$ or w_iw_j as in Section 4.2.2. They contribute to the matrix $B^{(\lambda)}$ with a first submatrix equal to B. Note that the left part of the matrix $B^{(\lambda)}$ is for powers of x while the right part is for powers of w. The next blocks take respectively into account the monomials of the form $x_i^{-1}w_i, x_iw_j^{-1}$, and x_iw_j. None of these terms ever add up or cancel out, which explains the plain diagonal and column structure of these submatrices. We have

$$B^{(\lambda)} = \left(\begin{array}{c|c} O^{r\times n} & B \\ \hline -I^{n\times n} & I^{n\times n} \\ \hline I^{n\times n} & -K^{(1)} \\ \vdots & \vdots \\ I^{n\times n} & -K^{(n)} \\ \hline I^{n\times n} & K^{(1)} \\ \vdots & \vdots \\ I^{n\times n} & K^{(n)} \end{array}\right),$$

where we define $K^{(i)}$ as

$$K^{(i)} = \left(\begin{array}{c|c|c} & 1 & \\ O^{n\times(i-1)} & \vdots & O^{n\times(n-i)} \\ & 1 & \end{array}\right).$$

Similarly, we collect all the coefficients of the monomials in $\bar{A}^{(\lambda)}$. We place the contributions relating to the equations for the x variables in the top part of the matrix while we place the contributions for w in the bottom part of the matrix. The contributions for the monomials $w_i^{-1}w_j$ and w_iw_j are exactly the same for the variables w as before; the first bottom submatrix is therefore equal to A. These monomials have no influence on the variables x and the first top submatrix is therefore equal to zero; except in the case where $w_i^{-1}w_j = 1$ corresponding to the $-\lambda_i$ terms in the equations for x, which generate a

column of $-\lambda_i$ in the otherwise empty matrix Q. The other submatrices take correspondingly into account the coefficients of the monomials of the form $x_i^{-1}w_i, x_i w_j^{-1}$, and $x_i w_j$. We thus have

$$\bar{A}^{(\lambda)} = \left(\begin{array}{c|c|ccc|ccc} Q^{n\times r} & I^{n\times n} & O^{n\times n} & \cdots & O^{n\times n} & O^{n\times n} & \cdots & O^{n\times n} \\ \hline A & O^{n\times n} & -L^{(1)} & \cdots & -L^{(n)} & L^{(1)} & \cdots & L^{(n)} \end{array}\right),$$

where we define Q and $L^{(i)}$ as

$$Q = \left(\begin{array}{c|c} O^{n\times(r-1)} & \begin{array}{c} -\lambda_1 \\ \vdots \\ -\lambda_n \end{array} \end{array}\right), \qquad L^{(i)} = \left(\begin{array}{c} O^{(i-1)\times n} \\ \hline A_{i1}\lambda_1 \quad \cdots \quad A_{in}\lambda_n \\ \hline O^{(n-i)\times n} \end{array}\right).$$

The following corollary is an immediate consequence of the previous developments (Sect. 4.4 and 4.4.2) and of our main theorem.

Corollary 4.3. *For any dynamical perceptron with self-connections (4.20) of dimension n, there exists a Lotka–Volterra system (4.12) of dimension $(7n^2 + n + 2)/2$ for which the dynamics of the first n variables is identical to the dynamics of the perceptron.*

The structure of the matrices $\bar{A}^{(\lambda)}$ and $B^{(\lambda)}$ shows that the dimension of the quasi-monomial form is $2n$ and the number of monomials is $r^{(\lambda)} = (3n^2 - n+2)/2 + n + 2n^2$ (where n is the dimension of the dynamical perceptron with self-connections), which added together give the dimension of the corresponding Lotka–Volterra system. Note that, while in Corollary 4.2 the dynamics of the first n variables was equivalent to the dynamics of the dynamical perceptron by a smooth transformation, the dynamics the first n variables of the Lotka–Volterra system is here identical to the dynamics of the dynamical perceptron with self-connections. But, again, the Lotka–Volterra system is here about twice as large as the one in Corollary 4.2.

4.4.3 Singularities of quasi-monomial transformations

A difficulty arises with the transformation of the quasi-monomial forms into Lotka–Volterra systems. Consider, as an example, the dynamical perceptron with self-connections (4.20). We represent it by the extended system (4.21) or by the quasi-monomial form (4.22). The extended system and the quasi-monomial form are equal. But we have put x_i and w_i factors in evidence in (4.22), which causes the formal appearance of x_i^{-1} and w_i^{-1} factors. This structure is only formal at this point as these factors all cancel out. However, when transforming the quasi-monomial form into a Lotka–Volterra system, we consider the quasi-monomials as new variables. In our case, we have three new sets of variables of the form $x_i^{-1}w_i, x_j w_i^{-1}$, and $w_i^{-1}w_j$. The fact that these

quasi-monomials contain inverses of the variable implies that, when any of the variables goes through zero, the quasi-monomials that contain it go to infinity. For the Lotka–Volterra system, this means that some of the supplementary variables become infinite in finite time. The vector field of the Lotka–Volterra system is not complete and the system suffers from finite escape time. The transformation that maps the variables of the quasi-monomial form into the variables of the Lotka–Volterra system is discontinuous.

Depending on the question we study, this discontinuity may or may not be a problem. For numerical questions, it is often possible to make an additional translation of the variables of the original system such that the behavior under study remains in the positive quadrant. None of the variables then become zero during the time interval for which we study the system; thus, the quasi-monomials never become infinite. This translation depends on the initial condition we choose for the original system. But numerical questions do not seem directly relevant to our analysis of dynamical neural networks because the Lotka–Volterra representation has a number of variables that is proportional to the square of the number of variables of the quasi-monomial forms. Therefore, it seems easier to simulate the original system than to simulate the Lotka–Volterra representation.

However, the problem may not be as severe as it appears at first. Although the variables that we add to the quasi-monomial form may blow up in finite time, they do not have an influence on the first variables of the Lotka–Volterra system, which represent the behavior of the original system. Taking the quasi-monomial $x_i^{-1} w_i$ in (4.22), we see that this quasi-monomial becomes infinite when $x_i = 0$. In the first equation of (4.22), this infinite quasi-monomial is multiplied by x_i, which is zero. In fact, the factors x_i^{-1} and x_i cancel out and force the infinity to disappear from the first equation of (4.22). What happens is that, when one of the original variables goes through zero, some of the quasi-monomials blow up to infinity; but these quasi-monomials come directly back from infinity as soon as the variable has gone through zero. Again, depending on the question under study, this behavior may or may not cause a problem. For the analysis of the convergence and stability of dynamical neural networks through Lotka–Volterra representations, we expect that this phenomena will cause some difficulties for the application of existing theorems.

It is possible to avoid this problem further by limiting the choice of the activation function to the logistic sigmoid. Indeed, the main property of the logistic sigmoid $\sigma(\xi) = 1/(1 + e^{-\xi})$ is that

$$\sigma'(\xi) = \frac{e^{-\xi}}{(1 + e^{-\xi})^2} = \sigma(\xi)(1 - \sigma(\xi)).$$

Because σ corresponds to the variables of the quasi-monomial form, we can put it in evidence directly so we can avoid inverses of the variables. We could not do this with the hyperbolic tangent because $\tanh'(\xi) = 1 - \tanh^2(\xi)$. Moreover,

we should note that there is a simple relation between the hyperbolic tangent and the logistic sigmoid. If we write $\sigma(\xi) = 1/(1+e^{-\xi})$ and $\rho(\xi) = \tanh(\xi)$, we have

$$\rho(\xi) = 2\sigma(\xi) - 1,$$

which we can check by a simple computation.

These tools can help us control the singularities in the Lotka–Volterra representations. We illustrate these arguments further in the coming section on the representation of dynamical multilayer perceptrons as Lotka–Volterra systems.

4.5 Dynamical multilayer perceptrons

The transformations we have showed for the dynamical perceptron and the dynamical perceptron with self-connections apply also to the dynamical multilayer perceptron. To avoid an unnecessary clutter of notation, we take the specific example of a two–hidden-layer continuous-time multilayer perceptron:

$$\dot{x} = \rho(C\rho(B\rho(Ax + a_0) + b_0) + c_0),$$

where we have n states, p hidden units in the first layer, q hidden units in the second layer, and the activation function ρ is the hyperbolic tangent. This means $x(t) \in \mathbb{R}^n, A \in \mathbb{R}^{p\times n}, B \in \mathbb{R}^{q\times p}, C \in \mathbb{R}^{n\times q}, a_0 \in \mathbb{R}^p, b_0 \in \mathbb{R}^q, c_0 \in \mathbb{R}^n$. The argument applies to many kinds of multilayer perceptron by doing the same type of computations again.

In the previous subsection, we showed that, if we use the hyperbolic tangent as our activation function, the transformation of the quasi-monomial form into a Lotka–Volterra system is discontinuous. To avoid this problem, we transform the network to use the logistic sigmoid as our activation function. If we denote by $I^{n\times 1}$ a column vector of ones, we have

$$\begin{aligned}\dot{x} &= \rho(C\rho(B2\sigma(2Ax + 2a_0) - BI^{p\times 1} + b_0) + c_0)\\ &= \rho(C2\sigma(4B\sigma(2Ax + 2a_0) + 2(b_0 - BI^{p\times 1})) - CI^{q\times 1} + c_0)\\ &= 2\sigma(4C\sigma(4B\sigma(2Ax + 2a_0) + 2(b_0 - BI^{p\times 1})) + 2(c_0 - CI^{q\times 1})) - I^{n\times 1}.\end{aligned}$$

By a trivial change of variables, we see that it is enough to consider systems of the form

$$\dot{x} = \sigma(C\sigma(B\sigma(Ax + a_0) + b_0) + c_0) + d_0$$

to cover the case of the two–hidden-layer continuous-time multilayer perceptron with the hyperbolic tangent as its activation.

If we write the equations for this new dynamical multilayer perceptron coordinate by coordinate, we get

$$\dot{x}_i = \sigma(\sum_{l=1}^{q} C_{il}\sigma(\sum_{k=1}^{p} B_{lk}\sigma(\sum_{j=1}^{n} A_{kj}x_j + a_{0k}) + b_{0l}) + c_{0i}) + d_{0i}, (i = 1, \ldots, n).$$

We define the following vector of p new variables: $y = \sigma(Ax + a_0)$. Again, we want to discuss the rank of the matrix A. If the matrix A is of rank n, then we can uniquely retrieve x from $y = \sigma(Ax + a_0)$. If the rank s of the matrix A is inferior to n, we have the same argument as in Section 4.2: Two initial conditions $\bar{x}_0$ and $\tilde{x}_0$ such that $A\bar{x}_0 = A\tilde{x}_0$ have the same dynamics in the new variables $\bar{y}(t) = \tilde{y}(t)$ for all times, thus $\bar{x}(t) = \sigma(C\sigma(B\bar{y}(t)+b_0)+c_0) = \tilde{x}(t)$ for all times. This relation means that the trajectories $\bar{x}(t)$ and $\tilde{x}(t)$ through $\bar{x}_0$ and $\tilde{x}_0$ always remain parallel. Since the kernel of A is of dimension $n - s$, this parallelism means that the dynamics is in fact s-dimensional. The dynamics of the variables $y_i = \sigma(\sum_{j=1}^{p} A_{ij}x_j + a_{0i})$ is thus

$$\begin{aligned}
\dot{y}_i &= \sigma'(\sum_{j=1}^{n} A_{ij}x_j + a_{0i})(\sum_{j=1}^{n} A_{ij}\dot{x}_j) \\
&= \sigma(\sum_{j=1}^{n} A_{ij}x_j + a_{0i})(1 - \sigma(\sum_{j=1}^{n} A_{ij}x_j + a_{0i}))(\sum_{j=1}^{n} A_{ij}\dot{x}_j) \\
&= y_i(1 - y_i)(\sum_{j=1}^{n} A_{ij}(\sigma(\sum_{l=1}^{q} C_{jl}\sigma(\sum_{k=1}^{p} B_{lk}y_k + b_{0l}) + c_{0j}) + d_{0j})).
\end{aligned}$$

We can remark that the form of the dynamics of y closely resembles that of the dynamics of x; but that the first hidden layer of the multilayer perceptron has disappeared. This encourages us to define, in the same fashion as before, a new vector of q variables: $z = \sigma(By + b_0)$. The dynamics of the variables $z_i = \sigma(\sum_{k=1}^{p} B_{ik}y_k + b_{0i})$ is thus

$$\begin{aligned}
\dot{z}_i &= \sigma'(\sum_{k=1}^{p} B_{ik}y_k + b_{0i})(\sum_{k=1}^{p} B_{ik}\dot{y}_k) = z_i(1 - z_i)(\sum_{k=1}^{p} B_{ik}\dot{y}_k) \\
&= z_i(1 - z_i)(\sum_{k=1}^{p} B_{ik}y_k(1 - y_k)(\sum_{j=1}^{n} A_{kj}(\sigma(\sum_{l=1}^{q} C_{jl}z_l + c_{0j}) + d_{0j}))).
\end{aligned}$$

Enthusiastically, we pursue further by defining a new vector of n variables: $w = \sigma(Cz + c_0)$. The dynamics of the variables $w_i = \sigma(\sum_{l=1}^{q} C_{il}z_l + c_{0i})$ is thus

$$\begin{aligned}
\dot{w}_i &= \sigma'(\sum_{l=1}^{q} C_{il}z_l + c_{0i})(\sum_{l=1}^{q} C_{il}\dot{z}_l) = w_i(1 - w_i)(\sum_{l=1}^{q} C_{il}\dot{z}_l), \\
\dot{w}_i &= w_i(1 - w_i)(\sum_{l=1}^{q} C_{il}z_l(1 - z_l)(\sum_{k=1}^{p} B_{lk}y_k(1 - y_k)(\sum_{j=1}^{n} A_{kj}(w_j + d_{0j})))).
\end{aligned}$$

Now, substituting the definitions for y, z, w in the previous equations and noting that, because of the same definitions, $\dot{x} = w + d_0$, we get the following system:

$$\dot{x}_i = w_i + d_{0i},$$

$$
\begin{aligned}
\dot{y}_i &= y_i(1-y_i)(\sum_{j=1}^{n} A_{ij}(w_j + d_{0j})), \\
\dot{z}_i &= z_i(1-z_i)(\sum_{k=1}^{p} B_{ik} y_k (1-y_k)(\sum_{j=1}^{n} A_{kj}(w_j + d_{0j}))), \\
\dot{w}_i &= w_i(1-w_i)(\sum_{l=1}^{q} C_{il} z_l (1-z_l)(\sum_{k=1}^{p} B_{lk} y_k (1-y_k)(\sum_{j=1}^{n} A_{kj}(w_j + d_{0j})))).
\end{aligned}
$$

The dynamics of the first n variables of this $(n+p+q+n)$-dimensional system (with appropriate initial conditions: $x_0 = x_0, y_0 = \sigma(Ax_0 + a_0), z_0 = \sigma(By_0 + b_0), w_0 = \sigma(Cz_0 + c_0)!$) is identical to the dynamics of the original system. Further, we see that we have written the ODE for each variable as a product of the variable itself by a linear combination of products of the variables (and a constant), which is precisely the definition of the quasi-monomial form. An exception is that the first equation is not in quasi-monomial form. But we know that $y = \sigma(Ax + a_0)$ so that we can retrieve x from the knowledge of y. We can thus restrict our attention to the system in y, z, and w, which is a Lotka–Volterra system. Furthermore we know that a fixed point of the original system ($\dot{x} = 0$) must be a fixed point of the Lotka–Volterra system because y, z, and w are static functions of x. Moreover, since $x = w + d_0$, $\dot{x} = 0$ is equivalent to $w = -d_0$. To find fixed points of the original systems, we have to find the fixed points of the Lotka–Volterra system for which $w = -d_0$. We therefore see that there is a strong relationship between the fixed points of the neural network and those of the Lotka–Volterra system, and that we have been capable to avoid the singularities associated with the use of the hyperbolic tangent.

The transformation of the equations for y, z, and w into a Lotka–Volterra system proceeds exactly in the same way as in Sections 4.2.3 and 4.4.2. However, writing down the matrices $\bar{A}$ and B is a lengthy and tedious procedure and does not bring any further insight. We therefore do not perform it here. This computation will be helpful only when we use a computer algebra system to analyze the resulting Lotka–Volterra system.

4.6 Choice of the activation function

We chose the activation function σ as $\sigma(\xi) = \tanh(\xi)$ for simplicity. But we are not directly limited to this activation function. The essential property that we used was the fact that $\sigma'(\xi) = 1 - \sigma^2(\xi)$. For the logistic sigmoid $\sigma(\xi) = 1/(1+e^{-\xi})$, we have $\sigma'(\xi) = \sigma(\xi)(1-\sigma(\xi))$ and we can do the calculations again without difficulty. The condition that the activation function has to satisfy is that its derivative be a sum of quasi-monomials of itself. If we do

not require saturation at plus and minus infinity, we can use $\sigma(\xi) = e^{\xi}$, but the system is not globally Lipschitz and the solution might blow-up in finite time, which is an annoyance. A square function $\sigma(\xi) = \xi^2$ suffers from the same problem. The hyperbola $\sigma(\xi) = 1/\xi$ gives $\sigma'(\xi) = -1/\sigma^2(\xi)$, but then the vector field is not even bounded, which is even worse. In general, we could consider the condition $\sigma'(\xi) = L(\sigma(\xi))$, where $L(.)$ is a finite Laurent series $L(y) = (1/y^m)\sum_{k=0}^{n} a_k y^k$, or even more generally, a Puiseux series. For the Laurent series, the solution of the differential equation is given by separation of variables $\frac{\sigma^m(\xi)}{L(\sigma(\xi))} d\sigma(\xi) = d\xi$. We can then expand the term $\sigma^m(\xi)/L(\sigma(\xi))$ as a polynomial plus a sum of simple fractions, and integrate them. This integration produces a rational function plus a sum of logarithmic terms and arctangent terms and $\sigma(\xi)$ is found by inverting this relation. This inversion does not lead to an explicit solution in general. If we further require that the vector field be globally Lipschitz to guarantee existence of the trajectories for all times, the possible solutions become scarce. From the most frequently used activation functions, we have been able to identify only the hyperbolic tangent and the logistic sigmoid as satisfying these conditions.

4.7 Conclusion

An important corollary of this representation of continuous-time recurrent multilayer perceptrons into higher-dimensional Lotka–Volterra systems is that we can use Lotka–Volterra systems to approximate arbitrarily closely the dynamics of any finite-dimensional dynamical system for any finite time. The argument for this universal approximation property is exactly the same as in [42] and says that, for any given finite time, the trajectories of two sufficiently close dynamical systems remain close (this is true only for a finite time). This statement is based on the classical use of the Gronwall inequality for dynamical systems and we shall come back to it in Chapter 6. It could therefore be interesting to use Lotka–Volterra systems as approximators for dynamical systems.

The restriction presented in Section 4.6 on the choice of the activation function (which is in practice limited to the hyperbolic tangent or the logistic sigmoid) is theoretically strong: There is no guarantee that systems with different choices of activation function have qualitatively the same behavior. Indeed, we will see (in Chapter 5) the example of systems with inputs for which the controllability properties can differ for different types of activation function, even if they are monotone increasing sigmoids. However, the practical limitation is fairly weak in the sense that the possible activation functions are the ones most commonly used and that we have universal approximation properties for dynamical systems.

We have presented techniques to represent different types of neural networks (dynamical perceptrons, dynamical perceptrons with self-connections, and multilayer perceptrons) into predator-prey (Lotka–Volterra) systems. These techniques relied on looking at the dynamics of the output of every neuron in the network and using the special properties of the derivative of the hyperbolic tangent or logistic sigmoid activation function to write the system as what we called a quasi-monomial form—that is a dynamical system where the vector field is a linear combination of products of powers of the state variables. Once we had this quasi-monomial form, we looked at the dynamics of these products of powers of the state variables to obtain directly a Lotka–Volterra system [21]. Our expectation is that these transformations will be useful to migrate techniques for the analysis of Lotka–Volterra systems into the field of dynamical neural networks—for example, techniques to study convergence or stability properties of dynamical systems, or discover first integrals of motion. The analysis of neural networks we gave in this chapter is one of the main contributions of this dissertation.

Chapter 5

Control theory of neural networks

In the previous chapter, we have considered continuous-time neural networks as dynamical systems. These systems were autonomous, which is a significant limitation if we want to move towards applications. We therefore wish to focus now on systems for which an external output drives the dynamics of the neural network. As a general class of nonlinear control systems, we consider systems of the form

$$\begin{cases} \dfrac{dx(t)}{dt} &= f(x(t), u(t)), \\ y(t) &= h(x(t)). \end{cases} \tag{5.1}$$

In this chapter, we will consider different types of neural networks depending on the problem we are studying. Examples include Hopfield-type networks with inputs,

$$\dot{x} = -\Lambda x + A\sigma(x) + u,$$

or state-space models whose dynamics is modeled by a perceptron,

$$\begin{cases} \dot{x} &= \sigma(Ax + Bu), \\ y &= Cx. \end{cases}$$

We start by presenting a solution to the problem of training a continuous-time neural network. We then move into control theory and discuss the concepts of observability and controllability applied to neural networks, giving conditions that guarantee these desirable properties. As we did in the Chapter 2, we consider the problem of finding equivalent sets of parameters for a network,

but this time for *dynamical* neural networks. Finally, we review the theory of discrete-time neural networks as these are more often used than continuous-time networks in practical applications.

5.1 Learning in continuous-time neural networks

One of the first concerns, if we want to use neural networks for system modeling, is to devise procedures by which we can teach them to reproduce the behavior of the system we want to model. In the terminology of control, this corresponds to system identification, although we do not deal here with many of the practical issues of system identification (such as persistence of the excitation, or robustness of the estimation). Typically, a number of input–output pairs $(u^{(1)}(t), z^{(1)}(t))$ with $t \in [t_0^{(1)}, t_1^{(1)}], \ldots, (u^{(r)}(t), z^{(r)}(t))$ with $t \in [t_0^{(r)}, t_1^{(r)}]$ will be the measurements representative of the system to model. We will then try to minimize the error that the neural network commits at its output when driven by the same inputs. In what follows, we consider only the case of a single input–output measurement. This limitation is not significant limitation since, in the case of multiple trajectories, we decompose the error function into the sum of the contributions of each trajectory; and we thus update the weights by taking the sum of the corrections provided by each trajectory separately.

We consider the case where the neural network is of the form:

$$\frac{dx(t)}{dt} = -\Lambda x(t) + \sigma(Ax(t)) + u(t). \tag{5.2}$$

The results we describe now also apply to other types of continuous-time networks. Two main types of learning algorithms exist for continuous-time neural networks: real-time recurrent learning (RTRL) [119, 148] and backpropagation through time (BPTT) [114, 125, 32]. These algorithms have discrete-time equivalents bearing the same name, but we will not consider them here. Narendra and Parthasarathy [108, 109] presented similar algorithms in the context of system identification and control under the name of dynamic backpropagation. While RTRL has the advantage of being an on-line learning algorithm, its computational complexity is an order of magnitude larger in the number of neurons than BPTT. On-line learning procedures are necessary in the case of adaptive control, but if we want to design a single controller once and for all, an off-line procedure will suffice. This is the case we consider here and we therefore limit our presentation to the BPTT algorithm. Sontag [131] and Żbikowski [151] noticed that the problem of learning a trajectory is a problem from optimal control. Indeed, we have a cost function that is an integral along

a trajectory $x(t)$ of the system (5.2) (this trajectory depending on the choice of the weights Λ and A):

$$\min_{\Lambda,A} \frac{1}{2} \int_{t_0}^{t_1} \|x(t) - z(t)\|^2 \, dt = \min_{\Lambda,A} E(\Lambda, A). \tag{5.3}$$

We want to minimize this cost by an appropriate choice of the "command" Λ and A. We can derive the BPTT algorithm as the optimal control solution to this problem by making use of the Pontryagin minimization principle, as presented by Draye in [32]. We write our cost E as

$$E = \int_{t_0}^{t_1} q(x(t), t) dt,$$

where $q(x(t), t) = (1/2)(x(t) - z(t))^T (x(t) - z(t))$ is our instantaneous cost (it depends both on $x(t)$ and t; the dependence on t comes through the target $z(t)$). Our objective is now to estimate the gradient of the error with respect to the parameters of the neural network.

Consider the following initial-value problem for a time-varying system with an unconstrained command v:

$$\begin{aligned} \frac{dx(t)}{dt} &= f(x(t), t; v(t)), \\ x(t_0) &= x_0. \end{aligned} \tag{5.4}$$

We have to be careful to understand what this initial-value problem means in the particular case of the neural network $\dot{x}(t) = -\Lambda x(t) + \sigma(Ax(t)) + u(t)$. The command $v(t)$ is our free parameter. When we train a neural network, the weights Λ and A are the free parameters, which we choose such that the network approximates the given input–output pair. Since the weights are fixed during the evolution of the neural network, $v(t) = v(0)$ for all times. The input–output pair $(u(t), x(t))$ is fixed a priori. This fact means that although the input u varies over time, it is not a free parameter anymore. The temporal evolution of f must account for the input and we stress, again, that the input $u(t)$ does not bear any relation with the command $v(t)$. The methods of optimal control, which we present now, allow a time-varying command. This fact explains why we introduce the command $v(t)$ while we know it to be equal to the pair (Λ, A). For the boundary value problem (5.4), we want to find the control v that minimizes the cost

$$Q(v(t), t_f) = \int_{t_0}^{t_f} q(x(t), v(t), t) dt \,,$$

where the initial time t_0 and final time t_f are fixed, and the final state $x(t_f)$ is free. The evolution of the state x depends on the command v; the instantaneous cost q depends on the command (for example, if v is a force applied to

modify the state, the cost could be the power needed to apply this force); the instantaneous cost is also time dependent (in the case of learning a trajectory, the cost depends on the difference between the actual output and the target that varies over time). We call the minimal cost $Q^*(t) = \min_v Q(v(t), t)$.

We define a new n-dimensional variable p^* called the adjoint variable as the gradient of the minimal cost with respect to the output trajectory:

$$p^*(t) = \nabla_x Q^* = \left(\frac{dQ^*(t)}{dx_1(t)}, \ldots, \frac{dQ^*(t)}{dx_n(t)} \right)^T;$$

which implies $\frac{dp^*(t)}{dt} = \frac{d}{dt}(\nabla_x Q^*) = \frac{\partial q(x(t), v(t), t)}{\partial x} + p^{*T}(t) \frac{\partial f(x(t), t; v(t))}{\partial x}$.

The Pontryagin minimum principle relies on setting up the following Hamiltonian H:

$$H(v(t), t) = q(x(t), t) + p^T(t) \frac{dx(t)}{dt}. \tag{5.5}$$

The right-hand side is indeed a Hamiltonian since

$$\begin{aligned} \frac{dx(t)}{dt} &= \nabla_p H, \\ \frac{dp(t)}{dt} &= -\nabla_x H. \end{aligned}$$

If we consider the minimum of the Hamiltonian $H^*(t) = \min_v H(v(t), t)$ with respect to the command v, the minimizer $v^*(t)$ is unique by Bellman's optimality principle [15]. Pontryagin showed that this minimum has the property that

$$\begin{aligned} \frac{dx^*(t)}{dt} &= \nabla_p H^* = \left(\frac{\partial H^*}{\partial p_1}, \ldots, \frac{\partial H^*}{\partial p_n}\right)^T, \\ \frac{dp^*(t)}{dt} &= -\nabla_x H^* = \left(-\frac{\partial H^*}{\partial x_1}, \ldots, -\frac{\partial H^*}{\partial x_n}\right)^T; \\ x^*(t_0) &= x_{0,} \\ p^*(t_1) &= 0, \end{aligned}$$

where $x^*(t)$ is the trajectory of the system for the optimal command v^* and $p^*(t)$ is the trajectory of the adjoint system for this optimal command. Since H^* determines $p^*(t)$ and $x^*(t)$ and that we can integrate $p^*(t)$ and $x^*(t)$ to get the minimal cost Q^*, the command that minimizes the Hamiltonian H is also the command that minimizes the cost. We can add that a command giving a local minimum of H will also give a local minimum of the cost Q. Many applications of optimal control rely on computing the first and second-order partial derivatives of the Hamiltonian with respect to the command to determine algebraic conditions for an extremum of the Hamiltonian. The resulting

system of equations is then solved explicitly. For our application, we need an iterative approach because we cannot solve the algebraic conditions explicitly.

Returning to our neural network $\dot{x}(t) = -\Lambda x(t) + \sigma(Ax(t)) + u(t)$ and our tracking error $E(\Lambda, A) = \frac{1}{2}\int_{t_0}^{t_1} \|x(t) - y(t)\|^2 \, dt$, the Hamiltonian (5.5) becomes

$$H(\Lambda, A, t) = q(x(t), t) + p^T(t)(-\Lambda x(t) + \sigma(Ax(t)) + u(t)).$$

Let us suppose we knew the optimal weights Λ^* and A^* for the neural network. We take into account that, in our case, $q(x(t), t) = (1/2)(x(t) - z(t))^T(x(t) - z(t))$, so we have $\partial q(x(t), t)/\partial x = x(t) - z(t)$. The equations for the neural network and the adjoint variables then become

$$\begin{aligned}
\frac{dx^*(t)}{dt} &= -\Lambda^* x^*(t) + \sigma(A^* x^*(t)) + u(t) && (5.6)\\
&= \nabla_p H^*, \\
\frac{dp^*(t)}{dt} &= (x^*(t) - y(t)) + p^{*T}(-\Lambda^* + \mathrm{diag}(\sigma'(A^* x^*(t)))A^*) && (5.7)\\
&= -\nabla_x H^*; \\
x^*(t_0) &= x_{0,} \\
p^*(t_1) &= 0.
\end{aligned}$$

Of course, these equations are not directly available since we do not know the optimal weights. We therefore proceed iteratively. If we give ourselves an initial guess $(\Lambda^{(0)}, A^{(0)})$ and act as if it was the optimal solution, we can integrate the equation of the network forward in time

$$\begin{aligned}
\frac{dx(t)}{dt} &= \nabla_p H(\Lambda, A) = -\Lambda x(t) + \sigma(Ax(t)) + u(t), && (5.8)\\
x(t_0) &= x_{0,}
\end{aligned}$$

with $\Lambda = \Lambda^{(0)}$ and $A = A^{(0)}$. After this, we integrate the equation of the adjoint variables *backward* in time

$$\begin{aligned}
\frac{dp(t)}{dt} &= (x(t) - z(t)) + p^T(-\Lambda + \mathrm{diag}(\sigma'(Ax(t)))A) && (5.9)\\
&= -\nabla_x H(\Lambda, A), \\
p(t_1) &= 0
\end{aligned}$$

to satisfy the boundary condition $p(t_1) = 0$. Remark that we can integrate the equations sequentially because the equation (5.6) for the neural network is independent from the equation (5.7) for the adjoint variables (although this is not true the other way round). Once we have integrated these equations we can determine the error associated to our choice of weights $\Lambda^{(0)}, A^{(0)}$ because $E = \int_{t_0}^{t_1} H(t)dt$. If we can estimate the gradients of the error E with respect

to the weights Λ and A, we can update the weights by gradient descent. We compute the gradients as

$$\begin{aligned}\frac{\partial E}{\partial \lambda_i} &= \int_{t_0}^{t_1} \frac{\partial H}{\partial \lambda_i} dt = -\int_{t_0}^{t_1} p_i(t)\lambda_i x_i(t) dt, \\ \frac{\partial E}{\partial A_{ij}} &= \int_{t_0}^{t_1} \frac{\partial H}{\partial A_{ij}} dt = \int_{t_0}^{t_1} p_i(t)\sigma'\left(\sum_{k=1}^{n} A_{ik}x_k(t)\right) x_j(t) dt,\end{aligned}$$

with $(i = 1, \ldots, n)$. Or, in vector notation,

$$\nabla_\lambda E = -\int_{t_0}^{t_1} \mathrm{diag}(p(t))\Lambda x(t) dt, \tag{5.10}$$

$$\nabla_A E = \int_{t_0}^{t_1} \mathrm{diag}(\sigma'(Ax(t)))p(t)x^T(t) dt. \tag{5.11}$$

Thus, after the integration of the network (5.8) and the adjoint variables (5.9), we can determine the gradient of the error with respect to the weights by integrating (5.10) and (5.11). Once we have determined these gradients, we update the weights by gradient descent,

$$\Lambda^{(i+1)} = \Lambda^{(i)} + \eta\nabla_\lambda E, \tag{5.12}$$

$$A^{(i+1)} = A^{(i)} + \eta\nabla_A E, \tag{5.13}$$

or any other gradient-based optimization algorithm, such as conjugate gradient or Levenberg–Marquardt. The iterative computation of Equations 5.8 to 5.13 is therefore guaranteed to converge to a local minimum of the cost function, as usual for gradient algorithms. The method is obviously computationally intensive since we need to integrate four systems of nonlinear ODEs with a total of $n^2 + 3n$ differential equations, where n is the number of neurons.

5.2 Observability

We now consider properties of neural networks that are directly relevant to control theory. We investigate first conditions guaranteeing the observability of the following type of neural networks:

$$\begin{aligned}\dot{x} &= \sigma(Ax + Bu), \\ y &= Cx,\end{aligned} \tag{5.14}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}^m, y \in \mathbb{R}^p$. We have changed here the notation of the matrices of the neural network slightly to parallel the notation for linear systems:

$$\begin{aligned}\dot{x} &= Ax + Bu, \\ y &= Cx + Du,\end{aligned}$$

so as to make a comparison with these easier. Observability guarantees that, given two different initial conditions for the neural network (5.14), it is always possible to choose an appropriate input that distinguishes the initial conditions. This notion means that if we run two different copies of the same system but for different initial conditions, we can find an input signal for which there is a time where the output of the two copies of the system are different. Recall that, for linear systems, we check the observability of the system by computing the rank of its observability matrix. A linear system is observable if and only if

$$\operatorname{rank}\left(\left[C^T, A^T C^T, \ldots, \left(A^T\right)^{n-1} C^T\right]\right) = n.$$

The first condition that the neural network has to fulfill is that the activation function satisfies the independence property presented in Chapter 2, Definition 2.6. Recall that this property requires that, if we consider a set of sigmoidal functions $\sigma(a_i\xi + a_{0i})$ together with a constant function, the functions in this set will always be linearly independent as long as the parameters are all different (up to a change of sign):

$$b_0 + b_i \sum_{i=1}^{n} \sigma(a_i\xi + a_{0i}) = 0 \text{ if and only if } b_0 = b_1 = \cdots = b_n = 0.$$

The next concept that was then introduced was that of an irreducible network. Since the networks here do not have a hidden layer as such and since the input parameter is here u, we have to reformulate the definition of irreducibility.

Definition 5.1. An $n \times m$ matrix B is *irreducible* if it belongs to the following set of matrices $\mathcal{B}^{n\times m}$:

$$\mathcal{B}^{n\times m} = \left\{ \begin{array}{l} B \in \mathbb{R}^{n\times m} \text{ such that } (B_{i1}, \ldots, B_{in}) \neq 0 \\ \text{and } (B_{i1}, \ldots, B_{in}) \neq \pm(B_{j1}, \ldots, B_{jn}) \text{ for all } j \neq i \end{array} \right\}.$$

This set contains the matrices whose rows are all nonzero and for which no two rows are equal up to a change of sign. It plays an important role in the proof of this result and other results on the controllability of the neural network and on the uniqueness of the parameterization, which we will discuss in the next sections. This set is dense in the set of all matrices since the matrices that do not belong to $\mathcal{B}^{n\times m}$ are a solution to the linear system of equations given by turning the inequalities above into equality. It represents a nondegeneracy condition corresponding to the condition that the matrix B has maximal rank in the case of linear systems. Even though this class of matrices is generic, the condition of irreducibility is nontrivial and imposes restrictions on the applicability of the results presented hereafter. Indeed, this condition requires that, among other things, the input influences all the state variables directly (since $(B_{i1}, \ldots, B_{in}) \neq 0$ for all i). This requirement is stringent.

It is, for example, not satisfied by systems containing chains of integrators ($\dot{x}_1 = x_2, \dot{x}_2 = x_2 + u$) or by cascade systems [128] similar to those presented in Chapter 3. For such cases, it is necessary to reformulate the problem to take the particular structure of B into account; and then prove again the property under consideration. There is no systematic way to do this at this point, although this problem is currently given proper attention [132]. The coming results hold for networks (5.14) whose matrix B is in $\mathcal{B}_{n\times m}$ and whose activation function satisfies the independence property.

Considering the canonical basis $\{u_1, \dots, u_n\}$ of $\mathbb{R}^n$, we define some particular subspaces with respect to this basis:

Definition 5.2. A *canonical subspace* $\mathcal{V}$ is a subspace spanned by a subset of the canonical basis vectors:

$$\{v_1, \dots, v_m\} \subset \{u_1, \dots, u_n\} \text{ and } \mathcal{V} = \operatorname{span}\{v_1, \dots, v_m\}.$$

Now we consider the projections π_i $(i = 1, \dots, n)$ on the basis vectors:

$$\pi_i : \mathbb{R}^n \to \mathbb{R}^n \text{ is such that } \pi_i(x) = (x^T u_i)u_i.$$

The canonical subspaces are the only subspaces that are invariant under all these projections.

Definition 5.3. An *invariant subspace* $\mathcal{V} = \operatorname{span}\{v_1, \dots, v_m\}$ *of a pair of matrices* (A, C) with $A \in \mathbb{R}^{n\times n}, C \in \mathbb{R}^{p\times n}$ is a subspace that is invariant under A and is included in the kernel of C:

$$\begin{cases} Cv_i = 0, \\ Av_i \in \mathcal{V}, \end{cases} \quad \text{for all } i = 1, \dots, m.$$

In particular, we call *maximal invariant canonical subspace* $\mathcal{I}(A, C)$ the largest canonical subspace $\operatorname{span}\{v_1, \dots, v_m\}$, with $\{v_1, \dots, v_m\} \subset \{u_1, \dots, u_m\}$, that is invariant for the pair of matrices (A, C). This maximal invariant canonical subspace exists and is unique; this uniqueness is a consequence of the fact that the sum of two canonical subspaces is again a canonical subspace.

We finally formalize the notion of observability.

Definition 5.4. A control system (5.1) is *observable* if, for any pair of initial conditions $(x_0^{(1)}, x_0^{(2)})$ with $x_0^{(1)} \neq x_0^{(2)}$, there exists an input $u(t)$ such that the output of these systems are different:

$$\exists u(t) \text{ and } t > 0 \text{ such that } y(t; u(t), x_0^{(1)}) \neq y(t; u(t), x_0^{(2)}).$$

The main result that Sontag and Albertini gave is that our neural network (5.14) is observable if the kernels of A and C have an empty intersection and the maximal invariant canonical subspace is the empty set.

Theorem 5.1 (Albertini and Sontag). *Consider a neural network of the form (5.14) for which $B \in \mathcal{B}^{n\times m}$ and σ satisfies the independence property. The neural network is observable if and only if*

$$\ker A \cap \ker C = \mathcal{I}(A, C) = \{0\}.$$

This condition is less restrictive than the condition for linear system, which is that there is no subspace (not necessarily canonical) that is invariant for A and included in the kernel of C. Observability of the linear system $\dot{x} = Ax + Bu, y = Cx$ implies observability of the corresponding neural network if $B \in \mathcal{B}^{n\times m}$. However, the observability of the neural network depends on the matrix B belonging to $\mathcal{B}^{n\times m}$, while B does not play a role in the case of linear systems. The condition $\mathcal{I}(A, C) = \{0\}$ is not difficult to check because there is an iterative algorithm to compute it. We initiate our iterations with $\mathcal{I}^{(0)} = \ker C$ and iterate with

$$\mathcal{I}^{(i+1)} = \mathcal{I}^{(i)} \cap \left(A^{-1}(\mathcal{I}^{(i)})\right) \cap \left(\pi_1^{-1}(\mathcal{I}^{(i)})\right) \cap \ldots \cap \left(\pi_n^{-1}(\mathcal{I}^{(i)})\right),$$

for $i = 1, \ldots, n-1$; where $A^{-1}(\mathcal{I}^{(i)}) = \left\{x \in \mathbb{R}^n : Ax \in \mathcal{I}^{(i)}\right\}$ and where we denote the set of inverse images by $\pi_i^{-1}(\mathcal{I}^{(i)}) = \left\{x \in \mathbb{R}^n : \pi_i(x) \in \mathcal{I}^{(i)}\right\}$. We have that $\mathcal{I}(A, C) = \mathcal{I}^{(n)}$. This algorithm runs in polynomial time. For the proof of the theorem, we refer to [7].

5.3 Controllability

Just as we have considered the problem of the observability of a neural network, it is possible to consider the question of its controllability. The property of controllability means that, given any initial state and any final state for the system, it is always possible to find a control that steers the system from the initial condition to the final condition in some finite time. Because we are only concerned with the internal state of the system and not with its output, we consider only the neural network

$$\dot{x} = \sigma(Ax + Bu), \tag{5.15}$$

with $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}$. The sigmoid is globally Lipschitz, so the solution of the differential equation is defined for all times on the whole of $\mathbb{R}^n$ as long as the control $u(t)$ is measurable and essentially bounded. Recall that, for a linear system $\dot{x} = Ax + Bu$, we can check controllability by computing the rank of the controllability matrix. A linear system is controllable if and only if

$$\operatorname{rank}\left(\left[B, AB \ldots, A^{n-1}B\right]\right) = n.$$

For the neural networks (5.15), Sontag and Sussman [133] give the following sufficient condition for controllability.

Theorem 5.2 (Sontag and Sussman). *If the matrix B is irreducible and the activation function σ is locally Lipschitz with the following properties:*

1. *It is an odd function.*
2. *It has a positive and finite limit at infinity:* $0 < \lim_{\xi\to\infty} \sigma(\xi) = L < \infty$.
3. *It is bounded above by its limit:* $\sigma(\xi) < L, \forall \xi \in \mathbb{R}$.
4. *The limit of the following ratio is zero:*

$$\text{For all } a > 0, a_0 \in \mathbb{R}, \text{ we have } \lim_{\xi\to\infty} \frac{L - \sigma(a\xi + a_0)}{L - \sigma(\xi)} = 0.$$

Then the neural network (5.15) is controllable.

The most commonly used activation functions: the hyperbolic tangent and the logistic sigmoid, satisfy the above conditions on the activation function. However, an activation like arctangent does not satisfy these conditions. We also see the class $\mathcal{B}^{n\times m}$ of irreducible matrices appear again. The condition does not put any limitation on the choice of A as long as B is irreducible. This situation parallels the case of linear systems, where the only choice of B that makes controllability independent of A is if B has rank n. The proof of the theorem is based on the following result on convex hulls and on a local controllability lemma.

Lemma 5.3. *For any $a_0 \in \mathbb{R}^n$, denote by $\mathcal{S}(B, a_0)$ the following set of n-dimensional vectors:*

$$\mathcal{S}(B, a_0) = \{\sigma(Bu + a_0) : u \in \mathbb{R}^m\}.$$

Suppose that σ satisfies the conditions of Theorem 5.2 and that B is irreducible ($B \in \mathcal{B}^{n\times m}$). Then, the origin belongs to the interior of the convex hull of $\mathcal{S}(B, a_0)$.

We refer to [133] for the proof.

Lemma 5.4. *Consider the control system $\dot{x} = f(x, u)$ defined on an open subset $\mathcal{D}$ of $\mathbb{R}^n$, where u takes values in a finite set of controls $\mathcal{U}$, and $f(x, u)$ is continuous with respect to x for any choice of u. Consider a point x_0 of $\mathcal{D}$ for which the set $\{f(x_0, u) : u \in \mathcal{U}\}$ of control vectors at x_0 is such that its convex hull contains the zero vector. Then there exists a neighborhood $\mathcal{W} \subset \mathcal{D}$ of x_0 where the system is controllable (we can steer any initial state $x^{(1)} \in \mathcal{W}$ into any final state $x^{(2)} \in \mathcal{W}$ in a finite time).*

This lemma does not make any reference to the particular case of a neural network. Its proof [133] is somewhat cumbersome; we therefore do not detail it further here. With these two lemmas, we are now in a position to prove the theorem.

Proof of Theorem 5.2. We want to show that the set of points that we can reach from any point of $\mathbb{R}^n$ is the whole of $\mathbb{R}^n$. We achieve this result by taking an arbitrary point $x_0 \in \mathbb{R}^n$ and showing that the set of points $\mathcal{R}(x_0)$ that we can reach from x_0 is both open and closed. Since $\mathcal{R}(x_0)$ is nonempty (it must at least contain x_0) and that the only nonempty subset of $\mathbb{R}^n$ that is both open and closed is $\mathbb{R}^n$ itself, $\mathcal{R}(x_0)$ must be equal to $\mathbb{R}^n$.

We show first that $\mathcal{R}(x_0)$ is open. Take $x^{(1)} \in \mathcal{R}(x_0)$ and set $a_0 = Ax^{(1)}$, then by Lemma 5.3 there is a finite set of controls $\mathcal{U} = \{u_1 \in \mathbb{R}^n, \dots, u_k \in \mathbb{R}^n\}$ belonging to $\mathbb{R}^n$ such that the origin is in the interior of the convex hull of the set of vectors $\mathcal{S}(B, a_0) = \{\sigma(Bu + a_0) : u \in \mathbb{R}^m\}$. Defining $\dot{x} = f(x, u)$ with $u \in \mathcal{U}$, we have by Lemma 5.4 that there is a neighborhood $\mathcal{W}$ of $x^{(1)}$ such that all its points are reachable from $x^{(1)}$. This reachability implies that this same control guarantees that there is an open neighborhood of $x^{(1)}$ where the neural network (5.15) is controllable. Since, at each point of $\mathcal{R}(x_0)$, we can find an open neighborhood that is completely included in $\mathcal{R}(x_0)$, the reachable set is open. We show now that $\mathcal{R}(x_0)$ is closed. Take $x^{(2)}$ in the closure of $\mathcal{R}(x_0)$, again by using Lemmas 5.3 and 5.4, there is a neighborhood $\mathcal{W}$ of $x^{(2)}$ for which all points can be steered into $x^{(2)}$. Since $x^{(2)}$ is in the closure of $\mathcal{R}(x_0)$, the intersection of $\mathcal{W}$ and $\mathcal{R}(x_0)$ cannot be empty, so there must exist a point of $\mathcal{W}$ that we can steer into $x^{(2)}$. Thus, $x^{(2)} \in \mathcal{R}(x_0)$—every point of the closure belongs to $\mathcal{R}(x_0)$—and $\mathcal{R}(x_0)$ is closed. As it is both open and closed, and it is nonempty, $\mathcal{R}(x_0)$ must be equal to $\mathbb{R}^n$, for any x_0. The neural network (5.15) is controllable. □

Theorem 5.2 led Sontag and Sussman to the surprising observation that it is possible to find a system with the arctangent activation function that is not controllable while the same system with the hyperbolic tangent activation is controllable—even though the activation functions qualitatively look the same. They give the following example.

Proposition 5.5. *The following system is not controllable if* $\sigma(\xi) = \arctan(\xi)$*:*

$$\begin{cases} \dot{x}_1 &= \sigma(x_1 + x_2 + x_3 + x_4 + 2u), \\ \dot{x}_2 &= \sigma(x_1 + x_2 + x_3 + x_4 + 12u), \\ \dot{x}_3 &= \sigma(-3u), \\ \dot{x}_4 &= \sigma(-4u). \end{cases} \tag{5.16}$$

We can write the system as $\dot{x} = \sigma(Ax + Bu)$ with

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } B = \begin{pmatrix} 2 \\ 12 \\ -3 \\ -4 \end{pmatrix},$$

so $B \in \mathcal{B}^{n\times 1}$ (since no element of B is zero and no two of its elements have identical absolute value). By Theorem 5.2, the above system (5.16) with $\sigma(\xi) = \tanh(\xi)$ is controllable.

The proof of this counter-example relies on showing that there is no control that can steer states in the half-hyperplane $x_1 + x_2 + x_3 + x_4 > 1$ outside of this half-plane. To prove this fact, it is enough to show that any control on the hyperplane $x_1+x_2+x_3+x_4 = 1$ will strictly force the state into the hyperplane $x_1+x_2+x_3+x_4 > 1$. Otherwise said, the scalar product of the vector field (for any choice of the control u) with the normal $(1,1,1,1)$ to the hyperplane must be positive. Thus, if $x_1+x_2+x_3+x_4 = 1$, then $(1,1,1,1)(\dot{x}_1,\dot{x}_2,\dot{x}_3,\dot{x}_4)^T > 0$, or $\dot{x}_1 + \dot{x}_2 + \dot{x}_3 + \dot{x}_4 > 0$ for any u. We thus need to show that the function $f(u) = \sigma(4+2u) + \sigma(4+12u) + \sigma(-3u) + \sigma(-4u)$ is strictly positive for any u. We can show this by analyzing the graph of $f(u)$ (Fig. 5.1), which is a little tedious but elementary.

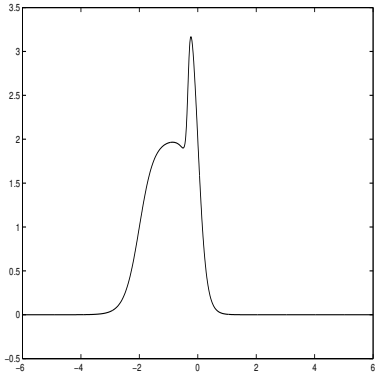


Figure 5.1: Graph of $f(u)$. The function is always positive, which guarantees that the control vectors of the system (5.16) force the state to stay within the half-space $x_1 + x_2 + x_3 + x_4 > 1$. Hence the system (5.16) is uncontrollable.

Note further that, if the matrix B is not in $\mathcal{B}^{n\times m}$, it is still possible that the neural network be controllable (5.15), but for an appropriate choice of A—for example,

$$\begin{aligned} \dot{x}_1 &= \sigma(x_2), \\ \dot{x}_2 &= \sigma(u), \end{aligned} \tag{5.17}$$

with $\sigma(\xi) = \tanh(\xi)$ is controllable (although $B = (0,1)^T \notin \mathcal{B}^{2\times 1}$). In this example $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$, but if $A = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ the system becomes uncontrollable (since $\dot{x} \equiv 0$).

Albertini and Dai Pra [5] gave a weaker result than the sufficient conditions for controllability Theorem 5.2. Their result guarantees forward accessibility—that is, that from every point we can reach some open neighborhood of this point. This property is weaker than controllability (with controllability we can reach the whole of the state-space), but the result only requires that σ satisfies the independence property and that B be irreducible.

5.4 Uniqueness of the weights of dynamical neural networks

For static neural networks, we saw that, as long as σ satisfied the independence property and that the network was irreducible, the map realized by the neural network was unique up to the obvious group of symmetries (permutation of two nodes and change of sign of all incoming and outgoing weights to a neuron for odd activation functions). We can ask the same question for recurrent neural networks. The answer to this question is analogous to the one for static networks. The input–output behavior of continuous-time networks is unique up to the trivial group of symmetries—although the conditions under which this result holds are somewhat different. Albertini and Sontag [6] considered a fairly large class $N(m, n, p; A, B, C, D, E)$ of neural networks with m inputs, n neurons, and p outputs:

$$\begin{aligned} \dot{x} &= Dx + \sigma(Ax + Bu) + Eu, \\ y &= Cx, \end{aligned} \tag{5.18}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}^m, y \in \mathbb{R}^p$, and the matrices A, B, C, D, E are of corresponding dimensions. The vector field of the network combines a sigmoidal part and a linear part. If $A = B = 0$, we recover a linear system. If $D = E = 0$, we recover the neural networks (5.14) presented in Section 5.2. We denote a particular solution $y(t)$ of the following initial value problem with input $u(t)$ by $y(t) = \Phi(x_0, u(.), t; A, B, C, D, E)$:

$$\begin{aligned} \dot{x} &= Dx + \sigma(Ax + Bu) + Eu, \\ y &= Cx; \\ x(0) &= x_0. \end{aligned}$$

Consider two systems with the same number of inputs and outputs, but not necessarily the same state dimension. Initialize both systems with $x(0) = 0$ (again, we might not have the same number of states) and apply the input $u(t)$. In the next two definitions, we consider the two neural networks:

$$N(m, n^{(1)}, p; A^{(1)}, B^{(1)}, C^{(1)}, D^{(1)}, E^{(1)}), \tag{5.19}$$

$$N(m, n^{(2)}, p; A^{(2)}, B^{(2)}, C^{(2)}, D^{(2)}, E^{(2)}). \tag{5.20}$$

Definition 5.5. Two continuous-time neural networks (5.19) and (5.20) are *input–output equivalent* if, choosing a zero initial condition for both systems and applying the same input at all times, the output of the two systems are equal for all times:

$$\Phi(0, u(.), t; A^{(1)}, B^{(1)}, C^{(1)}, D^{(1)}, E^{(1)}) = \Phi(0, u(.), t; A^{(2)}, B^{(2)}, C^{(2)}, D^{(2)}, E^{(2)}), \text{ for all } t \in \mathbb{R}. \quad (5.21)$$

In the particular case of linear systems;

$$\begin{cases} \dot{x}^{(1)} = A^{(1)}x^{(1)} + B^{(1)}u, \\ y^{(1)} = C^{(1)}x^{(1)}, \\ x^{(1)}(0) = 0 \end{cases} \text{ and } \begin{cases} \dot{x}^{(2)} = A^{(2)}x^{(2)} + B^{(2)}u, \\ y^{(2)} = C^{(2)}x^{(2)}, \\ x^{(2)}(0) = 0; \end{cases}$$

we have that the condition for input–output equivalence is that there exists an invertible change of coordinates between the two systems: There exists a $T \in \mathbb{R}^{n\times n}$ with $\det T \neq 0$ such that

$$T^{-1}A^{(1)}T = A^{(2)},\ T^{-1}B^{(1)} = B^{(2)}, \text{ and } C^{(1)}T = C^{(2)}.$$

For nonlinear systems, we will have a similar equivalence, but the class of transformations producing this equivalence will not be the class of invertible matrices, but a finite group. Again, we consider the set $\mathcal{P}$ of the matrices representing all permutations of n elements and the set $\mathcal{S}$ of the matrices with 1 or -1 on the diagonal and zero elsewhere.

Definition 5.6. Two continuous-time neural networks (5.19) and (5.20) are *sign–permutation equivalent* if $n^{(1)} = n^{(2)}$ and there exists $T = PS$ with $P \in \mathcal{P}$ and $S \in \mathcal{S}$ such that

$$\begin{aligned} T^{-1}A^{(1)}T &= A^{(2)}, \\ T^{-1}B^{(1)} &= B^{(2)}, \\ C^{(1)}T &= C^{(2)}, \\ T^{-1}D^{(1)}T &= D^{(2)}, \\ T^{-1}E^{(1)} &= E^{(2)}. \end{aligned} \quad (5.22)$$

With these definitions we can state the main result [6] (with $\sigma^{(q)}$ denoting the qth derivative of σ).

Theorem 5.6 (Albertini and Sontag). *Consider a neural network (5.18). Assume that the activation function σ satisfies the following conditions:*

1. *It is nonlinear and infinitely differentiable ($\sigma \in C^\infty$).*
2. *It is odd: $\sigma(-\xi) = -\sigma(\xi)$, implying for the derivatives that $\sigma^{(2q)}(0) = 0$ for all $q \in \mathbb{N}$.*

3. *It satisfies the following conditions on its derivatives at the origin:*

$$\sigma'(0) \neq 0, \quad \lim_{\xi \to \infty} \sigma'(\xi) = 0,$$
$$\text{there exists a } q > 1 \text{ such that } \sigma^{(2q+1)}(0) \neq 0.$$

Assume further that the matrices of the network satisfy the following conditions:

4. *The matrix B is irreducible:* $B \in \mathcal{B}^{n \times m}$.

5. *The matrix* $\begin{bmatrix} A, & B \end{bmatrix}$ *has maximal rank:* $\operatorname{rank}(\begin{bmatrix} A, & B \end{bmatrix}) = n$.

6. *The pair $(A + D, C)$ is observable:*

$$\operatorname{rank}\left(\left[C^T, (A+D)^T C^T, \ldots, \left((A+D)^T\right)^{n-1} C^T\right]\right) = n.$$

7. *The pair $(A + D, B + G)$ is controllable:*

$$\operatorname{rank}\left(\left[B+G, (A+D)(B+G), \ldots, (A+D)^{n-1}(B+G)\right]\right) = n.$$

Then two neural networks are input–output equivalent if and only if they are sign–permutation equivalent.

The proof of the necessary part of the theorem is obvious and does not use of any of the hypotheses. Using the change of variables $z = Tx$, the neural network (5.18) becomes

$$\begin{aligned} \dot{z} &= TDx + T\sigma(Ax + Bu) + TEu, \\ y &= Cx. \end{aligned}$$

Since T is invertible, $x = T^{-1}z$. Furthermore, the matrix T has the particular property that $T\sigma(v) = \sigma(Tv)$. This property results from the fact that the matrix $T = PS$ is the product of a permutation matrix P and a sign-change matrix S. Since σ operates coordinate by coordinate, we can do the permutation before the sigmoid as well as after: $P\sigma(v) = \sigma(Pv)$. Since σ is odd, the change of sign commutes with the sigmoid: $S\sigma(v) = \sigma(Sv)$. Thus, we have that

$$\begin{aligned} \dot{z} &= TDT^{-1}z + \sigma(TAT^{-1}z + TBu) + TEu, \\ y &= CT^{-1}z. \end{aligned} \tag{5.23}$$

Identifying the matrices of the system (5.23) to these of (5.18), we obtain the desired relations (5.22). The sufficient part of the theorem is significantly more difficult. It uses techniques using Lie derivatives, a concept that we will only introduce in Chapter 7. We refer to [6] for the proof. A difficulty with the previous result is the constraint that B be irreducible. Indeed, if for example

$B = 0$ in Equation (5.18), the previous result does not apply. Albertini and Sontag, proved their result again in this particular case and in some others.

We can, however, illustrate the idea of the proof by considering the trivial case of a single-input–single-output network with one neuron. Suppose we have two networks of the form (remember that we consider a zero initial condition)

$$\begin{array}{rcl} \dot{x}^{(1)} &=& -x^{(1)} + \sigma(b^{(1)}u), \\ y^{(1)} &=& c^{(1)}x^{(1)}, \\ x^{(1)}(0) &=& 0; \end{array} \quad \text{and} \quad \begin{array}{rcl} \dot{x}^{(2)} &=& -x^{(2)} + \sigma(b^{(2)}u), \\ y^{(2)} &=& c^{(2)}x^{(2)}, \\ x^{(2)}(0) &=& 0. \end{array}$$

The output of the first network, as a function of the input, is

$$y^{(1)}(t) = \int_0^t e^{t-s}c^{(1)}\sigma(b^{(1)}u(\tau))d\tau = \Phi^{(1)}(0, u(.), t)$$

as given by convolution of the impulse response of the system with the input. Since σ is odd, if the weights of the second network are opposite to those of the first ($b^{(2)} = -b^{(1)}, c^{(2)} = -c^{(1)}$), the two networks have the same behavior:

$$y^{(2)}(t) = \int_0^t e^{t-s}c^{(2)}\sigma(b^{(2)}u(\tau))d\tau = \int_0^t e^{t-s}c^{(1)}\sigma(b^{(1)}u(\tau))d\tau = y^{(1)}(t).$$

This relation corresponds to the sufficient part of the result. We want to show now that, if the two networks have the same behavior, the weights must be equal up to a change of sign ($(b^{(2)}, c^{(2)}) = \pm(b^{(1)}, c^{(1)})$). To avoid degenerate cases, assume $c^{(1)} \neq 0, c^{(2)} \neq 0, b^{(1)} \neq 0, b^{(2)} \neq 0$. Since, $y^{(2)}(t) = y^{(1)}(t)$ for all t, we need to have that the integrands are always equal and thus

$$c^{(2)}\sigma(b^{(2)}\nu) = c^{(1)}\sigma(b^{(1)}\nu) \text{ for all } \nu \in \mathbb{R}. \tag{5.24}$$

Condition 3 of Theorem 5.6 requires that $\sigma'(0) \neq 0, \sigma^{(2q)}(0) = 0$, and there is some $q > 0$ such that the $(2q+1)$th derivative of σ at zero is nonzero ($\sigma^{(2q+1)}(0) \neq 0$). By taking the first derivative of the equality (5.24) and evaluating it at zero, we have $c^{(2)}b^{(2)} = c^{(1)}b^{(1)}$. By taking the qth derivative of the equality and evaluating it at zero, we have $c^{(2)}\left(b^{(2)}\right)^{2q+1} = c^{(1)}\left(b^{(1)}\right)^{2q+1}$. Thus, $\left(b^{(2)}\right)^{2q} = \left(b^{(1)}\right)^{2q}$, which implies $b^{(2)} = \pm b^{(1)}$; which implies in turn $(b^{(2)}, c^{(2)}) = \pm(b^{(1)}, c^{(1)})$.

Now, if our network is of the form

$$\begin{array}{rcl} \dot{x} &=& -x + \sigma(ax) + bu, \\ y &=& cx, \\ x(0) &=& 0, \end{array}$$

we do not write an expression for $y(t)$, but study its derivatives around the origin; we have—assuming a constant control $u(t) = 1$ for all t and $\sigma^3(0) \neq 0$ (recall that $\sigma(0) = \sigma''(0) = 0$ and $\sigma'(0) \neq 0$)—

$$\begin{aligned} x(0) &= 0, \\ y &= cx; \end{aligned}$$

$$\begin{cases} \dot{x} &= -x + \sigma(ax) + bu, \\ \dot{y}(0) &= cb; \end{cases}$$

$$\begin{cases} \ddot{x} &= -\dot{x} + \sigma'(ax)a\dot{x}, \\ \ddot{y}(0) &= cb(1 - a\sigma'(0)); \end{cases}$$

$$\begin{cases} \dddot{x} &= \ddot{x} + (\sigma''(ax)(a\dot{x})^2 + \sigma'(ax)a\ddot{x}), \\ \dddot{y}(0) &= -cb(1 - a\sigma'(0))^2; \end{cases}$$

$$\begin{cases} \ddddot{x} &= -\dddot{x} + (\sigma'''(ax)(a\dot{x})^3 + 3\sigma''(ax)a^2\dot{x}\ddot{x} + \sigma'(ax)a\dddot{x}), \\ \ddddot{y}(0) &= cb(1 - a\sigma'(0))^3 + cb^3\sigma'''(0)a^3. \end{cases}$$

If we have two networks with weights $(a^{(1)}, b^{(1)}, c^{(1)})$ and $(a^{(2)}, b^{(2)}, c^{(2)})$, we deduce from the preceding relations:

$$c^{(2)}b^{(2)} = c^{(1)}b^{(1)} \text{ and } c^{(2)}b^{(2)}(1 - a^{(2)}\sigma'(0)) = c^{(1)}b^{(1)}(1 - a^{(1)}\sigma'(0)),$$

which, together with the previous relations, implies

$$a^{(2)} = a^{(1)} \text{ and } c^{(2)}\left(b^{(2)}\right)^3\sigma'''(0)\left(a^{(2)}\right)^3 = c^{(1)}\left(b^{(1)}\right)^3\sigma'''(0)\left(a^{(1)}\right)^3,$$

which implies again $c^{(2)}\left(b^{(2)}\right)^3 = c^{(1)}\left(b^{(1)}\right)^3$. Hence, $(b^{(2)}, c^{(2)}) = \pm(b^{(1)}, c^{(1)})$. The idea of studying the behavior around the initial condition is the central idea of the proof of Theorem 5.6.

5.5 Discrete-time systems

Although the focus of this work is on continuous-time systems, it is impossible to overlook discrete-time systems when considering neural networks from the point of view of control theory. Most work on neural networks in control that is directed towards applications uses discrete-time networks. These networks are therefore of importance from an engineering standpoint. The use of discrete-time models is the result of the pervasiveness of digital controllers and there is no sign that this trend will reverse in the near future. However, the study

in continuous time of recurrent neural networks makes more intuitive and elegant techniques available for their analysis. These results will often serve as an inspiration for discrete-time results. Among the results presented previously, almost all have discrete-time equivalents. Learning algorithms for discrete-time networks are well-known and completely similar to the ones presented in Section 5.1, such as RTRL [148], BPTT [122], or dynamic backpropagation [109] (which is a principled extension of BPTT to more complex network configurations). Conditions for uniqueness of the weights are available [131]. Albertini [7] solved the problem of observability—this question being actually easier to answer in discrete time than in continuous time—giving conditions identical to the continuous-time case. Characterizations of the controllability properties of discrete-time networks are, however, not yet available. Conditions for stability similar to these of Lur'e systems presented in Chapter 3 have been thoroughly investigated and generalized by Suykens [136], as well as the application of these stability criteria to the design of neurocontrollers based on techniques using linear matrix inequalities. We can say that discrete-time neural networks have found a broader application in control, which also means that a discussion of practical control schemes for continuous-time neural networks instead of discrete-time ones would be quite artificial. We therefore move to a short presentation of neural control schemes in discrete time.

The main landmark in the application of discrete-time neural networks to system identification and controller design is the work of Narendra and his students Parthasarathy [108, 109] and Levin [76, 77]. Many control strategies are possible. We can first of all consider different types of models for the plant or the controller we want to model. State-space models are one possibility:

$$\begin{aligned} x(k+1) &= f(x(k), u(k)), \\ y(k) &= h(x(k)). \end{aligned} \tag{5.25}$$

The main property of neural networks that we use in this context is that, whatever f and h, we can always approximate them using a neural networks, since we have shown that neural networks could approximate any map. We can similarly consider input–output models such as Nonlinear AutoRegressive models with eXternal inputs (NARX):

$$y(k+1) = f(y(k), \ldots, y(k-m), u(k), \ldots, u(k-p)). \tag{5.26}$$

We first review input–output models and come back to state-space models after that.

5.5.1 Input–output models

We assume here that the process has (at least approximately) a finite memory and that the knowledge of its past outputs over m time steps and past inputs

over p time steps is sufficient to determine its future output. We can furthermore incorporate noise by considering Nonlinear AutoRegressive Moving Average models with eXternal inputs (NARMAX):

$$y(k+1) = f(y(k), \ldots, y(k-m), u(k), \ldots, u(k-p), e(k), \ldots, e(k-q)),$$

where $e(k)$ is a discrete-time noise process. Of course, we can extend state-space models by incorporating noise into the model:

$$\begin{aligned} x(k+1) &= f(x(k), u(k), e_1(k)), \\ y(k) &= h(x(k), e_2(k)), \end{aligned}$$

where $e_1(k)$ and $e_2(k)$ are noise processes. The philosophy of Narendra's techniques is to approximate both the dynamics of the plant and of the controller using neural networks. This is an indirect control approach. It is also possible to build controllers directly without creating a model of the plant as done in direct control [123]. But direct control is usually less powerful than indirect control and we will not consider it here. In indirect control, we first build a model of the plant from input–output measurements; then, we develop our controller on the basis of this model; once we have developed the controller, we can use it to control the real plant.

The first step in indirect control is thus the identification of the plant. We consider first the case of input–output models. If we use, for example, a NARX model (5.26), we try to build an approximation for f. Given that we have all the input–output measurements $(y(1), \ldots, y(K), u(1), \ldots, u(K))$ available, we can approximate f by choosing a neural network architecture (which must be sufficiently rich) and find the parameters A that minimize the difference between the output predicted by the network and the observed output:

$$\begin{aligned} y(k+1) &= f(y(k), \ldots, y(k-m), u(k), \ldots, u(k-p)), \\ \hat{y}(k+1) &= N(y(k), \ldots, y(k-m), u(k), \ldots, u(k-p); A), \\ & \min_A \textstyle\sum_{k=\max(m,p)+1}^{K} \|y(k+1) - \hat{y}(k+1)\|^2 . \end{aligned} \tag{5.27}$$

We call this procedure the series–parallel mode. The solution to this problem is simple in the sense that it only requires the training of a feedforward network since when the input of the network is $(y(k), \ldots, y(k-m), u(k), \ldots, u(k-p))$, the target output will be $y(k+1)$ and all this data is available to us.

Another possibility is to approximate f by minimizing the difference between the output predicted by the network on the basis of its previous predictions and the observed output:

$$\begin{aligned} y(k+1) &= f(y(k), \ldots, y(k-m), u(k), \ldots, u(k-p)), \\ \hat{y}(k+1) &= N(\hat{y}(k), \ldots, \hat{y}(k-m), u(k), \ldots, u(k-p); A), \\ & \min_A \textstyle\sum_{i=\max(m,p)+1}^{K} \|y(k+1) - \hat{y}(k+1)\|^2 . \end{aligned} \tag{5.28}$$

Note the presence of $(\hat{y}(k), \ldots, \hat{y}(k-m))$ in the second equation of (5.28), instead of $(y(k), \ldots, y(k-m))$ in (5.28). We call this model the parallel mode. In this case, the identification of f amounts to the training of a recurrent network instead of a feedforward one. This task is much more difficult and we will tend to avoid it. However, if the dynamics of the system we want to identify is too complex, the series–parallel mode might not capture the dynamics of the plant well enough. Indeed, when developing our controller, we will use the neural network that we have identified from the measurements, but we will have to use this model in parallel mode. Indeed, when we develop our controller, we may want to make "virtual experiments" (i.e., simulations) for which no real data is available. If the controller developed on the series–parallel model fails to work on the plant, we may have no choice but to identify a parallel model for the plant [136].

Once we have identified a model of the plant, we can develop a controller using one of several strategies. The simplest one—supervised control—does not even require a model of the plant. Suppose we had extensive data available about the control actions that a human expert would take depending on the behavior of the plant. We might want to train a network to reproduce the same control actions and then use the network to replace the human operator, thereby cutting costs. Or we could have already a controller available but which might be too complex and too slow for a direct implementation (it could be for example based on a complex optimization procedure), we can then train a network to produce the same control actions and then replace the slow controller.

Another strategy that does not necessarily require a model of the plant is direct inverse control (Fig. 5.2). Here we train a neural network to reproduce the input of the plant when exposed to its output; this task is the converse of identification. When we want to produce a given output sequence, we use the neural network to produce the necessary input. However, the problem is often ill-posed because such an inverse is not guaranteed to exist, except locally, and it is in general not robust. Moreover, the neural network was trained as the right inverse of the plant, while it is used as its left inverse.

A strategy overcoming this problem is feedforward inverse control (Fig. 5.3); this approach is also the first to use a model of the plant. Here, we train a first neural network N_1 to identify the plant; then, we train a second neural network N_2 to produce the input that guarantees that the output of the model approaches the target output. Although more efficient than direct inverse control, feedforward inverse control remains an open loop strategy and suffers from a serious lack of robustness.

But we can extend this strategy of feedforward inverse control to use a feedback loop, thereby giving the strategy of internal model control (Fig. 5.4). There are two possibilities here. We can perform the training without taking the feedback loop into account and close the feedback loop only after training,

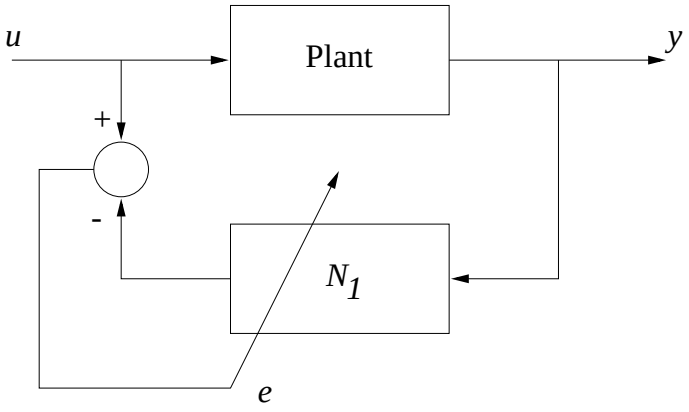


Figure 5.2: Direct inverse control. The arrow crossing the neural network N_1 indicates that we use the error e to adapt the weights of the network.

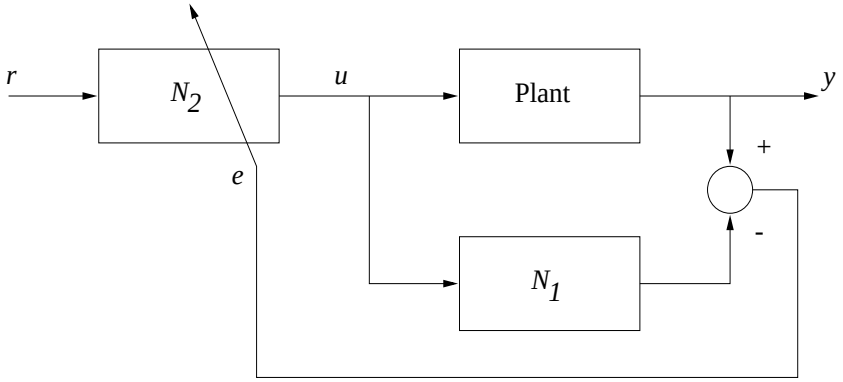


Figure 5.3: Feedforward inverse control. We train a neural network N_1 as a model of the plant and then a network N_2 as the right inverse of N_1.

this approach gives rise to a problem that we can solve by static backpropagation (we assume that we have already estimated N_1 during the identification step). Or we can perform the training including the feedback loop, which means that we then need dynamic backpropagation to train the network N_2.

Another strategy is model-reference adaptive control (Fig. 5.5). In this approach, we choose a reference system that models the desired behavior of the controlled plant (for example, a linear system with desired stability performances or a model trained after an expert as in supervised control). We then train a neural network so that, in combination with the plant, it produces

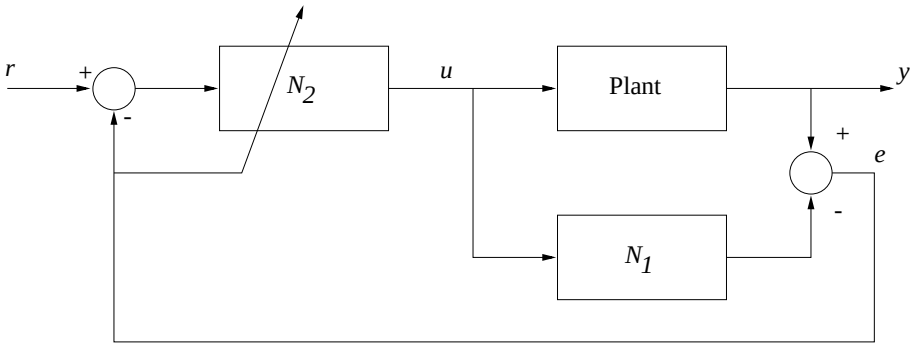


Figure 5.4: Internal model control. The neural networks are integrated into a feedback loop.

approximately the same behavior as the reference model. A corresponding direct control strategy exists but there exists no algorithm for training the neural network in this case.

A last strategy we mention is model predictive control. In this case we solve an optimization problem that gives us the input sequence necessary to produce a desired output sequence. At the next time step, we repeat the optimization procedure and update the input sequence. Of course, many variations exist on these strategies and their implementation may take some of the specific aspects of the plant into account.

5.5.2 State-space models

For state-space models, such as (5.25), the identification procedure is somewhat different. This difference comes from the fact that we now have to estimate both f and h in (5.25). We can estimate f and h separately if we can measure the state $x(k)$ of the system. But, in general, we have access only to $y(k)$, which contains only part of the information about $x(k)$ ($y(k)$ is often of a lower dimension than $x(k)$). But, if we model simultaneously f and h using two neural networks N_f and N_h, we can estimate the parameters of these two networks by fitting them with the available input–output measurements (this scheme is depicted on Figure 5.6). However, if we perform a nonlinear change of coordinates on the state of the system, we obtain another system with the same input–output behavior. The function f and h are, therefore, never uniquely determined by the input–output data and the solution to the optimization problem is not unique (although all solutions are equivalent). In many cases, we can a priori assume that h is a linear map as the output of a system is often a projection of its internal state. This assumption simplifies the

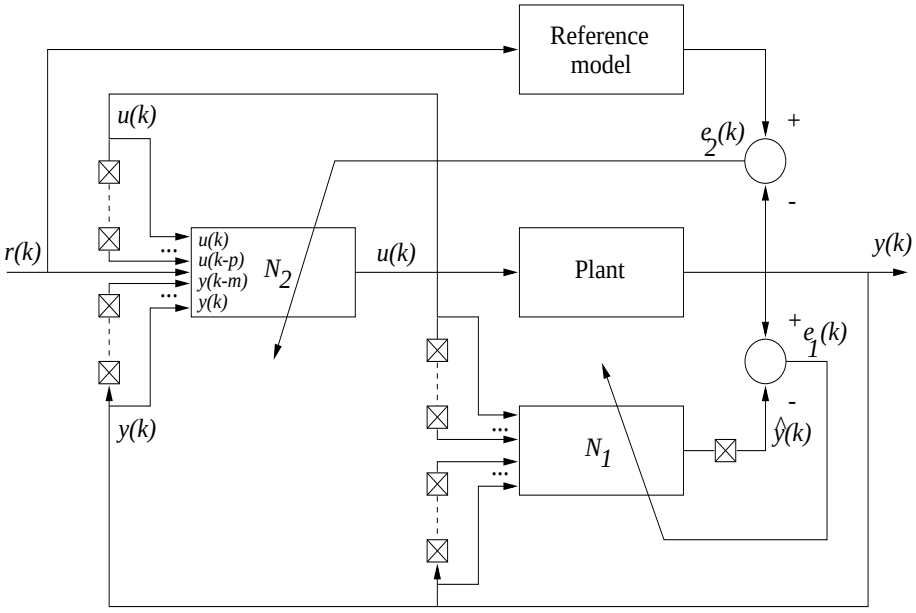


Figure 5.5: Model-reference adaptive control. The crossed squares indicate a delay of one unit of time (z^{-1}), which are used to produce the inputs $(y(k), \ldots, y(k-m))$ and $(u(k), \ldots, u(k-p))$ to the neural model N_1 of the plant and to the neural controller N_2.

estimation of the parameters, which is helpful since dynamic backpropagation is necessary for this identification procedure.

Control can then take several forms. We can try to perform a feedback linearization of the system—that is, we can try to find a feedback law that makes the resulting system linear (after which we can apply linear control procedures). This procedure does, however, not work in general and is not extremely robust.

Another control procedure is to build an observer and a controller for the identified system separately. Indeed, we do not have directly access to the state of the system but only to the output measurements $y(k)$. These measurements typically contain only partial information about the internal state $x(k)$. It is in general possible to reconstruct the internal state $x(k)$ from a sequence of inputs and outputs of length at most $2n+1$. More precisely, there exists an invertible map Ψ such that $x(k) = \Psi(y(k), \ldots, y(k-2n-1), u(k), \ldots, u(k-2n-1))$, by the Takens–Aeyels theorem [139, 4]. We want to approximate this map by a neural network N_Ψ. We then use simulations on our model of the plant to produce input sequences, output sequences, and state sequences (in the

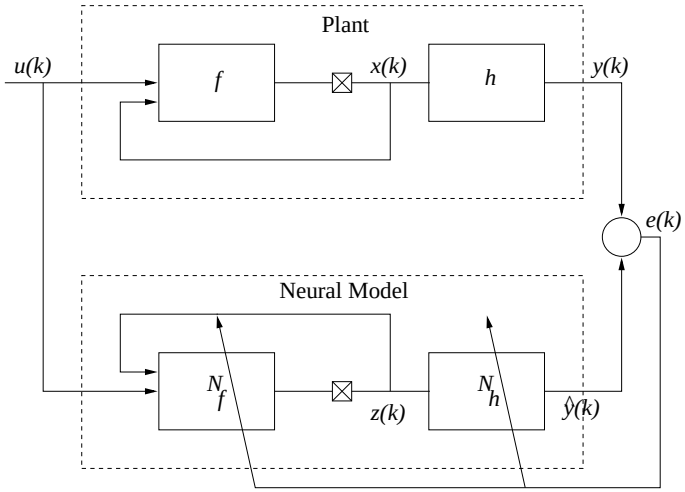


Figure 5.6: State-space model for identification. We identify the dynamics f and the output map h of the plant with the networks N_f and N_h. We adapt the weights simultaneously using dynamic backpropagation.

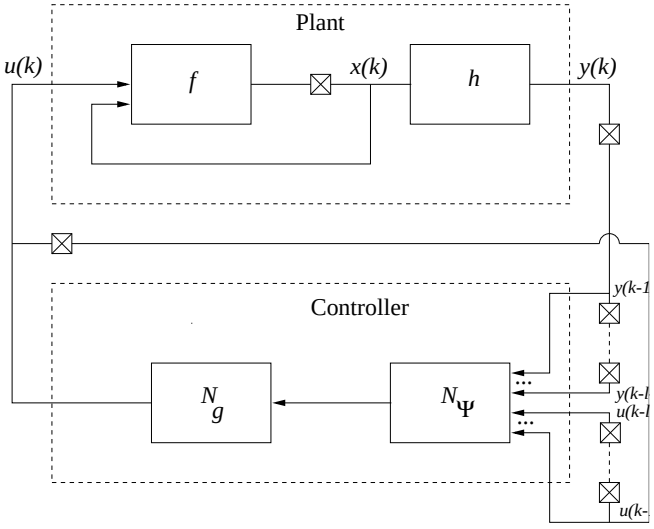


Figure 5.7: State-space control. We develop the observer and the controller separately, on the basis of the identified model. We then combine the observer and the controller to control the plant.

model of the plant, we have access to the state while on the real plant, we do not). Using this data we can estimate the observer Ψ independently from the controller. Using similar sequences, we can separately train a controller $u(k) = g(x(k))$ approximated by a neural network N_g. For this training, we use dynamic backpropagation to guarantee that the feedback loop created by N_f and N_g is stable. This procedure is presented in Figure 5.7. To run the controller on the real plant, we just combine the observer together with the controller. A last remark is that we can perform state-space control and input–output on-line as well as off-line. On-line training is in general less efficient than off-line training, but it is the only possibility in the case where the parameters of the plant change slowly over time.

5.6 Conclusion

After considering neural networks as dynamical systems in Chapters 3 and 4, we have studied dynamical neural networks with inputs in this chapter. One of the goals of control theory is precisely the study the behavior of systems with inputs and we have applied some of its concepts to neural networks.

We first considered the problem of training a neural network because the ability to match a system to observed data is essential for the data-driven approach typical of neural networks. We saw that we could formulate this estimation problem as an optimal control problem and, using Pontryagin's maximum principle, we derived the well-known algorithm of backpropagation through time.

After this, we reviewed the essential notions of observability and controllability. The work of Albertini and Sontag [6] provided conditions guaranteeing that a neural network is observable, while the work of Sontag and Sussman [133] provided conditions for controllability. Although we could not detail the proofs of these results, two central concepts emerged from their statements. These concept are the property of independence for an activation function and the irreducibility of the weight matrix of the dynamics of a network. These concepts correspond to those presented in the Chapter 2, Section 2.7, but applied to control systems. In particular, the concept of an irreducible matrix appears to play a role similar to that of a matrix having maximal rank in the case of linear systems. Furthermore, we saw that the uniqueness of the weights—but this time for dynamical neural networks—also depended on these key concepts.

Finally, we reviewed a number of control schemes for discrete-time neural networks. The reason why we focused on discrete-time system is that most practical applications of neural networks are for discrete-time systems since most controllers are digital. We saw that input–output schemes were available as well as state-space schemes. The input–output strategies were based on NARX or NARMAX modeling [108, 109] and ranged from direct inverse control

to model-reference adaptive control. The state-space models [76, 77] relied on plant identification and observer design before building a controller. In the light of this review of the observability and controllability properties of neural networks, and of the number of control-schemes they can implement, we can conclude that neural networks find naturally their place in the field of nonlinear control.

Chapter 6

Approximation capabilities of dynamical neural networks

Many neural network practitioners regard the dynamical neural networks without hidden layer presented in the previous chapters as of little practical interest. Their objection is that the absence of a hidden layer prevents the neural network that parameterizes the dynamics of the system from approximating vector fields universally. This limitation leads them to think that these networks can approximate only a limited class of dynamical systems. This belief is a misconception, as Sontag [131] sketched how dynamical single–hidden-layer perceptrons can universally approximate dynamical systems. We consider this result at the end of this chapter as well as a related result by Funahashi and Nakamura [42] that deals with Hopfield-type networks. Sontag's argument is existential and is only directed towards proving universal approximation of dynamical systems by dynamical perceptrons without hidden layer. The argument is that, for each single–hidden-layer dynamical perceptron, there is some dynamical perceptron without hidden layer such that the following holds. To each trajectory of the single–hidden-layer dynamical perceptron must correspond some trajectory of the dynamical perceptron (which has a higher dimensional state).

In contrast, the main emphasis of this chapter is to specify this transformation and generalize it. Our result is more precise than Sontag's and is applicable to a larger class of systems than only dynamical perceptrons with and without hidden layer. We describe in detail the transformation under which a dynamical neural network with a hidden-layer is input–output equivalent to some dynamical neural network without hidden layer that has as output some linear projection of its internal state (for some appropriate equivalent initial condition). The precise description of this transformation is necessary because it transforms the problem of studying a dynamical single–hidden-layer neural

network into the problem of studying some equivalent dynamical neural network without hidden layer. This possibility is important because the latter are more amenable to analytical approaches, as we have seen in the previous chapters (conditions for stability and convergence in Chapter 3; conditions observability and controllability in Chapter 5) and shall see again in the coming chapters (conditions for invertibility in Chapter 7).

The main key to this transformation is a generalization of Sontag's result. It is a theorem on transdimensional linear changes of coordinates in neural networks—that is, changes of coordinates between systems of different dimensions. This theorem gives precisely the parameters of the equivalent system and it is applicable to a wide range of neural-like systems. It complements the work by Albertini and Sontag [6] (see Chapter 5) on equivalences between dynamical neural networks without hidden layer (uniqueness of parameterization). It also complements the results on equivalence between neural networks and Lotka–Volterra systems we have discussed in Chapter 4. We use this theorem at the end of this chapter when we review Sontag's approximation result. We shall also demonstrate an elementary application of this transformation in Chapter 8 where we use it as a tool to help us develop new integration rules for dynamical neural networks [101].

6.1 Immersibility

In this chapter, we consider dynamical neural networks with external inputs. We study the behavior of these networks as multiple-input–multiple-output (MIMO) control systems, similar to the systems (5.1) we considered in Chapter 5. We define a MIMO system Γ as follows.

$$\Gamma : \begin{cases} \dot{x}(t) &= f(x(t), u(t)), \\ y(t) &= g(x(t), u(t)), \\ x(0) &= x_0, \end{cases}$$

where $x(t) \in \mathbb{R}^d, u(t) \in \mathbb{R}^m$, and $y(t) \in \mathbb{R}^n$. In our case, the dynamics f is parameterized by a feedforward neural network. We compare two classes of neural networks. First, feedforward neural networks without hidden layer; more precisely, transformations of the form $z = \Psi(A[x; u] + a_0)$, where Ψ is a continuous function. Second, feedforward neural networks with a hidden layer; that is transformations of the form $z = B\Psi(A[x; u] + a_0)$. The latter definition covers a wide range of neural networks called generalized regularization networks [47]: single–hidden-layer perceptrons, radial–basis-function networks, hinging hyperplanes, multiple adaptive regression splines, and so on. This definition is much broader than the single–hidden-layer perceptrons we have discussed so far, but the reader unfamiliar with these models may consider the following discussion from the restricted point of view of perceptron-type

networks without harm. We thus have an m-input–n-output system with a d-dimensional state. We are going to pay much attention to the evolution of the output of the system, which we write as

$$y(t) = \Gamma(x_0, u(.), t).$$

We assume the classical conditions on the vector field f and on the admissible controls $u(.)$ to guarantee the existence and uniqueness of solutions [130] for some time interval. Furthermore, the vector fields that we consider here always satisfy a global Lipschitz conditions; this guarantees that the solution exists for all times, for well-behaved controls.
We would like to describe the situation where two systems with the same number of inputs and outputs, but with possibly a different number of states, exhibit identical input–output behavior. Let us consider two systems Γ and Ξ:

$$\Gamma : \left\{ \begin{array}{rcl} \dot{x}(t) & = & f(x(t), u(t)), \\ y(t) & = & g(x(t), u(t)), \\ x(0) & = & x_0; \end{array} \right. \qquad \Xi : \left\{ \begin{array}{rcl} \dot{z}(t) & = & h(z(t), v(t)), \\ w(t) & = & k(z(t), u(t)), \\ z(0) & = & z_0, \end{array} \right.$$

where $x(t) \in \mathbb{R}^d, u(t) \in \mathbb{R}^m$, and $y(t) \in \mathbb{R}^n$, and where $z(t) \in \mathbb{R}^{d'}, v(t) \in \mathbb{R}^{m'}$, and $w(t) \in \mathbb{R}^{n'}$. We can write the outputs of the systems as

$$\begin{aligned} y(t) &= \Gamma(x_0, u(.), t), \\ w(t) &= \Xi(z_0, v(.), t). \end{aligned}$$

Under what conditions are these two outputs equal for all times? The main idea in our discussion is that a system with a larger number of states can reproduce the behavior of a system with smaller number of states if, to each initial condition of the smaller system, we can associate an initial condition of the larger system so that the input–output behavior of the systems are identical. This idea is similar to the one we studied in Chapter 4 but we restrict our discussion here to linear transformations. As a matter of fact, we considered these linear changes of coordinates before studying the relationship between neural networks and Lotka–Volterra systems. We express this notion of equivalence through the following definitions. We first consider some properties of maps and then some properties of control systems.

Definition 6.1. A map $\varphi : \mathbb{R}^d \to \mathbb{R}^{d'}$ is an *immersion* if $\text{rank}(\nabla\varphi(x)) = d$ for all $x \in \mathbb{R}^d$.

This definition implies that φ maps $\mathbb{R}^d$ onto a differentiable submanifold of $\mathbb{R}^{d'}$ because the tangent plane exists at all points (the gradient does not degenerate). We always need $d \leq d'$ to have an immersion.

Definition 6.2. A map $\varphi : \mathbb{R}^d \to \mathbb{R}^{d'}$ is a *univalent immersion* if it is an immersion and it is injective.

Recall that φ is injective if, for any $x_0, x_1 \in \mathbb{R}^d$, we have that $x_0 \neq x_1$ implies $\varphi(x_0) \neq \varphi(x_1)$. We refer to Isidori [64] for examples highlighting the two definitions. We now apply these definitions to control systems and show under which conditions different systems can have the same input–output behavior.

Definition 6.3. The control system Γ is *immersible* into Ξ if

1. The number of inputs of Γ and Ξ are equal and Ξ has a higher state dimension than Γ: $m = m', n = n'$, and $d \leq d'$.
2. There exists a univalent immersion $\varphi : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that

$$\forall x_0, \forall \text{ admissible } u(.) : \Gamma(x_0, u(.), t) = \Xi(\varphi(x_0), u(.), t), \forall t.$$

Furthermore, the injectivity of φ implies that we can unambiguously retrieve x_0 from the knowledge of $\varphi(x_0)$. If the systems have the same number of states, φ may be a bijection; in which case we say that the two systems are *conjugate*.

When a system is immersible into another or when two systems are conjugate, we speak loosely of input–output equivalence. For convenience, we will slightly abuse the notation and denote immersibility or conjugacy as follows:

$$\Gamma : \begin{cases} \dot{x} = f(x,u), \\ y = g(x,u), \\ x(0) = x_0 \end{cases} \quad \begin{array}{c} \text{is immersible into} \\ \text{(is conjugate to)} \end{array} \quad \Xi : \begin{cases} \dot{z} = h(z,u), \\ y = k(z,u) \\ z(0) = \varphi(x_0). \end{cases}$$

We always carefully write the control system together with its initial condition. We shall see that this care is necessary because we have to make sure that φ is injective. As an illustration of these concepts, consider the following figure (Fig. 6.1). Suppose we have the two-dimensional autonomous system on the left, which has a periodic orbit. We say that this two-dimensional system is immersible into the three-dimensional system on the right. Indeed, using an output transformation from $\mathbb{R}^3$ to $\mathbb{R}^2$ (in this case, a linear projection $y = Bz$), we can retrieve the dynamics of the original system. This example is only an illustration, and the definition of immersibility covers a broader concept. We consider here only linear changes of coordinates, but nonlinear changes of coordinates may prove useful also.

6.2 Changes of coordinates between systems of different dimensions

Consider the following general continuous-time architecture, as discussed in Section 6.1,

$$\Gamma_B : \begin{cases} \dot{x} &= B\Psi(x,u), \\ y &= x, \\ x(0) &= x_0, \end{cases}$$

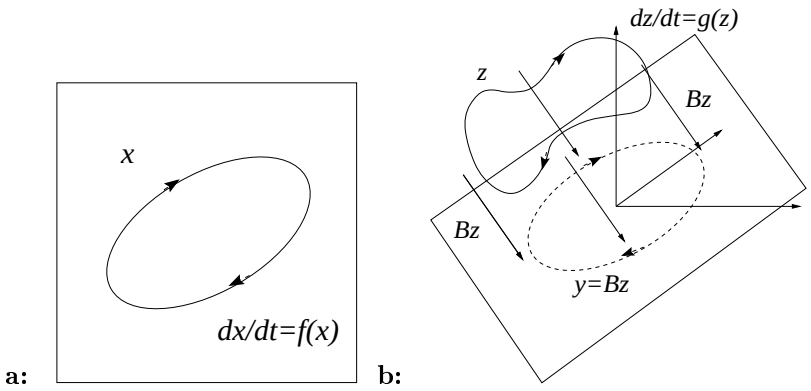


Figure 6.1: Illustration of the concept of immersibility. **a:** A two-dimensional system. This system is immersible into the three-dimensional system on the right. **b:** An equivalent three-dimensional system. We can recover the original system from the higher-dimensional system through a projection.

where $x \in \mathbb{R}^d$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^d$, B is a $d \times n$ matrix, Ψ is a C^1 function from $\mathbb{R}^d \times \mathbb{R}^m$ to $\mathbb{R}^n$ satisfying a global Lipschitz condition. In the next three subsections, we prove the following theorem about changes of coordinates between systems of different dimensions.

Theorem 6.1. *Γ_B is immersible into*

$$\Xi_B : \begin{cases} \dot{z} &= \Psi(Bz + (I - BB^\dagger)x_0, u), \\ y &= Bz + (I - BB^\dagger)x_0, \\ z(0) &= B^\dagger x_0, \end{cases}$$

where $z \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^d$, and $B^\dagger$ is the Moore–Penrose pseudoinverse of B.

Once we have proved this theorem, we shall apply it to neural networks, in particular to single–hidden-layer perceptrons. The proof of the theorem relies heavily on the use of linear algebra, in particular of the singular value decomposition (SVD). We first recall the concepts of linear changes of coordinates and of SVD.

6.2.1 Linear changes of coordinates

We show first that a linear change of coordinates $z = Qx$ with Q an invertible matrix transforms a system Γ into a conjugate system Γ':

$$\Gamma : \left\{ \begin{array}{rcl} \dot{x} & = & f(x,u), \\ y & = & g(x), \\ x(0) & = & x_0 \end{array} \right. \text{ is equal to } \left\{ \begin{array}{rcl} Q\dot{x} & = & Qf(Q^{-1}Qx,u), \\ y & = & g(Q^{-1}Qx), \\ Qx(0) & = & Qx_0, \end{array} \right.$$

$$\text{which is conjugate to } \Gamma' : \left\{ \begin{array}{rcl} \dot{z} & = & f(Q^{-1}z,u), \\ y & = & g(Q^{-1}z), \\ z(0) & = & Qx_0. \end{array} \right.$$

We can apply this elementary change of coordinates to a dynamical single–hidden-layer perceptron

$$\left\{ \begin{array}{rcl} \dot{x} & = & B\sigma(Ax + a_0), \\ y & = & x, \\ x(0) & = & x_0 \end{array} \right.$$

in the case where $d = n$ (number of states = number of neurons) and B is an invertible matrix. We can then use the change of coordinates $z = B^{-1}x$. We see that the single–hidden-layer network (6.2.1) is conjugate to

$$\left\{ \begin{array}{rcl} \dot{z} & = & \sigma(ABz + a_0), \\ y & = & Bz, \\ z(0) & = & B^{-1}x_0, \end{array} \right.$$

This conjugacy is the kind of result we aim for, but here it is limited to the case where the number of states equals the number of neurons and B is invertible. It also points at the general idea of the method, which is to try to find an invertible change of coordinates that cancels the effect of the matrix B that comes after the activation function. The problem is that, if B is not square or it is not of maximal rank, B^{-1} does not exist as such and the change of coordinates we just used does not make sense. We need to find an alternative to the use of the inverse of B. This necessity hints at the use of the pseudoinverse, and thus, of the SVD.

We now briefly recall the basics properties of the SVD. If B is a $d \times n$ matrix of rank r, the SVD decomposes the matrix as follows:

$$B = U\Sigma V^T,$$

where U is a $d \times d$ orthogonal matrix, Σ is $d \times n$ matrix with all elements equal to zero except for the first r elements on the diagonal, V is a $n \times n$ orthogonal matrix. The matrix Σ writes as

$$\Sigma = \begin{bmatrix} \mu_1 & 0 & \cdots & \cdots & \cdots & 0 & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & \cdots & 0 & \cdots & 0 \\ \vdots & 0 & \mu_r & & & \vdots & & \vdots \\ \vdots & \vdots & & 0 & & \vdots & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & \cdots & 0 \end{bmatrix}. \tag{6.1}$$

The matrices U and V are invertible changes of coordinates. Also, the orthogonality of U and V implies that $U^{-1} = U^T$ and $V^{-1} = V^T$.

6.2.2 Rank $B = d$

For the clarity of the presentation, we start by proving a simplified version of Theorem 6.1; we treat the case where rank $B = d$. After we have proved this result, we tackle the main theorem. In fact, Theorem 6.1—which is the case rank $B = r \leq d$—encompasses the first result as a corollary, but proving only the second case would obscure what the path of our reasoning is. In what follows, we denote by $O^{l \times m}$ an $l \times m$ zero matrix and by $I^{l \times m}$ an $l \times m$ matrix with ones on the diagonal and zeros elsewhere. We also denote by O a zero submatrix that appropriately fits into the main matrix. We thus now prove the following corollary of Theorem 6.1.

Corollary 6.2. *If rank $B = d$, Γ_B is immersible into*

$$\Xi_B : \begin{cases} \dot{z} &= \Psi(Bz, u), \\ y &= Bz, \\ z(0) &= B^\dagger x_0. \end{cases}$$

Proof. First, note that rank $B = d$ implies $d \leq n$. We start with

$$\Gamma_B : \begin{cases} \dot{x} &= B\Psi(x, u), \\ y &= x, \\ x(0) &= x_0, \end{cases}$$

where B is a $d \times n$ matrix. By the SVD of B, we know that there exist U, Σ, and V for which $B = U\Sigma V^T$. So, by substituting $U\Sigma V^T$ for B in Γ_B, we have trivially that

$$\Gamma_B : \begin{cases} \dot{x} &= U\Sigma V^T \Psi(x, u), \\ y &= x, \\ x(0) &= x_0. \end{cases}$$

Our goal is to eliminate the matrices at the left of Ψ. Since U is unitary and thus $U^{-1} = U^T$, we can already eliminate U by using the change of coordinates $v = U^T x$, which amounts to multiplying the first and the third equation by U^T:

$$\Gamma_B : \left\{ \begin{array}{rcl} U^T \dot{x} & = & U^T U \Sigma V^T \Psi(x, u), \\ y & = & x, \\ U^T x(0) & = & U^T x_0. \end{array} \right.$$

The change of coordinates $v = U^T x$, also gives us $x = Uv$ and $U^T \dot{x} = \dot{v}$; so Γ_B is conjugate to

$$\left\{ \begin{array}{rcl} \dot{v} & = & \Sigma V^T \Psi(Uv, u), \\ y & = & Uv, \\ v(0) & = & U^T x_0. \end{array} \right.$$

We want to proceed further by using a change of coordinates that cancels the effect of Σ. The problem is that Σ is not an invertible matrix. Since Σ is of rank d, we can write $\Sigma = \left[\begin{array}{cc} \Lambda & O^{d \times (n-d)} \end{array} \right]$ where Λ is a diagonal matrix: $\Lambda = \mathrm{diag}(\mu_1, \ldots, \mu_d)$. To circumvent the problem of Σ not being invertible, we try to rewrite our system not in terms of Σ but in terms of a diagonal square matrix $\bar{\Lambda} = \left[\begin{array}{cc} \Lambda & O \\ O & I^{(n-d)\times(n-d)} \end{array} \right]$ that is invertible. To achieve this modification, we need first to define a matrix $\bar{J}$ by $\bar{J} = \left[\begin{array}{cc} I^{d \times d} & O^{d \times (n-d)} \end{array} \right]$. Consider now the following differential equation:

$$\dot{w} = \bar{\Lambda} V^T \Psi(U \bar{J} w, u), \qquad w \in \mathbb{R}^n.$$

We want to show that this differential equation can replace the differential equation for v. Because w belongs to $\mathbb{R}^n$ while v belongs to $\mathbb{R}^d$, we identify only the first d components of w to v. We define for this purpose $w_d = (w_1, \ldots, w_d)^T$ and $w_{\bar{d}} = (w_{d+1}, \ldots, w_n)^T$. We see that, because $\bar{J} w = w_d$, the differential equation for w_d does not depend on $w_{\bar{d}}$. By restricting the differential equation to its first d components, we have that

$$\dot{w}_d = \Sigma V^T \Psi(U w_d, u), \qquad w_d \in \mathbb{R}^d.$$

Hence, the differential equation for w_d is identical to the differential equation for v. This identity implies that for an appropriate choice of the initial condition, the dynamics of w reproduces the dynamics of v through its first d components. By setting $w_d(0) = v(0)$ and $w_{\bar{d}}(0) = 0$, we have that

$$\left\{ \begin{array}{rcl} \dot{v} & = & \Sigma V^T \Psi(Uv, u), \\ y & = & Uv, \\ v(0) & = & U^T x_0 \end{array} \right.$$

is immersible into

$$\left\{ \begin{array}{rcl} \dot{w} & = & \bar{\Lambda} V^T \Psi(U \bar{J} w, u), \\ y & = & Uv = U \bar{J} w, \\ w(0) & = & \left[\begin{array}{c} v(0) \\ O^{(n-d)\times 1} \end{array} \right] = \bar{J}^T U^T x_0. \end{array} \right.$$

What we have achieved with this transformation is that $\bar{\Lambda}$ is an invertible matrix with $\bar{\Lambda}^{-1} = \begin{bmatrix} \Lambda^{-1} & O \\ O & I^{(n-d)\times(n-d)} \end{bmatrix}$. Therefore, $z = V\bar{\Lambda}^{-1}w$ is an invertible change of coordinates (as V is also invertible). Proceeding as before, we have that

$$\begin{cases} \dot{w} &= \bar{\Lambda}V^T\Psi(U\bar{J}w,u), \\ y &= U\bar{J}w, \\ w(0) &= \bar{J}^TU^Tx_0 \end{cases}$$

is conjugate to

$$\begin{cases} \dot{z} &= \Psi(U\bar{J}\bar{\Lambda}V^Tz,u), \\ y &= U\bar{J}\bar{\Lambda}V^Tz, \\ y(0) &= V\bar{\Lambda}^{-1}w(0) = V\bar{\Lambda}^{-1}\bar{J}^TU^Tx_0. \end{cases}$$

All we have to do now is simplify the above system. To achieve this simplification, we work out the following expression:

$$\begin{aligned} \bar{J}\bar{\Lambda} &= \begin{bmatrix} I^{d\times d} & O^{d\times(n-d)} \end{bmatrix} \begin{bmatrix} \Lambda & O \\ O & I^{(n-d)\times(n-d)} \end{bmatrix} \\ &= \begin{bmatrix} \Lambda & O^{d\times(n-d)} \end{bmatrix} \\ &= \Sigma. \end{aligned}$$

In the first two equations of the previous system, we see that $U\bar{J}\bar{\Lambda}V^T = U\Sigma V^T = B$. Similarly, we work out the expression

$$\bar{\Lambda}^{-1}\bar{J}^T = \begin{bmatrix} \Lambda^{-1} & O \\ O & I^{(n-d)\times(n-d)} \end{bmatrix} \begin{bmatrix} I^{d\times d} \\ O^{(n-d)\times d} \end{bmatrix} = \begin{bmatrix} \Lambda^{-1} \\ O^{(n-d)\times d} \end{bmatrix} = \Sigma^\dagger,$$

where $\Sigma^\dagger$ is the pseudoinverse of Σ. Thus, $V\bar{\Lambda}^{-1}\bar{J}^TU^T = V\Sigma^\dagger U^T = B^\dagger$, where $B^\dagger$ is the Moore–Penrose pseudoinverse of B [48].
Taking these simplifications together, we have

$$\Gamma_B : \begin{cases} \dot{x} = B\Psi(x,u), \\ y = x, \\ x(0) = x_0 \end{cases} \quad \text{is immersible into } \Xi_B : \begin{cases} \dot{z} = \Psi(Bz,u), \\ y = Bz, \\ z(0) = B^\dagger x_0, \end{cases}$$

which proves Corollary 6.2. □

6.2.3 Rank $B \leq d$

In the case where rank $B = r < d$, things get a little trickier. The problem is that the matrix $\bar{\Lambda} = \begin{bmatrix} \Lambda & O \\ O & I^{(n-d)\times(n-d)} \end{bmatrix}$ is not invertible anymore, since the $(r+1)^{th}$ to the d^{th} rows are filled with zeros (the $d-r$ last singular values $\mu_{r+1},\ldots,\mu_d$ are zero since the rank of B is r). We have to devise a way to remove these rows. We do this by working only with the first r rows of Σ.

Proof of Theorem 6.1. We start as before with the change of coordinates $v = U^T x$. We have that

$$\Gamma_B : \left\{ \begin{array}{rcl} \dot{x} & = & B\Psi(x,u), \\ y & = & x, \\ x(0) & = & x_0 \end{array} \right. \text{ is conjugate to } \left\{ \begin{array}{rcl} \dot{v} & = & \Sigma V^T \Psi(Uv,u), \\ y & = & Uv, \\ v(0) & = & U^T x_0. \end{array} \right.$$

But, Σ is of the form (6.1) and thus has its last $d-r$ rows equal to zero. We define the $r \times n$ matrix Σ_r that has the r nonzero singular values of B on its diagonal. This matrix is such that $\Sigma = \left[\begin{array}{c} \Sigma_r \\ O^{(d-r)\times n} \end{array} \right]$. Given that the last $d-r$ rows of Σ are zero, the dynamics of the last $d-r$ state variables is zero. We partition the state vector z to separate the variables with a nonzero dynamics from those that remain constant; we define $v_r = (v_1, \ldots, v_r)^T$, $v_{\bar{r}} = (v_{r+1}, \ldots, v_d)^T$. We rewrite the previous equations according to this partition.

$$\left\{ \begin{array}{rcl} \dot{v}_r & = & \Sigma_r V^T \Psi(U \left[\begin{array}{c} v_r \\ v_{\bar{r}} \end{array} \right], u), \\ \dot{v}_{\bar{r}} & = & 0, \\ y & = & U \left[\begin{array}{c} v_r \\ v_{\bar{r}} \end{array} \right], \\ v_r(0) & = & U_r^T x_0, \\ v_{\bar{r}}(0) & = & U_{\bar{r}}^T x_0. \end{array} \right.$$

In the above equations, we also have partitioned the matrix U as follows: $U = \left[\begin{array}{cc} U_r & U_{\bar{r}} \end{array} \right]$ where U_r is the matrix formed by the r first columns of U and $U_{\bar{r}}$ is the matrix formed by the remaining $d-r$ columns of U. The matrix $U_{\bar{r}}$ is a basis of the left nullspace of B. Using this partition of U further, we can throw away the variable $v_{\bar{r}}$ because we can replace it by its constant value as $v_{\bar{r}} = v_{\bar{r}}(0) = U_{\bar{r}}^T x_0$; we have

$$\left\{ \begin{array}{rcl} \dot{v}_r & = & \Sigma_r V^T \Psi(U_r v_r + U_{\bar{r}} U_{\bar{r}}^T x_0, u), \\ y & = & U_r v_r + U_{\bar{r}} U_{\bar{r}}^T x_0, \\ v_r(0) & = & U_r^T x_0, \end{array} \right.$$

Now, the matrix Σ_r has a rank equal to its number of rows. We can thus proceed as in the previous section by extending Σ_r to a diagonal matrix thereby obtaining an invertible change of coordinates. We need to define the following matrices. We form the $r \times r$ matrix Λ_r by taking the r first columns of Σ_r: $\Sigma_r = \left[\begin{array}{cc} \Lambda_r & O^{r\times(n-r)} \end{array} \right]$. Define $\bar{\Lambda} = \left[\begin{array}{cc} \Lambda_r & O \\ O & I^{(n-r)\times(n-r)} \end{array} \right]$, which is an invertible matrix. Define $w_r = (w_1, \ldots, w_r)^T$ and $w_{\bar{r}} = (w_{r+1}, \ldots, w_n)^T$. Define the matrix $\bar{J}_r = \left[\begin{array}{cc} I^{r\times r} & O \\ O & O^{(d-r)\times(n-r)} \end{array} \right]$. As we identify w_r with v_r, we have that

$U_r v_r = U\bar{J}_r w$. If we set $w_r(0) = v(0)$ and $w_{\bar{r}}(0) = 0$, we have that the previous system is immersible into

$$\left\{\begin{array}{rcl} \dot{w} & = & \bar{\Lambda}_r V^T \Psi(U\bar{J}_r w, u), \\ y & = & U\bar{J}_r w + U_{\bar{r}} U_{\bar{r}}^T x_0, \\ w(0) & = & \bar{J}_r^T U^T x_0. \end{array}\right.$$

But now, $\bar{\Lambda}_r$ is an invertible matrix with $\bar{\Lambda}_r^{-1} = \left[\begin{array}{cc} \Lambda_r^{-1} & O \\ O & I^{(n-r)\times(n-r)} \end{array}\right]$. Thus, $z = V\bar{\Lambda}_r^{-1} w$ is an invertible change of coordinates. Proceeding as in the previous section, we have that the previous system is conjugate to

$$\left\{\begin{array}{rcl} \dot{z} & = & \Psi(U\bar{J}_r\bar{\Lambda}_r V^T z, u), \\ y & = & U\bar{J}_r\bar{\Lambda}_r V^T z + U_{\bar{r}} U_{\bar{r}}^T x_0, \\ z(0) & = & V\bar{\Lambda}_r^{-1}\bar{J}_r^T U^T x_0. \end{array}\right. \tag{6.2}$$

Again, all we have to do now is simplify the above expressions. To do this, we work out the following:

$$\begin{array}{rcl} \bar{J}_r\bar{\Lambda}_r & = & \left[\begin{array}{cc} I^{r\times r} & O \\ O & O^{(d-r)\times(n-r)} \end{array}\right]\left[\begin{array}{cc} \Lambda_r & O \\ O & I^{(n-r)\times(n-r)} \end{array}\right] \\ & = & \left[\begin{array}{cc} \Lambda_r & O \\ O & O^{(d-r)\times(n-r)} \end{array}\right] \\ & = & \Sigma. \end{array}$$

Using the above equality, we see that, in the first two equations of (6.2), we have

$$U\bar{J}_r\bar{\Lambda}_r V^T = U\Sigma V^T = B. \tag{6.3}$$

Similarly, we compute the expression

$$\begin{array}{rcl} \bar{\Lambda}_r^{-1}\bar{J}_r & = & \left[\begin{array}{cc} \Lambda_r^{-1} & O \\ O & I^{(n-r)\times(n-r)} \end{array}\right]\left[\begin{array}{cc} I^{r\times r} & O \\ O & O^{(d-r)\times(n-r)} \end{array}\right] \\ & = & \left[\begin{array}{cc} \Lambda_r^{-1} & O \\ O & O^{(d-r)\times(n-r)} \end{array}\right] = \Sigma^\dagger, \end{array}$$

which we can use in the third equation of (6.2) as we have

$$V\bar{\Lambda}_r^{-1}\bar{J}_r^T U^T = V\Sigma^\dagger U^T = B^\dagger. \tag{6.4}$$

We can also simplify the product $U_{\bar{r}}U_{\bar{r}}^T$ as follows. Because U is unitary, we have

$$\begin{aligned} I &= UU^T \\ &= \begin{bmatrix} U_r & U_{\bar{r}} \end{bmatrix} \begin{bmatrix} U_r^T \\ U_{\bar{r}}^T \end{bmatrix} \\ &= \left(\begin{bmatrix} U_r & O^{n\times(n-r)} \end{bmatrix} + \begin{bmatrix} O^{n\times r} & U_{\bar{r}} \end{bmatrix} \right) \left(\begin{bmatrix} U_r^T \\ O^{(n-r)\times n} \end{bmatrix} + \begin{bmatrix} O^{r\times n} \\ U_{\bar{r}}^T \end{bmatrix} \right) \\ &= U_r U_r^T + U_{\bar{r}} U_{\bar{r}}^T. \end{aligned}$$

Thus, $U_{\bar{r}}U_{\bar{r}}^T = I - U_r U_r^T$. If we define $\bar{I}_r = \begin{bmatrix} I^{r\times r} & O \\ O & O^{(d-r)\times(d-r)} \end{bmatrix}$, we have $U_r U_r^T = U\bar{I}_r U^T$. From the definition of the pseudoinverse of Σ, we have $\bar{I}_r = \Sigma\Sigma^\dagger$. Hence,

$$\begin{aligned} U_{\bar{r}}U_{\bar{r}}^T &= I - U\Sigma\Sigma^\dagger U^T \\ &= I - U\Sigma V V^T \Sigma^\dagger U^T \\ &= I - BB^\dagger. \end{aligned} \tag{6.5}$$

Taking the simplifications (6.3), (6.4), and (6.5) into account in (6.2), we have that

$$\Gamma_B : \begin{cases} \dot{x} = B\Psi(x,u), \\ y = x, \\ x(0) = x_0 \end{cases}$$

$$\text{is immersible into } \Xi_B : \begin{cases} \dot{z} = \Psi(Bz + (I - BB^\dagger)x_0, u), \\ y = Bz + (I - BB^\dagger)x_0, \\ z(0) = B^\dagger x_0, \end{cases}$$

which proves Theorem 6.1. □

Note that, if rank $B = d$, then $BB^\dagger = I$; so the correction terms $(I - BB^\dagger)x_0$ fall away and we recover Corollary 6.2. Also, if the initial condition x_0 is zero, these terms disappear again and we fall back onto Corollary 6.2. The result is thus applicable when Ψ is an arbitrary C^1 function from $\mathbb{R}^n \times \mathbb{R}^m$ to $\mathbb{R}^n$ satisfying a global Lipschitz condition.

6.2.4 Systems with a nonlinear output map

Theorem 6.1 holds when the state is directly accessible ($y = x$). We can extend this result immediately to the case of a nonlinear output map. Adding an arbitrary output map to the system Γ_B, we have the following proposition.

Proposition 6.3.

$$\left\{\begin{array}{rcl}\dot{x} &=& B\Psi(x,u),\\ y &=& g(x,u),\\ x(0) &=& x_0\end{array}\right. \quad \textit{is immersible into} \quad \left\{\begin{array}{rcl}\dot{z} &=& \Psi(Bz+(I-BB^\dagger)x_0,u),\\ y &=& g(Bz+(I-BB^\dagger)x_0,u),\\ z(0) &=& B^\dagger x_0.\end{array}\right.$$

But we see that the addition of an output map does not have any influence on the reasoning behind the proof of Theorem 6.1. Indeed, the output function g has no effect on the dynamics of the system. We can thus trivially extend Theorem 6.1 to Proposition 6.3.

6.3 Equivalence between neural networks with and without hidden layer

To give a clear picture of how our results apply to dynamical neural networks, we show that Theorem 6.1 specializes to the following illustrative cases. We consider a dynamical single–hidden-layer perceptron:

$$\left\{\begin{array}{rcl}\dot{x} &=& B\sigma(Ax+a_0),\\ y &=& x,\\ x(0) &=& x_0,\end{array}\right. \tag{6.6}$$

where $x \in \mathbb{R}^d$, $y \in \mathbb{R}^d$, B is a $d \times n$ matrix, A is a $n \times d$ matrix, a_0 is a $n \times 1$ vector, and σ is a sigmoidal activation function. We show the following corollary.

Corollary 6.4. *The system (6.6) is immersible into*

$$\left\{\begin{array}{rcl}\dot{z} &=& \sigma(ABz+A(I-BB^\dagger)x_0+a_0),\\ y &=& Bz+(I-BB^\dagger)x_0,\\ z(0) &=& B^\dagger x_0.\end{array}\right. \tag{6.7}$$

In particular, if rank $B = d$, then (6.6) is immersible into

$$\left\{\begin{array}{rcl}\dot{z} &=& \sigma(ABz+a_0),\\ y &=& Bz,\\ z(0) &=& B^\dagger x_0.\end{array}\right.$$

Note that, in our developments, we have not had to make any assumption about the function Ψ. Theorem 6.1 is in particular applicable when Ψ is an element-by-element activation function σ. We consider the system (6.6). Setting $\Psi = B\sigma(A.(.)+a_0)$ and using Theorem 6.1, we immediately get a proof of the above corollary.

6.4 Extension to networks with self-connections

Further, we also consider the following system with self-connections:

$$\left\{\begin{array}{rcl} \dot{x} & = & Cx + B\sigma(Ax + a_0), \\ y & = & x, \\ x(0) & = & x_0, \end{array}\right. \tag{6.8}$$

where $x \in \mathbb{R}^d$, $y \in \mathbb{R}^d$, C is a $d \times d$ matrix, B is a $d \times n$ matrix, A is a $n \times d$ matrix, a_0 is a $n \times 1$ vector. We consider the two cases of B having rank d and of B not being of maximal rank. If B has rank d, we obtain in the next subsection a corollary of Theorem 6.1. If rank $B = r < d$, the proposed method fails. We discuss in the following subsection under which additional restrictions we can still apply the proposed method. We also show that we can immerse the system (6.8) into a larger system with mixed element-by-element activation functions, so that the first result is directly applicable. Clearly, we can extend the previous results to dynamical single–hidden-layer perceptrons with inputs and with an output map, by making use of Proposition 6.3.

6.4.1 Networks with self-connections where rank $B = d$

We have the following corollary.

Corollary 6.5. *If rank $B = d$ then (6.8) is immersible into*

$$\left\{\begin{array}{rcl} \dot{z} & = & B^\dagger CBz + \sigma(ABz + a_0), \\ y & = & Bz, \\ z(0) & = & B^\dagger x_0. \end{array}\right. \tag{6.9}$$

Proof. If rank $B = d$ then $BB^\dagger = I$. If we set $D = B^\dagger C$, we can write $C = BD$. Thus, (6.8) is equal to

$$\left\{\begin{array}{rcl} \dot{x} & = & BDx + B\sigma(Ax + a_0), \\ y & = & x, \\ x(0) & = & x_0. \end{array}\right.$$

If we set $\Psi(.) = D.(.) + \sigma(A.(.) + a_0)$, we have

$$\left\{\begin{array}{rcl} \dot{x} & = & B\Psi(x), \\ y & = & x, \\ x(0) & = & x_0. \end{array}\right.$$

Thus, we have that (6.8) is immersible into

$$\left\{\begin{array}{rcl} \dot{z} & = & DBz + \sigma(ABz + a_0), \\ w & = & Bz, \\ z(0) & = & B^\dagger x_0, \end{array}\right.$$

which is equal to (6.9). This proves Corollary 6.5. □

6.4.2 Networks with self-connections where rank $B < d$

If rank $B < d$, the previous attack fails because we cannot write C as B times D, hence we cannot define Ψ as in the previous section. Instead of trying to use our main theorem, we could try to reapply the method of the proof to this problem. This approach fails in general, at least with the method we proposed. It is possible to show that a necessary and sufficient condition for the application of the proposed method is that there exists a matrix Q whose first r columns are a basis of some invariant subspace of both C and BA. This situation is in general impossible. One important case where it is possible is when $C = cI$, where c is some scalar (in this case, any subspace of C is invariant). We then have the following result:

$$\left\{\begin{array}{rcl} \dot{x} &=& cx + B\sigma(Ax + a_0), \\ y &=& x, \\ x(0) &=& x_0 \end{array}\right.$$

$$\text{is immersible into} \left\{\begin{array}{rcl} \dot{z} &=& cz + \sigma(ABz + A(I - BB^\dagger)e^{ct}x_0 + a_0), \\ y &=& Bz + (I - BB^\dagger)e^{ct}x_0, \\ z(0) &=& B^\dagger x_0. \end{array}\right.$$

Note that the differential equation is now time dependent (except if the initial condition is zero).
Another approach is to allow mixed activation functions. By this we mean that $\sigma(x)$ is still an element-by-element activation function, but $\sigma((x_1, \ldots, x_n)^T) = (\sigma_1(x_1), \ldots, \sigma_n(x_n))^T$, where each $\sigma_i (i = 1, \ldots, n)$ can be a different C^1 function from $\mathbb{R}$ to $\mathbb{R}$. We thus allow different activation functions for each state. Define an identity function ℓ such that $\ell(x) = x$. From (6.8), we have

$$\dot{x} = C\ell(Ix) + B\sigma(Ax + a_0) = \left[\begin{array}{cc} B & C \end{array}\right] \left[\begin{array}{c} \sigma(Ax + a_0) \\ \ell(Ix) \end{array}\right].$$

Defining the mixed activation function $\bar{\sigma}$ for $z \in \mathbb{R}^{n+d}$ as

$$\bar{\sigma}(z) = (\sigma(z_1), \ldots \sigma(z_n), \ell(z_{n+1}), \ldots, \ell(z_{n+d}))^T,$$

we get that (6.8) is conjugate to

$$\left\{\begin{array}{rcl} \dot{x} &=& \left[\begin{array}{cc} B & C \end{array}\right] \bar{\sigma}\left(\left[\begin{array}{c} A \\ I^{d\times d} \end{array}\right] x + \left[\begin{array}{c} a_0 \\ O^{d\times 1} \end{array}\right]\right), \\ y &=& x, \\ x(0) &=& x_0. \end{array}\right.$$

Setting $B' = \left[\begin{array}{cc} B & C \end{array}\right]$, $A' = \left[\begin{array}{cc} A; & I \end{array}\right]$, $a_0' = \left[\begin{array}{cc} a_0; & O^{1\times d} \end{array}\right]$, we have a system of the form (6.6) (except for the mixed activation functions, but we know from Theorem 6.1 that this is not a concern), which is immersible into

(6.7) (with mixed activation functions). We can avoid the problem of having a mixed activation function by noting that $\ell(z) \approx k\sigma(z/k)$, if k is a large scalar. Thus, defining $A'' = \begin{bmatrix} A; & (1/k)I \end{bmatrix}, B'' = \begin{bmatrix} B & kC \end{bmatrix}, a_0'' = a_0'$, we can approximate the output of (6.8) arbitrarily well with the output of

$$\begin{cases} \dot{z} &= \sigma(A''B''z + A''(I - B''B''^{\dagger})x_0 + a_0), \\ y &= B''z + (I - B''B''^{\dagger})x_0, \\ z(0) &= B''^{\dagger}x_0, \end{cases}$$

for sufficiently large k.

6.5 Dynamical system approximation with neural networks

In the preceding four sections, we have shown how to use different linear changes of coordinates to establish input–output equivalence between generalized neural networks with and without hidden layer. The most important application of such transformations is the proof that networks without hidden-layer but with an output map can approximate arbitrary dynamical systems. Such a universal approximation property is not obvious as the dynamical networks without hidden layer have a simple dynamics. The approximation property only comes thanks to the presence of an output map that projects the state trajectories onto a lower-dimensional space, thereby allowing more complex behaviors. We first review a theorem from dynamical system theory on how perturbations of the vector field affect the trajectories of a system. After this, we review a proof of the universal approximation property of Hopfield-type networks, which uses a transformation slightly different from the ones we discussed so far. We then move to the approximation of dynamical control systems by perceptron-type networks, which Sontag first discussed [131]; we modify his argument slightly to use Theorem 6.1.

6.5.1 Approximation of a dynamical system

An important elementary result from the theory of differential equations is the fact that two dynamical systems whose vector fields are close have trajectories that are close to each other. We use this fact to guarantee that, if we can approximate the vector field of a dynamical system by some static neural network, then the dynamical neural network having this static neural network as vector field will approximate the trajectories of the dynamical system under consideration. We have the following theorem.

Theorem 6.6. *Consider an open domain $\mathcal{D} \in \mathbb{R}^n$ and two Lipschitz continuous vector fields $f, g : \mathcal{D} \to \mathbb{R}^n$. Let the Lipschitz constant of the vector field f be K. Assume that f and g are uniformly close within ϵ:*

$$\|f(x) - g(x)\| < \epsilon \text{ for all } x \in \mathcal{D}.$$

If $x(t)$ and $y(t)$ are solutions to

$$\begin{cases} \dot{x} &= f(x), \\ x(0) &= x_0 \end{cases} \quad \text{and} \quad \begin{cases} \dot{y} &= f(y), \\ y(0) &= y_0, \end{cases}$$

for some time interval $[T_{\min}, T_{\max}]$, then the distance between the trajectories is bounded by the following expression:

$$\|x(t) - y(t)\| \leq \frac{\epsilon}{K} \left(e^{K t} - 1\right).$$

The proof uses an estimate of the distance between the two trajectories, which it bounds above through the use of the triangle inequality and of Gronwall's inequality, see [59]. In the case where the domain $\mathbb{D}$ is the whole of $\mathbb{R}^n$ and where the vector fields are globally Lipschitz with constant K, we have the following. We can guarantee that, if we give ourselves a maximum distance M between two trajectories that we are not allowed to exceed over a time span of T, we can choose a distance $\epsilon = MK/e^{KT}$ between two vector fields such that $\|f(x) - g(x)\| < \epsilon$ for all $x \in \mathbb{R}^n$ implies $\|x(t) - y(t)\| < M$ for all $t \leq T$. Thus, when approximating a dynamical system, if we can approximate its vector field closely, we can approximate its trajectories arbitrarily well for any *finite* time.

We should interpret this theorem with caution. Indeed, even if we are able to guarantee that we can approximate the trajectories of a dynamical system arbitrarily well, this fact does not imply that we can approximate this dynamical system arbitrarily well in all interesting senses. The main problem arises from the preservation of the important characteristics of a vector field such as its fixed points or its periodic orbits. It is possible to approximate a periodic orbit arbitrarily closely by a non-periodic orbit. If the periodic orbit is unstable we can achieve this non-periodic approximation with a dynamical system whose vector field is arbitrarily close to the vector field of the original system. This possibility implies that, although our approximation allows us to approximate trajectories arbitrarily well for any finite time, our approximation cannot guarantee that it preserves the periodic orbits of the system if they are unstable. The same statement holds for many other topological characteristics. This difficulty is related to the problem of structural stability [59] (structurally stable systems are systems for which all sufficiently close approximations have the same topological properties for their trajectories; they do *not* form a generic class of dynamical systems). The consequences of this difficulty for the

approximation of dynamical systems by neural networks does not seem to have been thoroughly studied.

A direct consequence of Theorem 6.6 is that dynamical single–hidden-layer perceptrons $\dot{x} = B\sigma(Ax + a_0)$ with sigmoidal activation function can universally approximate dynamical systems $\dot{x} = f(x)$ for a finite time because single–hidden-layer perceptrons $B\sigma(Ax + a_0)$ can universally and continuously approximate (Lipschitz) continuous functions $f(x)$. A rigorous proof of this statement requires some care in the case where we do not assume that f is globally Lipschitz, but it is only a somewhat tedious exercise (using a classical ϵ–δ argument, see [42]).

6.5.2 Hopfield-type networks

Funahashi and Nakamura [42] considered Hopfield-type networks:

$$\begin{aligned} \dot{v} &= \lambda v + W\sigma(v), \\ y &= (v_1, \ldots, v_d)^T, \end{aligned}$$

where $v \in \mathbb{R}^n, W \in \mathbb{R}^{n\times n}$, and $d \leq n$. They show that the output of such a network (for an appropriate choice of $n, d, \lambda \neq 0$, and W) can approximate an arbitrary dynamical system. They achieve this result by considering dynamical single–hidden-layer perceptrons with self-connections:

$$\left\{ \begin{aligned} \dot{x} &= \lambda x + B\sigma(Ax + a_0), \\ y &= x, \\ x(0) &= x_0, \end{aligned} \right. \tag{6.10}$$

where $x \in \mathbb{R}^d$, $y \in \mathbb{R}^d$, λ is a scalar, B is a $d \times n$ matrix, A is a $n \times d$ matrix, a_0 is a $n \times 1$ vector. They show that (6.10) is immersible into

$$\left\{ \begin{aligned} \dot{z} &= \lambda z - \lambda \bar{a}_0 + W\sigma(z), \\ y &= \begin{bmatrix} I^{d\times d} & O^{d\times n} \end{bmatrix} z, \\ z(0) &= \begin{bmatrix} x_0 & Bx_0 + a_0 \end{bmatrix}^T, \end{aligned} \right. \tag{6.11}$$

where $z \in \mathbb{R}^{d+n}$, $y \in \mathbb{R}^d$, $\bar{a}_0 = \begin{bmatrix} O^{d\times 1} \\ a_0 \end{bmatrix}$ and $W = \begin{bmatrix} O^{d\times d} & A \\ O^{n\times d} & AB \end{bmatrix}$.

By making λ sufficiently small, we know by Theorem 6.6 that the output of (6.11) can be arbitrarily well approximated by the output of the following system:

$$\left\{ \begin{aligned} \dot{z} &= \lambda z + W\sigma(z), \\ y &= \begin{bmatrix} I^{d\times d} & O^{d\times n} \end{bmatrix} z, \\ z(0) &= \begin{bmatrix} x_0 & Bx_0 + a_0 \end{bmatrix}^T. \end{aligned} \right. \tag{6.12}$$

Because (6.10) is immersible in (6.11), the system (6.12) can also approximate the output of (6.10). In fact, because λ has to be sufficiently small, the result amounts to showing that

$$\begin{cases} \dot{x} = B\sigma(Ax + a_0), \\ y = x, \\ x(0) = x_0 \end{cases} \text{ is immersible into } \begin{cases} \dot{z} = W\sigma(z), \\ y = \begin{bmatrix} I^{d\times d} & O^{d\times n} \end{bmatrix} z, \\ z(0) = \begin{bmatrix} x_0 & Bx_0 + a_0 \end{bmatrix}^T. \end{cases}$$

and neglecting the small contribution of λz. We can prove this relation of immersibility by setting $z = \begin{bmatrix} x \\ Ax + a_0 \end{bmatrix}$. The result of Funahashi and Nakamura is some kind of dual of ours, in the sense that the hidden matrix A is brought together with the output matrix B; while Theorem 6.1 does the transformation the other way round.

6.5.3 Dynamical perceptrons

We turn back to dynamical perceptrons with an output map,

$$\begin{aligned} \dot{z} &= \sigma(Wz + Vu), \\ y &= Qz, \end{aligned} \tag{6.13}$$

where $z \in \mathbb{R}^n, y \in \mathbb{R}^p, u \in \mathbb{R}^m$. Sontag [131] showed that these systems can approximate any dynamical control system,

$$\begin{aligned} \dot{x} &= f(x, u), \\ y &= h(x). \end{aligned} \tag{6.14}$$

The first step is to show that we can consider only the case of dynamical control systems with a linear output map. We prove this step by defining $v = (x; y)$ as the new state variable. Since $\dot{y} = (\partial h(x)/\partial x) f(x, u)$, the system (6.14) is immersible into

$$\begin{aligned} \dot{v} &= g(v, u), \\ y &= Cv, \end{aligned} \tag{6.15}$$

where $v \in \mathbb{R}^d, y \in \mathbb{R}^p, u \in \mathbb{R}^m$, and $g(v, u) = (f(x, u); (\partial h(x)/\partial x) f(x, u))$ and C is a projection matrix that keeps only the components of v corresponding to y. We can thus indeed limit our investigations to dynamical control systems with a linear output map.

The idea is now that we can approximate the system (6.15) by a dynamical single–hidden-layer network with a linear output map and then use Theorem 6.1. Note that, in everything we have written so far, we never manipulated the term a_0. We can therefore freely replace it by a time-dependent input term of the form $a_0 = Du(t)$, where $u \in \mathbb{R}^m$ and D is a $n \times m$ matrix. Moreover,

we can use Proposition 6.3 to treat any output map. Specifically, the system (for which we choose the number of neurons n and the matrices A, B, and D such that $B\sigma(Ax+Du)$, with $x \in \mathbb{R}^d$, approximates $g(x,u)$ within the desired accuracy)

$$\left\{ \begin{array}{rcl} \dot{x} & = & B\sigma(Ax+Du), \\ y & = & Cx, \\ x(0) & = & x_0 \end{array} \right.$$

is immersible into

$$\left\{ \begin{array}{rcl} \dot{w} & = & \sigma(ABw + A(I-BB^\dagger)x_0 + Du), \\ y & = & CBw + C(I-BB^\dagger)x_0, \\ z(0) & = & B^\dagger x_0. \end{array} \right.$$

This last system has almost the required form (6.13) except for the terms $A(I-BB^\dagger)x_0$ and $C(I-BB^\dagger)x_0$. To get rid of these constants, we use the classical trick that consists in introducing a supplementary variable that remains constant. Defining $z = (w; x_0)$, we have

$$\begin{array}{rcl} \dot{z} & = & \left(\begin{array}{c} \sigma\left(\left[\begin{array}{cc} AB & A(I-BB^\dagger) \end{array} \right] z + Du \right) \\ 0 \end{array} \right), \\ z(0) & = & \left(\begin{array}{c} B^\dagger x_0 \\ x_0 \end{array} \right), \\ y & = & \left[\begin{array}{cc} CB & C(I-BB^\dagger) \end{array} \right] z. \end{array}$$

Therefore, if we posit

$$\begin{array}{rcl} W & = & \left[\begin{array}{cc} AB & A(I-BB^\dagger) \\ O^{d\times n} & O^{d\times d} \end{array} \right], \\ V & = & \left[\begin{array}{cc} D & O^{p\times d} \end{array} \right], \\ Q & = & \left[\begin{array}{cc} CB & C(I-BB^\dagger) \end{array} \right] z, \end{array}$$

we obtain a dynamical perceptron (6.13) with $n+d$ neurons and with an output map that approximates the dynamical system (6.15) and, hence, the system (6.14). This approximation is possible if we have chosen A, B, and D so that $B\sigma(Ax+Du)$ approximates $g(x,u)$ within enough accuracy to guarantee that the trajectories of the dynamical single–hidden-layer perceptron approximate the trajectories of the dynamical control system within the desired accuracy.

6.5.4 Other dynamical neural networks without hidden layer

The argument of the previous subsection is applicable not only to dynamical perceptrons but to many other dynamical neural networks without hidden layer. As long as a generalized regularization network of the form $B\Psi(Ax+a_0)$ (with

Ψ a C^1 function from $\mathbb{R}^n \to \mathbb{R}^n$) has the universal approximation property for static functions, we can show that the dynamical networks

$$\begin{aligned} \dot{z} &= \Psi(Wz + Vu), \\ y &= Qz, \end{aligned}$$

can approximate arbitrary control systems (6.14). The procedure is exactly the same but uses Theorem 6.1 in its generality instead of its particular application to dynamical single–hidden-layer perceptrons (Corollary 6.4).

6.6 Conclusion

We have studied here conditions under which dynamical neural networks with and without hidden layer have equivalent input–output behaviors. The answer to this question was essential for the study of the capacity of neural networks to approximate dynamical control systems. We first introduced the concept of immersibility of a dynamical control system into another system of higher state dimension. This concept was based on the idea that an appropriately chosen subset of the higher-dimensional system could have a behavior equivalent to that of the lower-dimensional one. We then presented and proved a new general theorem showing how to compute linear changes of coordinates from a lower-dimensional system into a higher-dimensional one. This theorem involved a careful discussion of the rank of one of the matrices controlling the dynamics of the system.

We then applied this result to neural networks. We first considered dynamical single–hidden-layer perceptrons and showed that our theorem was directly applicable to these systems. After this, we considered dynamical single–hidden-layer perceptrons with self-connections, showing how they could be made input–output equivalent to dynamical networks without hidden layer. The application of our theorem required some care in this case and we discussed the conditions under which the theorem was applicable to this type of networks.

We finally recalled results on the approximation capabilities of dynamical neural networks [42, 131], which relied on changes of coordinates similar to those we presented. This presentation put our work into perspective within the field of neural networks. Our theorem on transdimensional changes of coordinates and its application to neural networks are original contributions of this dissertation.

Chapter 7

Lie algebra theory of nonlinear control systems

We consider here nonlinear control systems, but this time from the standpoint of differential-geometric methods. These methods are the most developed systematic way of analyzing nonlinear control systems. However, we will often have to restrict this analysis to what we call *control-affine systems*, where the input linearly influences the evolution of the state. We write these systems as

$$\begin{cases} \dot{x} &= f(x) + \sum_{i=1}^{m} g_i(x)u_i, \\ y &= h(x), \end{cases} \tag{7.1}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}^m, y \in \mathbb{R}^p$ and $f : \mathbb{R}^n \to \mathbb{R}^n, h : \mathbb{R}^n \to \mathbb{R}^p, g_i : \mathbb{R}^n \to \mathbb{R}^n (i = 1, \ldots, m)$ are C^∞ maps. The main references on differential geometric methods in control are the textbooks of Isidori [64], and Nijmeijer and van der Schaft [111]. Using H. Nijmeijer's words: "It was impossible not to be influenced by the trend-setting book 'Nonlinear Control Systems: An Introduction', written by A. Isidori in 1985." These methods give tools often based on an analysis of the local properties of a system and we can often patch these local properties together to obtain global results.

7.1 Differential-geometric control

We need to introduce a rather extensive mathematical apparatus, which explains the number of definitions in the next sections. We first introduce the notion of a directional derivative, which we call a Lie derivative from here on.

Definition 7.1. The *Lie derivative* or *directional derivative* of a real-valued function $V(x)$ with respect to a vector field $f(x)$ is a real-valued function equal to the inner product of f with the gradient of V; we denote it by $L_f V(x)$:

$$L_f V(x) = \nabla V(x) \cdot f(x) = \sum_{i=1}^{n} \frac{\partial V(x)}{\partial x_i} f_i(x).$$

The directional derivative (Fig. 7.1) gives the rate of change of the function V that we observe if we move along a trajectory of $\dot{x} = f(x)$; it maps the real-valued function $V(x)$ to another real-valued function $L_f V(x)$. An important remark is that we use the nabla operator ∇ to denote the gradient of function (when the operator acts on a real-valued function) as well as to denote the Jacobian of a map or a vector field (when the operator acts on a vector-valued function).

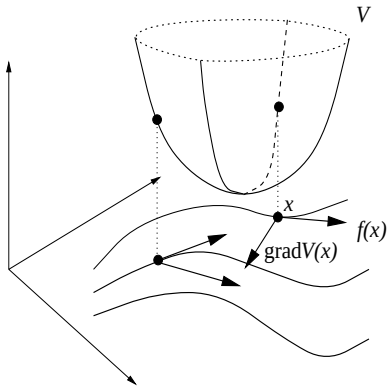


Figure 7.1: Lie derivative of a real-valued function. The dynamical system is two dimensional and evolves in the horizontal plane. The real-valued function resembles a quadratic function. We compute the Lie derivative at two points (black dots in the horizontal plane). The arrows give the vector field $f(x)$ and the gradient $\nabla V(x)$ at the point x. The scalar product of these two vectors is the Lie derivative at x.

An elementary application of the Lie derivative is as a notation for the condition that a Lyapunov function V for the system $\dot{x} = f(x)$ must see its value decrease along all trajectories of the system. In Lyapunov's second method (Theorem 3.1), we can replace the notation $\dot{V}(x) = \nabla V(x) \cdot f(x) \leq 0$ by $L_f V(x) \leq 0$. Although the notation might seem unnecessary in this case, it will prove useful when we study control-affine systems because of the multiple vector fields involved $(f, g_1, \ldots, g_m)$ and the iterated use of the Lie derivative. We can indeed iterate the application of the Lie derivative by writing

$$L_f^k V(x) = L_f L_f^{k-1} V(x) \text{ with } L_f^1 V(x) = L_f V(x).$$

This gives the kth derivative of V along a trajectory of $\dot{x} = f(x)$. We must remark that, although we can interpret the Lie derivative as a directional derivative along a trajectory, it is a property that is entirely determined locally. We can also compute iterated Lie derivatives for different vector fields, so $L_g L_f V(x)$ is a meaningful expression that computes the Lie derivative of V first along the vector field f and then along the vector field g. An important observation is that the computation of Lie derivatives is not commutative, so $L_g L_f V(x) \neq L_f L_g V(x)$ in general.

This noncommutativity leads us to introduce a second operation: The Lie bracket $[f, g]$ of two vector fields. For this, we also need the concept of a Lie algebra. A Lie algebra $\mathcal{A}$ is first of all an algebra, which means a family of functions that is closed under addition, scalar multiplication, and composition between elements of the algebra. In a Lie algebra, we call this composition law the Lie bracket and we require that it satisfies a number of properties. Bracketing is a bilinear, antisymmetric operation; which also satisfies the Jacobi identity.

Definition 7.2. A vector space $\mathcal{A}$ is a *Lie algebra* with *Lie bracket* $[.,.] : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ if, for all $A, B, C \in \mathcal{A}$, and $a \in \mathbb{R}$, we have

$$\begin{array}{ll} [A, B] = -[B, A], & \text{antisymmetry,} \\ [aA, B] = a[A, B], & \text{bilinearity,} \\ [A, B + C] = [A, B] + [A, C], & \text{bilinearity,} \\ [[A, B], C] + [[B, C], A] + [[C, A], B] = 0, & \text{Jacobi identity.} \end{array}$$

The bracket is an internal operation, so the bracket of two elements of the algebra is again an element of this algebra. In particular, we consider here the Lie algebra $\mathcal{V}$ that is the vector space of all smooth vector fields on $\mathbb{R}^n$; and the Lie bracket $[f, g]$ of two vector fields is itself a vector field. To the reader who is not acquainted with Lie algebra theory, this definition of a Lie algebra—having such unusual properties—might seem fairly gratuitous at first. But we can give an example that will clarify the usefulness of this definition. The vector product "$\wedge$" of two vectors u and v of $\mathbb{R}^n$ makes the Euclidean space a Lie algebra with Lie bracket $[u, v] = u \wedge v$. Indeed, the vector product is an internal bilinear operation. It is also antisymmetric: $u \wedge v = -v \wedge u$ (we could also call this property "anticommutativity", since commutativity requires $ab = ba$). The vector product has furthermore the property that $((u \wedge v) \wedge w) + ((v \wedge w) \wedge u) + ((w \wedge u) \wedge v) = 0$, which is the Jacobi identity (we could call this property "anti-associativity").

We can define a Lie bracket for the Lie algebra $\mathcal{V}$ of smooth vector fields in close correspondence with the Lie derivative that we defined before. We will investigate Lie algebras further in the next chapter and justify more completely our choice of the definition of the Lie bracket we present now. For two vector

fields f and g on $\mathbb{R}^n$, we define their bracket $[f,g](x)$ as the vector field whose components are

$$[f,g]_i(x) = \sum_{j=1}^{n} \left(\frac{\partial g_i}{\partial x_j} f_j(x) - \frac{\partial f_i}{\partial x_j} g_j(x) \right).$$

In gradient notation, we write $[f,g] = (\nabla g)f - (\nabla f)g$. We can iterate the computation of the Lie bracket and define for this purpose

$$\mathrm{ad}_f^k g = [f, \mathrm{ad}_f^{k-1} g] \text{ with } \mathrm{ad}_f^1 g = [f,g].$$

The relationship between the Lie bracket and the Lie derivative is that, for any real-valued function $V(x)$ and for any vector fields f and g, we have

$$L_{[f,g]}V(x) = L_f L_g V(x) - L_g L_f V(x),$$

which we can check by a direct computation. Other important properties of the Lie derivative and the Lie bracket are that

- If a is a real-valued function, f a vector field, and V a real-valued function, we have that

$$L_{af}V(x) = (L_f V(x))a(x).$$

- If a, b are real-valued functions and f, g are vector fields, we have that

$$[af, bg](x) = a(x)b(x)[f,g](x) + (L_f b(x))a(x)g(x) - (L_g a(x))b(x)f(x).$$

We now define the notion of a *covector field*, which is the object dual to a vector field. While a vector field f assigns a velocity vector $f(x)$ to any point x of $\mathbb{R}^n$, a covector field ω assigns a (different) linear map $\omega(x)$ from $\mathbb{R}^n \to \mathbb{R}$ to each point x of $\mathbb{R}^n$. The linear map $\omega(x)$ is called a covector because we can represent any linear map μ from $\mathbb{R}^n$ to $\mathbb{R}$ as $\mu(v) = (m_1, \ldots, m_n)(v_1, \ldots, v_n)^T$ for any $v \in \mathbb{R}^n$, so the map μ is uniquely determined by the vector $(m_1, \ldots, m_n)$. The set of all linear maps from $\mathbb{R}^n$ to $\mathbb{R}$ is called the dual of $\mathbb{R}^n$ and we denote it by $(\mathbb{R}^n)^*$. Some confusion arises from the fact that $(\mathbb{R}^n)^*$ is isomorphic to $\mathbb{R}^n$, as this isomorphism implies that a covector field ω formally looks like a vector field (except that it is a row vector instead of a column vector):

$$\omega(x) = [\omega_1(x_1, \ldots, x_n), \ldots, \omega_n(x_1, \ldots, x_n)],$$

with $\omega_i : \mathbb{R}^n \to \mathbb{R} (i = 1, \ldots, n)$; the main difference being that a vector field assigns a vector $f(x)$ to a point x while the covector $\omega(x)$, which the covector field assigns to x, will always act further on some other vector to give finally a scalar value. An example that should clarify this discussion is

that of considering the gradient operator as a covector field. The gradient of the function $V : \mathbb{R}^n \to \mathbb{R}$ is, as a covector field that we evaluate at $v = (v_1, \ldots, v_n)^T$,

$$(\nabla V(x))(v) = \left(\frac{\partial V(x)}{\partial x_1}, \ldots, \frac{\partial V(x)}{\partial x_n} \right) (v_1, \ldots, v_n)^T. \tag{7.2}$$

This notation means that, at the point x, we can compute the partial derivative (which is a scalar) of V in the direction of v by the previous expression (7.2). We can therefore have this gradient covector field act on a vector field $f(x)$ to produce a real-valued function. We have in this way redefined the Lie derivative $L_f V(x) = \nabla V(x)(f(x))$.

After the Lie derivative of a real-valued function along a vector field, which is a real-valued function, we define the *Lie derivative of a covector field*, which is the covector field $L_f \omega(x) = f^T(x)\big(\nabla \omega^T(x)\big)^T + \omega(x)\nabla f(x)$. An important property of covector fields and Lie derivatives is that, if V is a real-valued function and f is a vector field, we have that

$$L_f \nabla V(x) = \nabla(L_f V(x)). \tag{7.3}$$

With this arsenal of new tools, we can tackle a central result from differential-geometric control, which is Frobenius's theorem.

7.2 Frobenius's theorem

The main difference between a dynamical system and a control system is that the behavior of a dynamical system $\dot{x} = f(x)$ is uniquely determined at each point x by the velocity vector $f(x)$, while the dynamics of a control system $\dot{x} = f(x, u)$ at a point x can be any vector $f(x, u)$ produced by an admissible control u. The object that properly captures this difference is called a distribution. Consider the case of an control-affine system without drift: $\dot{x} = \sum_{i=1}^m g_i(x)u_i$. The distribution of this set of vector fields is the function that assigns, to any point x, the vector space of all possible velocity vectors $\sum_{i=1}^m g_i(x)u_i$. There is a close relationship between the local controllability properties of the system and its distribution.

Definition 7.3. The *smooth distribution* of a set of smooth vector fields $g_i(x) : \mathbb{R}^n \to \mathbb{R}^n, (i = 1, \ldots, m)$ is the function $\mathcal{G} : \mathbb{R}^n \to \mathcal{S}$ (where $\mathcal{S}$ is the set of all linear subspaces of $\mathbb{R}^n$) such that $\mathcal{G}(x) = \text{span}\{g_1(x), \ldots, g_m(x)\}$.

We can define a number of operations on distributions. If $\mathcal{G}(x)$ and $\mathcal{H}(x)$ are smooth distributions, their sum $(\mathcal{G} + \mathcal{H})(x)$ is such that $(\mathcal{G} + \mathcal{H})(x) = \mathcal{G}(x) \oplus \mathcal{H}(x)$, where "$\oplus$" denotes the direct sum of two vector spaces. Their intersection $(\mathcal{G} \cap \mathcal{H})(x)$ is such that $(\mathcal{G} \cap \mathcal{H})(x) = \mathcal{G}(x) \cap \mathcal{H}(x)$. We say that $\mathcal{G}$

contains $\mathcal{H}$ if $\mathcal{G}(x) \supset \mathcal{H}(x)$ for all x. Finally, the dimension of the distribution $\mathcal{G}$ at the point x is the dimension of the subspace $\mathcal{G}(x)$ and we have that $\dim(\mathcal{G})(x) = \text{rank}([g_1(x), \ldots, g_m(x)])$. We can now define the concept of a nonsingular distribution.

Definition 7.4. A distribution $\mathcal{G}$ is *nonsingular* if there is an integer d such that

$$\dim(\mathcal{G})(x) = d \text{ for all } x.$$

If not, we say that the distribution is singular or of variable dimension.

Now, we say that a vector field $v(x)$ belongs to the distribution $\mathcal{G}$ if the vector $v(x) \in \mathcal{G}(x)$ for all x: The velocity vector is, at all points, in the span of the vector fields of the distribution. This constraint on the velocity vector means that there exists a feedback law $u(x)$ such that the behavior of the system $\sum_{i=1}^{m} g_i(x)u_i$ under feedback is identical to the behavior of $\dot{x} = v(x)$. We are now led to the notion of an involutive distribution.

Definition 7.5. A distribution $\mathcal{G}$ is *involutive* if the bracket of any two vector fields v_1, v_2 belonging to $\mathcal{G}$ belongs itself to $\mathcal{G}$: It is involutive if

$$v_1(x), v_2(x) \in \mathcal{G}(x) \text{ for all } x \text{ implies } [v_1, v_2](x) \in \mathcal{G}(x) \text{ for all } x.$$

An exciting property of involutive distributions is that we can consider them as Lie algebras. Indeed, the definition of involutivity and of the appertaining of a vector field to a distribution implies that $[v_1, v_2] \in \mathcal{G}$ if $v_1, v_2 \in \mathcal{G}$. This closure under the Lie bracket means that any involutive distribution $\mathcal{G}$ is a Lie subalgebra of $\mathcal{V}$. Since any v belonging to $\mathcal{G}$ is such that $v(x)$ is some linear combination of $g_1(x), \ldots, g_m(x)$ (this linear combination being dependent on x) and since the Lie bracket is a bilinear operation, it is enough to check that

$$[g_i, g_j] \in \mathcal{G} \text{ for all } i \neq j; i = 1, \ldots, n; j = 1, \ldots, n$$

to guarantee that a distribution is involutive. Now, we want to relate this property of involutivity to the local behavior of the system $\dot{x} = \sum_{i=1}^{m} g_i(x)u_i$. The Frobenius theorem shows that under the condition of involutivity, the system is surprisingly well-behaved. To see this, we need to define the concept of an integrable distribution. We first recall that we can describe a d-dimensional surface or manifold in n-dimensional space locally around a point x by a function m from an open subset $\mathcal{K}$ of $\mathbb{R}^d$ to $\mathbb{R}^n$. This map gives us a local coordinate system for the d-dimensional manifold.

Definition 7.6. A manifold $\mathcal{M}$ with coordinate system $m : \mathcal{K} \subset \mathbb{R}^d \to \mathbb{R}^n$ is an *integral manifold* of a nonsingular distribution $\mathcal{G}$ of dimension d around the point $x = m(0)$ if $\nabla m(z) = \mathcal{G}(m(z))$ for every $z \in \mathcal{K}$.

This condition means that, steering our system into whichever direction $\sum_{i=1}^{m} g_i(x)u_i$, we always stay on the surface because, at any point, the hyperplane tangent to the surface is equal to the span of the control vectors $g_1(x), \ldots, g_m(x)$.

Definition 7.7. A distribution $\mathcal{G}$ is *integrable* if, at any point x, there is an integral surface going through this point.

The fact that the distribution as a whole is integrable forces the integral surfaces to connect to each other, such that the whole space is partitioned into a set of d-dimensional surfaces, see Fig. 7.2. We call such a partition a *foliation*. Frobenius's theorem allows us to check the integrability of a system easily by stating the equivalence between integrability and involutivity.

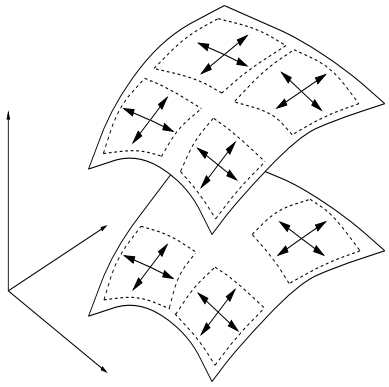


Figure 7.2: Integrable system and its associated integral manifolds. The system has a three-dimensional state space. The arrows around a point in state space indicate in which direction the state of the system can move under the action of the input. The small dashed patches are local integral surfaces going through a point. Because the system is integrable, the integral manifolds connect to each other. The state-space is partitioned into a foliation.

Theorem 7.1 (Frobenius). *A nonsingular distribution is integrable if and only if it is involutive.*

We refer to Isidori [64] for the proof of the theorem. The notions of integrability and involutivity are in no sense limited to nonlinear control theory. Essentially equivalent concepts play a major role in the study of ordinary differential equations and of partial differential equations. If we can associate an integrable distribution to a control system, powerful tools are available for its analysis.

7.3 Controllability and observability

In this section, we look at the problem of controllability and observability of nonlinear systems from the point of view of differential-geometric control. We shall see that this leads to a discussion that is much different from that of Chapter 5. When we look at a distribution and observe that it is not involutive, we may want to force involutivity anyway. We achieve this result by adding to the distribution the brackets that do not belong to it, in the hope that the resulting distribution is involutive. If this is not the case, we repeat the operation until we get an involutive distribution, which we call the involutive closure.

Definition 7.8. The *involutive closure* of a distribution $\mathcal{G}_0$ is the minimal involutive distribution $\mathcal{G}$ that contains $\mathcal{G}_0$. The involutive closure is the intersection of all the involutive distributions that contain $\mathcal{G}_0$.

7.3.1 Rank conditions

With this concept of involutive closure, we can tackle the problem of the controllability of nonlinear control systems, which we state in terms of the controllability rank condition [56]. We do not limit ourselves to affine systems but we consider any system of the form

$$\begin{aligned} \dot{x} &= f(x,u), \\ y &= h(x), \end{aligned} \tag{7.4}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}^m, y \in \mathbb{R}^p$.

A problem with the notion of controllability is that it is not symmetric: It may be that we can steer x_1 into x_2, while no input can steer x_2 into x_1. We want to make the relation symmetric and define to this end the notion of weak controllability. Moreover, controllability is a global concept, which means that the control path between two points that are close to each other is not guaranteed to be short; we therefore restrict the concept of controllability to a local one. We say that a system is *locally controllable* if we can steer it between two neighboring points while always staying in a neighborhood of the initial condition. When we say that a system is *weakly controllable*, we do not require that we can steer any point into any another, but only that we can find a chain connecting these two points where we can steer either $x^{(j)}$ into $x^{(j+1)}$ or $x^{(j+1)}$ into $x^{(j)}$. This property is weaker than controllability. Indeed, if we can find a chain where we can always steer from $x^{(j)}$ to $x^{(j+1)}$, we have controllability; so controllability implies weak controllability. The justification for defining these less intuitive notions is that they are easier to check. Indeed, we have the following rank condition.

Definition 7.9. A control system (7.4) satisfies the *controllability rank condition* at the point x if $\dim(\mathcal{F}(x)) = n$, where $\mathcal{F}$ is the involutive closure of the distribution $\mathcal{F}_0(x) = \text{span}\{f(x,u) \text{ for any } u \in \mathbb{R}^m\}$. We say that the system as a whole satisfies the controllability rank condition if it satisfies it at all points.

Proposition 7.2 (Hermann and Krener). *A control system (7.4) is locally weakly controllable at x if it satisfies the controllability rank condition at x. Moreover, if the system is analytic, local weak controllability at every point x is equivalent to weak controllability.*

See [56] for the proof of this proposition. To tackle the problem of observability, we need a criterion dual to the controllability rank condition, this dual criterion is the observability rank criterion. We need to work here with the object dual to a distribution, which we call a codistribution.

Definition 7.10. The *smooth codistribution* of a set of smooth covector fields $\omega_i(x) : \mathbb{R}^n \to (\mathbb{R}^n)^*, (i = 1, \dots, m)$ is the function $\mathcal{H} : \mathbb{R}^n \to \mathcal{S}^*$ (where $\mathcal{S}^*$ is the set of all subspaces of $(\mathbb{R}^n)^*$) such that $\mathcal{H}(x) = \text{span}\{\omega_1(x), \dots, \omega_m(x)\}$.

To construct the necessary codistribution, we consider first the set containing all the output functions $\mathcal{H}_0 = \{h_1(x), \dots, h_p(x)\}$. The output map $h(x)$ was a C^∞ map from $\mathbb{R}^n$ to $\mathbb{R}^p$, so the $h_i(x), (i = 1, \dots, p)$ are C^∞ maps from $\mathbb{R}^n$ to $\mathbb{R}$ (therefore, we cannot directly speak about distributions here). If we compute a Lie derivative of such a map, we still obtain a C^∞ map. We then find the smallest linear subspace $\mathcal{H}$ of the set of all smooth maps $C^\infty(\mathbb{R}^n, \mathbb{R})$ such that it contains $\mathcal{H}_0$ and it is closed with respect to the Lie differentiation of its elements with respect to the elements of $\mathcal{F}_0$. The set $\mathcal{H}$ is such that

1. $\mathcal{H} \subset C^\infty(\mathbb{R}^n, \mathbb{R}), \mathcal{H}_0 \subset \mathcal{H}$.
2. $h, l \in \mathcal{H}$ implies $\alpha h + \beta l \ \in \mathcal{H}$ for all $\alpha, \beta \in \mathbb{R}$.
3. For all $h \in \mathcal{H}$ and $f \in \mathcal{F}_0$, we have $L_f h \in \mathcal{H}$.
4. If $\mathcal{H}_1$ and $\mathcal{H}_2$ satisfy the above properties, then $\mathcal{H} \subset (\mathcal{H}_1 \cap \mathcal{H}_2)$.

Now that we have defined $\mathcal{H}_0$ and $\mathcal{H}$, we define further the "gradients" of these sets. These gradient sets are codistributions: $\nabla\mathcal{H}_0(x) = \{\nabla h(x) : h \in \mathcal{H}_0\}$ and $\nabla\mathcal{H}(x) = \{\nabla h(x) : h \in \mathcal{H}\}$. Since $\nabla(L_f h) = L_f(\nabla h)$ by the properties (7.3) of the Lie derivative of covector fields and that $L_f h$ belongs to $\mathcal{H}$ ($\mathcal{H}$ is by definition closed under Lie differentiation), the codistribution $\nabla\mathcal{H}$ is the smallest codistribution that contains $\nabla\mathcal{H}_0$ and is closed under Lie differentiation of covector fields with respect to the elements of $\mathcal{F}_0$.

Now, we are in a position to consider the observability rank condition, which is directly related to local weak observability. Again, observability is

a global concept: The input that distinguishes between two different initial conditions that are close to each other may have to bring the states far from the initial conditions to distinguish between them. We therefore define a stronger local concept. Similarly as before, we weaken the concept of observability by requiring only the possibility to distinguish a point from its neighbors. A control system is *locally weakly observable* at a point if, for every small enough neighborhood of this point, it is always possible to find a control that steers the point and any of its neighbors into states where they produce different outputs.

Definition 7.11. A control system (7.4) satisfies the *observability rank condition* if, for all x, we have that $\dim(\mathcal{H}(x)) = n$, where $\mathcal{H}$ is the closure under Lie differentiation with respect to the elements of $\mathcal{F}_0$ of the distribution $\mathcal{H}_0(x) = \operatorname{span}\{h_i(x) \ : i = 1, \ldots, p\}$.

Proposition 7.3 (Hermann and Krener). *A control system (7.4) is locally weakly observable at a point x if it satisfies the controllability rank condition at x. Moreover, if the system is analytic, local weak observability at every point x is equivalent to weak observability.*

For the proof, we refer to [56].

7.3.2 Application to neural networks

Żbikowski [150, 151] investigated the application of these controllability and observability rank conditions to recurrent neural networks. He considered single-input–single-output networks of the form

$$\begin{aligned} \dot{x} &= -x + \sigma(Ax) + bu, \\ y &= cx; \\ x(0) &= x_0, \end{aligned} \tag{7.5}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}, y \in \mathbb{R}$; and the matrix A, the vectors b, c are of corresponding dimension. Let us consider first the question of the controllability of such a network. We start by building the distribution $\mathcal{F}_0(x) = \operatorname{span}\{f_u(x) = -x + \sigma(Ax) + bu : u \in \mathbb{R}\}$. Note that the distribution is spanned by an uncountable number of vector fields, although its dimension at each point is at most n (all vector fields live in an n-dimensional space). We then build the involutive closure $\mathcal{F}$ of $\mathcal{F}_0$ by taking the iterated Lie brackets of all the elements of $\mathcal{F}_0$. For convenience, define $\rho(x) = \sigma(Ax)$. So, we have that the following functions must be spanning $\mathcal{F}$:

- The functions in $\mathcal{F}_0$ ($u \in \mathbb{R}$),

$$f_u = -x + \rho + bu$$

- The bracket of all pairs of functions in $\mathcal{F}_0$ ($u, v \in \mathbb{R}$),

$$\begin{aligned}[f_u, f_v] &= b(v-u) + [b,\rho](u-v) \\ &= (b - \mathrm{ad}_b^1\rho)(v-u)\end{aligned}$$

- The second-order brackets ($u, v, w \in \mathbb{R}$),

$$\begin{aligned}[f_w,[f_u,f_v]] = &\Big(b + \mathrm{ad}_x^1(\mathrm{ad}_b^1\rho) + \mathrm{ad}_\rho b \\ &+\mathrm{ad}_\rho^2 b - (\mathrm{ad}_b^2\rho)w\Big)(v-u)\end{aligned}$$

- The two types of third-order brackets ($u, v, w, r \in \mathbb{R}$),

$$\begin{aligned}[[f_u,f_v],[f_w,f_r]] &= 0, \\ [f_r,[f_w,[f_u,f_v]]] &= \big(b - \mathrm{ad}_x^2(\mathrm{ad}_b^1\rho) - \mathrm{ad}_x^1(\mathrm{ad}_\rho^1 b) \\ &\quad -\mathrm{ad}_x^1(\mathrm{ad}_\rho^2 b) - (\mathrm{ad}_x^1(\mathrm{ad}_b^2\rho))w \\ &\quad +\mathrm{ad}_\rho^2 b + \mathrm{ad}_\rho^1(\mathrm{ad}_x^1(\mathrm{ad}_b^1\rho)) \\ &\quad +\mathrm{ad}_\rho^2 b + \mathrm{ad}_\rho^3 b - (\mathrm{ad}_\rho^1(\mathrm{ad}_b^2\rho))w \\ &\quad +(\mathrm{ad}_b^1(\mathrm{ad}_x^1(\mathrm{ad}_b^1\rho)))r + (\mathrm{ad}_b^1(\mathrm{ad}_\rho^1 b))r \\ &\quad +(\mathrm{ad}_b^1(\mathrm{ad}_\rho^2))r - (\mathrm{ad}_b^3\rho)wr\big)(v-u)\end{aligned}$$

- All higher-order brackets

The higher-order brackets become fast too complex too handle and there is no way to determine what the involutive distribution $\mathcal{F}$ is. But, what we are looking for are conditions on the parameters A and b of the network (7.5) that are enough to guarantee that the distribution $\mathcal{F}$ has dimension n at every point x of $\mathbb{R}^n$. Although Żbikowski's discussion [151] is not absolutely conclusive (the difficulty is that we do not need to consider all the vector fields in $\mathcal{F}$, but we need to find—at each point—a set of control vectors whose span is $\mathbb{R}^n$), this problem appears intractable at this point. The same is true for the question of observability, where we need to determine the codistribution $\nabla\mathcal{H}$. This difficulty suggests that the controllability rank condition and the observability rank condition, which are part of the apparatus of nonlinear control theory, are inappropriate tools for the study of the observability and controllability properties of neural networks. This observation is in contrast with the results from Chapter 5, where we were able to determine conditions for observability, controllability, and uniqueness of parameters for some interesting classes of neural networks—at least under some nondegeneracy conditions on the control matrix B.

7.4 Inverses of recurrent neural networks

We turn now to local properties of single-input–single-output (SISO) systems. After the *revers de fortune* of the previous section, we show that the tools from

nonlinear control theory are useful for studying the conditions under which neural networks possess a local inverse.

7.4.1 Relative degree and inverses of nonlinear systems

The first tool we need is that of the relative degree of a nonlinear SISO system (7.4) (note that here $u \in \mathbb{R}$ and $y \in \mathbb{R}$) and of a control affine SISO system (7.1)

$$\begin{cases} \dot{x} &= f(x) + g(x)u, \\ y &= h(x), \end{cases} \tag{7.6}$$

with $u \in \mathbb{R}$, and $y \in \mathbb{R}$. The concept of relative degree for nonlinear systems is an extension of the notion of relative degree for linear systems. For a linear system, the relative degree is the difference between the degree of the denominator and the numerator of the transfer function of the system. The relative degree captures the way the system integrates the input before influencing the output explicitly. If the nonlinear control system (7.4) is SISO, control-affine, and analytic; we can define the relative degree as follows [128].

Definition 7.12. The *relative degree* of a control-affine system (7.1) at a point x_0 is the integer d such that

1. There exists a neighborhood $\mathcal{U}$ of x_0 for which $L_g L_f^k h(x) = 0$ for every $x \in \mathcal{U}$ and all positive $k < d - 1$.

2. $L_g L_f^{d-1} h(x_0) \neq 0$.

The relative degree is infinite if there exists a neighborhood $\mathcal{U}$ of x_0 for which $L_g L_f^k h(x) = 0$ for every $x \in \mathcal{U}$ and for all $k \in \mathbb{N}$. The relative degree is undefined if there is no such neighborhood but $L_g L_f^k h(x_0) = 0$ for all $k \in \mathbb{N}$. We say that a system has a *global relative degree of* d if it has relative degree d at every point.

It is important to remember that the concept of relative degree is essentially a local concept; assuming that a system admits a global relative degree is not innocuous. For example, the system $\dot{x} = u, y = \sin(x)$ has a relative degree of one at every $x_0 \neq \pi/2 + k\pi$, but the relative degree is undefined at $x_0 = \pi/2 + k\pi$. The justification for our definition comes from looking at how the input u influences the output y of the system. Suppose that we have a system of relative degree d at x_0. Looking at the output $y|_{x=x_0}$ of the system when it is in the state $x = x_0$, we have

$$y = h(x)$$

and, in this relation, the input does not appear explicitly. If we look at the first derivative of the output, we have

$$\dot{y} = \nabla h(x)\dot{x} = \nabla h(x)f(x) + (\nabla h(x)g(x))u = L_f h(x) + (L_g h(x))u,$$

which, in a neighborhood $\mathcal{U}$ where the conditions of Definition 7.12 hold, degenerates to

$$\dot{y} = L_f h(x), \quad \text{for all } x \in \mathcal{U},$$

if the relative degree is superior or equal to two—since then $L_g h(x) = 0$. Thus, the input does then not appear in the expression of the first derivative of the output. Looking at the second derivative (restricted to the neighborhood $\mathcal{U}$), we have

$$\begin{aligned} \ddot{y} &= \tfrac{d}{dt} L_f h(x) = \tfrac{d}{dx}(L_f h(x))\tfrac{dx}{dt} \\ &= \nabla(L_f h(x))(f(x) + g(x)u) = L_f^2 h(x) + (L_g L_f h(x))u, \end{aligned}$$

which degenerates to

$$\ddot{y} = L_f^2 h(x), \quad \text{for all } x \in \mathcal{U},$$

if the relative degree is superior or equal to three. Repeating this further, we have

$$\frac{d^k y}{dt^k} = L_f^k h(x), \quad \text{for all } x \in \mathcal{U}, \quad \text{for all } k < d;$$

and

$$\left.\frac{d^d y}{dt^d}\right|_{x=x_0} = L_f^d h(x_0) + L_g L_f^{d-1} h(x_0) u.$$

These relations show that the input appears explicitly for the first time only in the dth derivative of the input. With this definition of the relative degree, we obtain an inversion theorem for control-affine systems.

Theorem 7.4 (Hirschorn). *A control-affine system (7.6) of global relative degree d possesses a left inverse of the form*

$$\dot{z}(t) = f(z(t)) + g(z(t))v(t), \tag{7.7}$$

where

$$v(t) = \frac{(d^d y(t)/dt^d) - L_f^d h(z(t))}{L_g L_f^{d-1} h(z(t))} \tag{7.8}$$

and the initial condition of the inverse system is equal to the initial condition of the original system: $z(0) = x(0)$.

This theorem means the following. If the control-affine system (7.6) produces the output $y(t)$ when driven by the input $u(t)$, the inverse system (7.7) will produce an output $z(t)$ equal to $u(t)$—if we drive it with an input $v(t)$ computed by Equation 7.8. As $v(t)$ depends on $y(t)$, the system given by (7.6) and (7.7) taken together produces at its output the input $u(t)$ of the control-affine system (7.6) when driven by the output $y(t)$ of the system (7.6). We refer to [60] for the proof of this theorem.

7.4.2 Application to Hopfield-type networks

We demonstrate the usefulness of this inversion theorem by applying it to the following Hopfield-type neural network:

$$\begin{aligned} \dot{x} &= \Lambda x + A\sigma(x) + bu, \\ y &= \sigma(x_1), \end{aligned} \tag{7.9}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}, y \in \mathbb{R}$. This neural network is a control-affine system and we set $f(x) = \Lambda x + A\sigma(x), g(x) = b, h(x) = \sigma(x_1)$. Since we have Hirschorn's inversion theorem and we want to find the inverse of the neural network (7.9), we need to determine its relative degree (and make sure it is finite) and compute its iterated Lie derivatives $L_f^d h$ and $L_g L_f^{d-1} h$. We have then, for example,

$$\begin{aligned} L_g h(x) &= g_1(x)\sigma'(x_1), \\ L_g L_f h(x) &= \sigma'(x_1)\nabla f_1(x) g(x) + \sigma''(x_1) f_1(x) g_1(x), \end{aligned}$$

for the first terms we have to check. We need to give conditions for which all terms $L_g L_f^k h(x)$ are uniformly zero up to degree $d-1$. We prove the following proposition, which makes the restrictive assumption that $b_1 = b_2 = \cdots = b_{n-1} = 0$ and $b_n \neq 0$. We use this assumption to make the computations easier, but a similar result holds without this restrictive hypothesis [67].

Proposition 7.5 (Kambhampati et al.). *Consider a neural network (7.9) with an analytic activation function σ that is strictly increasing ($\sigma'(\xi) > 0$ for all $\xi \in \mathbb{R}$). Assume further that $b_1 = b_2 = \cdots = b_{n-1} = 0$ and $b_n \neq 0$. If the matrix A of weights has, for some d such that $3 \leq d \leq n$, an upper block of zeros of the form*

$$\begin{aligned} &A_{13} = A_{14} = \cdots = A_{1n} = 0, \\ &A_{24} = A_{34} = \cdots = A_{2n} = 0, \\ &\vdots \\ &A_{d-3,d-1} = \cdots = A_{d-3,n} = 0, \\ &A_{d-2,n} = 0, \end{aligned}$$

and that

$$A_{d-2,d}A_{d,n} \neq 0, \quad A_{d-2,d+1}A_{d+1,n} \neq 0, \quad \ldots, \quad A_{d-1,n}A_{nn} \neq 0;$$

then the neural network has a global relative degree of d.

Proof. Following the argument in [67], let us first remark that, if we want to have a relative degree of one, it suffices to set $b_1 \neq 0$, as $L_g h(x) = b_1 \sigma'(x_1) \neq 0$. Now, according to our assumptions, $b_1 = 0$, so $L_g h(x) \equiv 0$. We have further that $L_g L_f h(x) = \sigma'(x_1) A_{1n} \sigma'(x_n) b_n$ (taking into account the fact that $b = (0, \ldots, 0, 1)^T$ and $L_f h(x) = f_1(x)\sigma'(x_1)$). If $A_{1n} \neq 0$, then $L_g L_f h(x) \neq 0$ and the relative degree is two (this condition gives a method to generate a network with relative degree two); else $L_g L_f h(x)$ is identically zero and the relative degree is superior to two. Supposing now that the relative degree is superior to two and thus $A_{1n} = 0$, we compute $L_g L_f^2 h(x)$ to determine conditions for a relative degree of three or more. We first compute

$$L_f^2 h(x) = \nabla(f_1(x)\sigma'(x_1)) f(x) = \sigma'(x_1) \nabla(f_1(x)) f(x) + f_1^2(x)\sigma''(x_1), \quad (7.10)$$

and then, using $A_{1n} = 0$ so that $f_1(x) = \lambda_1 x_1 + A_{11}\sigma(x_1) + \cdots + A_{1,n-1}\sigma(x_{n-1})$ is independent of x_n,

$$\begin{aligned} L_g L_f^2 h(x) &= \frac{\partial}{\partial x_n}(L_g L_f h(x)) b_n = \sigma'(x_1) \nabla(f_1(x)) \frac{\partial f(x)}{\partial x_n} b_n \\ &= b_n \sigma'(x_1)\sigma'(x_n)(A_{12} A_{2n} \sigma'(x_2) + \cdots + A_{1,n-1} A_{n-1,n} \sigma'(x_{n-1})). \end{aligned}$$

This last expression is zero if $A_{2n} = \cdots = A_{n-1,n} = 0$, or if $A_{2n} = 0$ and $A_{13} = \cdots = A_{1,n-1} = 0$. The first condition leads to an infinite relative degree (since then $A_{1n} = \cdots = A_{n-1,n} = 0$, so that the dynamics of $x_1, \ldots, x_{n-1}$ is independent from the dynamics of x_n and thus $y = \sigma(x_1)$ is independent of u). We therefore assume

$$A_{2n} = 0 \text{ and } A_{13} = \cdots = A_{1n} = 0, \quad (7.11)$$

which guarantees a relative degree of three or more.

Although mathematically correct, the above condition is seriously restrictive. What we actually need is $A_{12} A_{2n} = \cdots = A_{1,n-1} A_{n-1,n} = 0$, which both of the above conditions guarantee—but there are many other possible conditions, such as $A_{12} = A_{3n} = A_{4n} = \cdots = A_{n-1,n} = 0$. Thus, the condition of the theorem will be sufficient to guarantee a given relative degree, but not necessary. Unfortunately, we cannot fix this weakness in the proof because each separate condition gives rise to a separate discussion of the relative degree, which becomes too complex as we go to higher relative degrees.

Now, we want to achieve a relative degree of four or more. Using our assumption (7.11) to simplify the expression (7.10), we obtain

$$\begin{aligned} L_f^2 h(x) = \; & \sigma''(x_1)(\lambda_1 x_1 + A_{11}\sigma(x_1) + A_{12}\sigma(x_2))^2 \\ & + \sigma'(x_1)(\lambda_1 + A_{11}\sigma'(x_1))(\lambda_1 x_1 + A_{11}\sigma(x_1) + A_{12}\sigma(x_2)) \\ & + \sigma'(x_1) A_{12} \sigma'(x_2) f_2(x). \end{aligned}$$

After some computations, we get for the fourth-order Lie derivative

$$\begin{aligned} L_g L_f^3 h(x) = \; & b_n \sigma'(x_1)\sigma'(x_2)\sigma'(x_n) A_{12} \\ & \times (A_{23}A_{3n}\sigma'(x_3) + A_{24}A_{4n}\sigma'(x_4) + \cdots + A_{2,n-1}A_{n-1,n}\sigma'(x_n)), \end{aligned}$$

which we can make equal to zero by requiring

$$A_{3n} = 0 \text{ and } A_{24} = \cdots = A_{2,n-1} = 0. \tag{7.12}$$

Again, this condition together with (7.11) gives a necessary condition for a relative degree superior to four. We must however note that the expression of $L_g L_f^3 h(x)$ explicitly depends on the first assumption (7.11) and would be much more complex too handle if we used only the necessary and sufficient condition for a relative degree superior to three: $A_{12}A_{2n} = \cdots = A_{1,n-1}A_{n-1,n} = 0$. Proceeding further inductively, we have (for $k < d$)

$$\begin{aligned} L_g L_f^k h(x) = \; & b_n(\sigma'(x_1)\ldots\sigma'(x_{k-1}))\sigma'(x_n)(A_{12}A_{23}\ldots A_{k-2,k-1}) \\ & \times (A_{k-1,k}A_{kn}\sigma'(x_k) + \cdots + A_{k-1,n-1}A_{n-1,n}\sigma'(x_n)), \end{aligned}$$

if we require

$$A_{k,n} = 0 \text{ and } A_{k-1,k+1} = A_{k-1,k+2} = \cdots = A_{k-1,n-1} = 0.$$

This condition (for $k = 1, \ldots, d-1$) together with $A_{1n} = 0$ is equivalent to the first condition in the statement of the theorem. The second condition comes from the fact that we want the relative degree to be exactly d. This completes the proof. □

This result was based on the assumption that $b = (0, \ldots, 0, b_n)$, but there is another version of this result that holds without this restriction [67]. It is also possible to give similar conditions for other types of neural networks such as

$$\begin{cases} \dot{x} = \Lambda x + \sigma(Ax)u, \\ y = c^T x; \end{cases} \quad \text{or} \quad \begin{cases} \dot{x} = \Lambda x + \sigma(Ax + bu), \\ y = c^T x; \end{cases}$$

see [67].

It is important to understand the exact relevance of this result. Checking the relative degree of a system is fairly elementary and we can do it by examining the equations of the system. We perform this check by counting how many times we need to integrate the input before influencing the output (for example, the system $\dot{x}_1 = \sigma(x_2), \dot{x}_2 = \sigma(x_1 + x_2 + u), y = x_1$ has a relative degree of two since u does not influence y directly but must be integrated first by x_2). But here we have conditions that allow us to enforce a given relative degree a priori. Indeed, a network (7.9) with, for example, a matrix A containing no zeros will always have a relative degree of at most two; so if we try to model

a system using this type of network, we will never be able to reproduce its relative degree if our learning algorithm does not estimate the necessary zeros exactly. But the most important consideration is that this particular choice of constraints makes the evaluation of the Lie derivatives tractable so that we can actually compute the control $v(t)$ of the inverse system in Equation 7.8. The possibility of such an evaluation is the interesting contribution of this result.

7.5 Canonical form of SISO systems

As we have studied changes of coordinates in neural networks rather extensively in the previous chapters, we wish to discuss in some details an important change of coordinates for SISO systems. Namely, the functions $L_f^k h(x)$ that we have repeatedly used in the previous section play another interesting role: They provide part of a local coordinate transformation that locally brings a control-affine system into some canonical form that we define precisely in a moment. First, recall that, if the system has relative degree d at x_0, then

$$\begin{aligned} y &= h(x), \\ \dot{y} &= L_f h(x), \\ &\vdots \\ \frac{d^{(d-1)}y}{dt^{(d-1)}} &= L_f^{d-1} h(x). \end{aligned}$$

We first have to prove the following lemma that shows the linear independence of the gradients of the first r Lie derivatives $L_f^k h(x)$ at a point x_0 where the control-affine system has relative degree r.

Lemma 7.6. *If the control-affine system (7.6) has relative degree d at $x = x_0$, then the codistribution*

$$\mathcal{H}(x) = \operatorname{span}\left\{\nabla h(x); \nabla\left(L_f h(x)\right); \ldots; \nabla\left(L_f^{d-1} h(x)\right)\right\}$$

has dimension d at $x = x_0$.

With the help of this lemma, we will show that, if we can add to the first d Lie derivatives another $n - d$ functions such that the gradients of these n functions is an invertible (rank n) matrix at x_0, then we have a local change of coordinates at x_0.

Proof. We first show that the following (somewhat complex) formula holds. If φ is a real-valued function defined in a neighborhood $\mathcal{U}$ of x_0 and s, k, d are arbitrary positive integers, we have

$$L_{\mathrm{ad}_f^{k+d} g} L_f^s \varphi(x) = \sum_{i=0}^{d} (-1)^i \binom{d}{i} L_f^{d-i} L_{\mathrm{ad}_f^k g} L_f^{s+i} \varphi(x), \tag{7.13}$$

where $\left\{ \begin{array}{c} d \\ i \end{array} \right\}$ denotes the number of combinations of i objects out of d. Note that the above expression is a real-valued function since it is the scalar product of a row vector of real-valued functions with a column vector of real-valued functions. This formula holds by induction on d since

$$\begin{aligned} L_{\mathrm{ad}_f^{k+d+1}g} L_f^s \varphi(x) &= L_{[f,\mathrm{ad}_f^{k+d}g]}(L_f^s \varphi(x)) \\ &= L_f \left(L_{\mathrm{ad}_f^{k+d}g} L_f^s \varphi(x) \right) - L_{\mathrm{ad}_f^{k+d}g} L_f^{s+1} \varphi(x) \end{aligned}$$

because of the properties of the Lie derivatives presented in Section 7.1. We use Formula 7.13 to prove that the condition that

$$L_g \varphi(x) = L_g L_f \varphi(x) = \cdots = L_g L_f^k \varphi(x) = 0, \text{ for all } x \in \mathcal{U},$$

is equivalent to the condition that

$$L_g \varphi(x) = L_{\mathrm{ad}_f g} \varphi(x) = \cdots = L_{\mathrm{ad}_f^k g} \varphi(x) = 0, \text{ for all } x \in \mathcal{U}.$$

Now, because of the definition of the relative degree (Def. 7.12) and using Equation 7.13, we have that, for all integers i, j for which $i + j \leq d - 2$,

$$L_{\mathrm{ad}_f^i g} L_f^j h(x) = 0, \text{ for all } x \in \mathcal{U}, \tag{7.14}$$

and, for all integers i, j for which $i + j = d - 1$, we have

$$L_{\mathrm{ad}_f^i g} L_f^j h(x_0) \neq 0. \tag{7.15}$$

If we take the matrix of the gradients of the Lie derivatives of h and multiply it by the matrix of iterated brackets of f with g, we obtain a matrix of rank d when we evaluate it at x_0:

$$\begin{aligned} &\left[\nabla h(x_0); \nabla L_f h(x_0); \ldots; \nabla L_f^{d-1} h(x_0) \right] \left[g(x_0), \mathrm{ad}_f g(x_0), \ldots, \mathrm{ad}_f^{d-1} g(x_0) \right] \\ &\qquad = \begin{bmatrix} L_g h(x_0) & \cdots & L_{\mathrm{ad}_f^{d-1} g} h(x_0) \\ \vdots & & \vdots \\ L_g L_f^{d-1} h(x_0) & \cdots & L_{\mathrm{ad}_f^{d-1} g} L_f^{d-1} h(x_0) \end{bmatrix}. \end{aligned}$$

Indeed, this matrix has zeros in the upper antidiagonal block (because of (7.14)) while none of its antidiagonal elements are zero (because of (7.15))—this triangular structure guarantees a rank of d. This structure of the matrix implies that the matrix

$$\left[\nabla h(x_0); \nabla L_f h(x_0); \ldots; \nabla L_f^{d-1} h(x_0) \right] \tag{7.16}$$

has also rank d and, therefore, the codistribution $\mathcal{H}$ has dimension d at x_0. This proves Lemma 7.6. □

We now use these functions as the d first components of a change of coordinates and use Frobenius's theorem to find the $n-d$ others.

Lemma 7.7. *Consider a system (7.6) with finite relative degree d at x_0, which is always less than or equal to n. Then there exists a change of coordinates*

$$\Phi(x) = [\varphi_1(x), \dots, \varphi_n(x)]^T$$

for which

$$\begin{aligned} \varphi_1(x) &= h(x), \\ \varphi_2(x) &= L_f h(x), \\ &\vdots \\ \varphi_d(x) &= L_f^{d-1} h(x), \end{aligned}$$

and the Jacobian matrix $\nabla\Phi(x)$ is invertible at $x = x_0$. Furthermore, we can choose $\varphi_{d+1}(x), \dots, \varphi_n(x)$ so that

$$L_g \varphi_i(x) = 0, \text{ for all } i \text{ for which } d+1 \le i \le n, \text{ for all } x \in \mathcal{U}. \tag{7.17}$$

Proof. We identify first functions that satisfy $L_g\varphi_i(x) = 0$ and could serve as the last $n-d$ components of our change of coordinates. The vector $g(x_0)$ must be different from zero, otherwise the relative degree is infinite. Consider the distribution $\mathcal{G}(x) = \text{span}\{g(x)\}$, it is nonsingular and involutive since it is one-dimensional and nonzero. Because of Frobenius's theorem (see the alternative formulation of the theorem in Isidori's book [64]), there exist $n-1$ covector fields $\omega_1(x), \dots, \omega_{n-1}(x)$ such that

$$\omega_1(x)g(x) = \cdots = \omega_{n-1}(x)g(x) = 0. \tag{7.18}$$

These covector fields determine the surface orthogonal to the integral surface passing through x_0 and form a codistribution of dimension $n-1$.

These covectors are such that, together with the covectors $\nabla L_f^k h(x)$, they form a codistribution of dimension n at x_0:

$$\dim\left(\text{span}\left\{\omega_1(x_0), \dots, \omega_{n-1}(x_0), \nabla h(x_0), \dots, \nabla L_f^{d-1} h(x_0)\right\}\right) = n. \tag{7.19}$$

We prove by contradiction that this codistribution has dimension n. Suppose it is not the case, then all the column vectors of (7.16) belong to the span of $\omega_1(x_0), \dots, \omega_{n-1}(x_0)$ as the covectors ω_i form a codistribution of dimension $n-1$. Thus, because of (7.18), $\nabla h(x_0)g(x_0) = \nabla L_f h(x_0)g(x_0) = \nabla L_f^{d-1} h(x_0)g(x_0) = 0$. However, $\nabla L_f^{d-1} h(x_0)g(x_0) = L_g L_f^{d-1} h(x_0)$, by the definition of the Lie bracket. But we have supposed that $L_g L_f^{d-1} h(x_0)$ is different from zero, which leads to a contradiction.

It is possible [64] to write the covectors ω_i as the gradient of some vector functions; thus, there exists real-valued functions $\psi_1(x), \ldots, \psi_{n-1}(x)$ such that $\omega_1(x) = \nabla\psi_1(x), \ldots, \omega_{n-1}(x) = \nabla\psi_{n-1}(x)$. Because the distribution

$$\mathcal{H}(x) = \text{span}\left\{\nabla h(x), \nabla L_f h(x), \ldots, \nabla L_f^{d-1} h(x)\right\}$$

has dimension d at $x = x_0$ by Lemma 7.6 and the distribution

$$\mathcal{K}(x) = \text{span}\left\{\psi_1(x), \ldots, \psi_{n-1}(x), \nabla h(x), \ldots, \nabla L_f^{d-1} h(x)\right\}$$

has dimension n at $x = x_0$ (because of Equation 7.19 and the relationship between distributions and codistributions), it must always be possible to find $\{\varphi_{d+1}, \ldots, \varphi_n\} \subset \{\psi_1, \ldots, \psi_{n-1}\}$ such that the matrix $[\varphi_1(x_0), \ldots, \varphi_n(x_0)]$ has rank n and is therefore invertible. Furthermore, $L_g\varphi_{d+1}(x) = \cdots = L_g\varphi_n(x) = 0$ for all $x \in \mathcal{U}$ because $L_g\varphi_j(x) = (\nabla\varphi_j(x))g(x) = \omega_i(x)g(x)$, for some $i < n-1$ depending on j, and $\omega_i(x)g(x) = 0$, for $i = 1, \ldots, n-1$. This argument completes the lemma. □

Now, with the help of the previous change of coordinates, we can show the main result on canonical forms.

Theorem 7.8 (Isidori). *Consider a system (7.6) with finite relative degree d at x_0, which is always less than or equal to n. There exists a local change of coordinates*

$$z = \Phi(x) = [\varphi_1(x), \ldots, \varphi_n(x)]^T$$

such that the system becomes, in the new coordinates,

$$\left\{\begin{array}{rcl} \dot{z}_1 &=& z_2, \\ &\vdots& \\ \dot{z}_{d-1} &=& z_d, \\ \dot{z}_d &=& a(z) + b(z)u, \\ \dot{z}_{d+1} &=& q_{d+1}(z), \\ &\vdots& \\ \dot{z}_n &=& q_n(z); \\ y &=& z_1. \end{array}\right. \tag{7.20}$$

Proof. The proof relies on using the properties of the change of coordinates defined in the previous lemma. We have

$$\begin{array}{rcl} \dot{z}_1 &=& \nabla\varphi_1(x)\frac{\partial x}{\partial t} = \nabla h(x)(f(x) + g(x)u) \\ &=& L_f h(x) + L_g h(x)u = L_f h(x) = \varphi_2(x) = z_2, \end{array}$$

if $L_g h(x) \neq 0$ for all $x \in \mathcal{U}$—that is, the relative degree d is superior or equal to two. Repeating this computation for z_2 to z_{d-1}, we have

$$\begin{aligned} \dot{z}_2 &= (\nabla L_f h(x))(f(x)+g(x)u) = L_f^2 h(x) + 0 = \varphi_3(x) = z_3, \\ &\vdots \\ \dot{z}_{d-1} &= L_f^{d-1} h(x) + 0 = \varphi_d(x) = z_d. \end{aligned}$$

For z_d, we have now

$$\begin{aligned} \dot{z}_d &= (\nabla L_f^{d-1} h(x))(f(x)+g(x)u) \\ &= L_f^d h(x) + L_g L_f^{d-1} h(x) u \\ &= L_f^d h(\Phi^{-1}(z)) + L_g L_f^{d-1} h(\Phi^{-1}(z)) u, \end{aligned}$$

since $z = \Phi(x)$ and $\Phi(x)$ is invertible in a neighborhood of x_0. Letting $a(z) = L_f^d h(\Phi^{-1}(z))$ and $b(z) = L_g L_f^{d-1} h(\Phi^{-1}(z))u$, we have

$$\dot{z}_d = a(z) + b(z)u.$$

Furthermore, taking into account Equation 7.17, we have

$$\begin{aligned} \dot{z}_{d+1} &= \nabla\varphi_{d+1}(x)(f(x)+g(x)u) = L_f\varphi_{d+1}(x) + 0 = q_{d+1}(z), \\ &\vdots \\ \dot{z}_n &= \nabla\varphi_n(x)(f(x)+g(x)u) = L_f\varphi_n(x) + 0 = q_n(z); \end{aligned}$$

if we set $q_i(z) = L_f\varphi_i(\Phi^{-1}(z))$ for all $i = d+1, \ldots, n$. We thus have all the equations of the canonical form and we have proved Theorem 7.8. □

7.6 The zero dynamics

We consider again the canonical form (7.20) of a control-affine system (7.6). As the relative degree was an extension of the concept of the relative degree of a linear system, we consider here an extension of the concept of the zeros of a linear system, which is the zero dynamics.

7.6.1 Computation of the zero dynamics and of local inverses of nonlinear systems

We shall look for inputs that force the output to be zero. To be more precise, for a system initialized at an equilibrium corresponding to a zero output, we look for all possible inputs $u(t)$ for which the output $y(t)$ remains zero in a neighborhood of $t = 0$. Suppose the system has a relative degree of $d \leq n$ in a neighborhood of $x = x_0$. Suppose that $f(x_0) = 0$ (x_0 is an equilibrium of the system as long as the input is zero) and that $h(x_0) = 0$ (the output is

zero). Because of these assumptions, $z_1(0) = \cdots = z_d(0) = 0$. Moreover, the requirement that $L_g\varphi_i(x) = 0$ for the last $n-d$ components of the change of coordinates allows us to translate these functions ($L_g(\varphi_i(x) + c_i) = 0$ if c_i is a constant). We can also choose the last $n-d$ components such that, going through the chain of integrators, $z_{d+1}(0) = \cdots = z_n(0) = 0$. Furthermore, since x_0 is an equilibrium of the original system, the origin must be an equilibrium point of the canonical form. This invariance of the equilibrium implies that

$$\begin{aligned} a(0) &= 0, \\ q_{d+1}(0) &= 0, \\ &\vdots \\ q_n(0) &= 0. \end{aligned}$$

Now, we want to find all the inputs for which the output $y(t) = z_1(t)$ stays identically zero for all positive times $t < T$. This identity implies $z_1(t) = \cdots = z_d(t) = 0$ since $z_2 = \dot{z}_1 = \dot{y} = 0, \ldots, z_d = \dot{z}_{d-1} = d^{d-1}y/dt^{d-1} = 0$. Because $\dot{z}_d = a(z) + b(z)u$, we now have that the input $u(t)$ must satisfy

$$\begin{aligned} 0 = \quad & a(0, \ldots, 0, z_{d+1}(t), \ldots, z_n(t)) \\ & + b(0, \ldots, 0, z_{d+1}(t), \ldots, z_n(t))u(t), \end{aligned} \tag{7.21}$$

for all positive times $t < T$. Because we force the dynamics of $z_1, \ldots, z_d$ to zero, the dynamics of $z_{d+1}, \ldots, z_n$ is given by

$$\begin{aligned} \dot{z}_{d+1} &= q_{d+1}(0, \ldots, 0, z_{d+1}, \ldots, z_n), \\ &\vdots \\ \dot{z}_n &= q_n(0, \ldots, 0, z_{d+1}, \ldots, z_n). \end{aligned} \tag{7.22}$$

Expressing $u(t)$ explicitly in Equation 7.21, we get

$$u(t) = -\frac{a(0, \ldots, 0, z_{d+1}(t), \ldots, z_n(t))}{b(0, \ldots, 0, z_{d+1}(t), \ldots, z_n(t))}, \tag{7.23}$$

where $z_{d+1}, \ldots, z_n$ are given by the previous set of equations (7.22) where the initial condition $z_{d+1}(0), \ldots, z_n(0)$ is arbitrary. We can choose the initial condition for the $n-d$ last variables arbitrarily because the choice of $u(t)$ guarantees $z_{d+1}(t) = 0$, so $z_d(t) = \cdots = z_1(t) = y(t) = 0$. For a given choice of the initial condition, the input $u(t)$ is unique; it will exist in a neighborhood of $t = 0$ because there must exist a neighborhood $\mathcal{U}$ of the origin for which $b(z) \neq 0$ for all $z \in \mathcal{U}$, otherwise the system cannot have relative degree d at x_0. We summarize the results of this section by a definition.

Definition 7.13. The *zero dynamics* of the control-affine system (7.6) with relative degree d at x_0 having the canonical form (7.20) is the system (7.22).

Note that the zero dynamics is independent from a change of coordinates on the control-affine system, which means that it is a geometric invariant of the system. The same remark holds for the relative degree. We can use the zero dynamics to find a local inverse for a control-affine system. We repeat the procedure we used to determine the zero dynamics, but this time $y(t)$ is not identically zero. So, we have, for the first d variables,

$$\begin{aligned} z_1(t) &= y(t), \\ &\vdots \\ z_d(t) &= \frac{d^{d-1}y(t)}{dt^d}. \end{aligned}$$

But we know that

$$d^d y(t)/dt^d = a(z(t)) + b(z(t))u(t),$$

which we rewrite as

$$u(t) = \frac{d^d y(t)/dt^d - a(y(t), \ldots, d^{d-1}y(t)/dt^{d-1}, z_{d+1}(t), \ldots, z_n(t))}{b(y(t), \ldots, d^{d-1}y(t)/dt^{d-1}, z_{d+1}(t), \ldots, z_n(t))}. \quad (7.24)$$

For the $n-d$ last variables, we have $\dot{z}_i = q_i(z)$ for $i = d+1, \ldots, n$; which we write as

$$\begin{aligned} \dot{z}_{d+1} &= q_{d+1}(y(t), \ldots, d^{d-1}y(t)/dt^{d-1}, z_{d+1}(t), \ldots, z_n(t)), \\ &\vdots \\ \dot{z}_n &= q_n(y(t), \ldots, d^{d-1}y(t)/dt^{d-1}, z_{d+1}(t), \ldots, z_n(t)). \end{aligned} \quad (7.25)$$

Using the solution of (7.25) for an arbitrary initial condition $z_{d+1}(0), \ldots, z_n(0)$ together with Equation 7.24 produces the input $u(t)$ that tracks the desired output $y(t)$ in some neighborhood of the origin.

7.6.2 Canonical dynamical neural networks

The local equivalence between a control-affine system and its canonical form, and the difficulty to structurally guarantee a given relative degree for the Hopfield-type neural network (7.9) led Delgado [30, 29] to propose a modification of the neural network (7.9) that makes its analysis much easier. This new architecture uses the principle of dynamic extension, which means that by adding a number of integrators between the input and the output of the system,

we can control its relative degree. This extended or canonical Hopfield-type network is of the form

$$\left\{\begin{array}{rcl} \dot{x}_1 &=& x_2, \\ &\vdots& \\ \dot{x}_{d-1} &=& x_d, \\ \dot{x}_d &=& -x_d + \sum_{j=1}^n A_{dj}\sigma(x_j) + b_d u, \\ \dot{x}_{d+1} &=& -x_{d+1} + \sum_{j=1}^n A_{d+1,j}\sigma(x_j), \\ &\vdots& \\ \dot{x}_n &=& -x_n + \sum_{j=1}^n A_{n,j}\sigma(x_j); \\ y &=& x_1, \end{array}\right. \tag{7.26}$$

which means that it is already in canonical form. We obtain the inverse trivially by using the zero dynamics of the system as presented above. Suppose we are given a reference output $y(t)$ and want to find the input $v(t)$ for which the neural network (7.26) will produce the desired reference signal at its output. The input is given by the system

$$\left\{\begin{array}{rcl} \dot{z}_{d+1} &=& -z_{d+1} + \sum_{j=1}^n A_{dj}\sigma(z_j), \\ &\vdots& \\ \dot{z}_n &=& -z_n + \sum_{j=1}^n A_{n,j}\sigma(z_j), \\ v(t) &=& \frac{\frac{d^d y(t)}{dt} - (-x_d + \sum_{j=1}^n A_{dj}\sigma(x_j))}{b_d}, \end{array}\right. \tag{7.27}$$

under the condition that

$$z_1(t) = y(t), \quad z_2(t) = dy(t)/dt, \quad \ldots, \quad z_d(t) = d^{(d-1)}y(t)/dt^{(d-1)}.$$

The main problem with the canonical dynamical neural network (7.26) and the Hopfield-type network satisfying the conditions of Proposition 7.5 is that, although we now are in a position to compute the relative degree and the inverse of these networks, it is not clear anymore what kind of control-affine systems they can approximate. While we know from Chapter 6 that dynamical perceptrons can approximate any control system arbitrarily well and that Hopfield-type networks can approximate any dynamical system arbitrarily well, we loose this guarantee by requiring a special structure for these systems. It will thus be necessary to show what the approximation capabilities of these networks are before we can use them as a substitute for the Hopfield-type architectures. This problem is made somewhat more difficult by the fact that the particular structure guarantees a specific relative degree; we therefore have to show that we can use a given structure of the network to approximate any control-affine system—having the corresponding relative degree—arbitrarily well. There is no such result available at this point. Another problem is that these authors

do not provide any method to estimate the relative degree of a system from input–output data; this shortcoming implies that we must provide this piece of information a priori, which is somewhat contradictory to the philosophy of a data-driven approach.

7.7 Feedback linearization

We cannot cover the topic of nonlinear control without briefly mentioning the procedure of feedback linearization. It consists in trying to find a state-feedback law under which a control-affine system becomes a linear system. We need to assume that we have access to the state of the system, be it directly or via an observer. For the control-affine (7.6) with input $u(t)$ and output $y(t)$, we determine its input through a reference input $v(t)$ and a static feedback of the state of the system:

$$u(t) = p(x(t)) + q(x(t))v(t).$$

The resulting closed-loop system is a system with input $v(t)$ and output $y(t)$, which remains control affine,

$$\begin{aligned} \dot{x} &= f(x) + g(x)p(x) + g(x)q(x)v, \\ y &= h(x). \end{aligned}$$

The problem is to find conditions under which it becomes possible to choose $p(x)$ and $q(x)$ such that the closed-loop system becomes linear. Suppose that the system has relative degree n. The change of coordinates necessary to produce the canonical form is then entirely specified by the repeated Lie derivatives of h with respect to f:

$$\Phi(x) = \Big[h(x), L_f h(x), \ldots, L_f^{n-1} h(x)\Big]^T.$$

The canonical form has a simple expression,

$$\left\{ \begin{array}{rcl} \dot{z}_1 &=& z_{2,} \\ & \vdots & \\ \dot{z}_{d-1} &=& z_{d,} \\ \dot{z}_d &=& a(z) + b(z)u, \\ y &=& z_1. \end{array} \right.$$

This form implies that, if we choose as state feedback law

$$u(t) = \frac{-a(z)}{b(z)} + \frac{1}{b(z)} v(t), \tag{7.28}$$

the canonical form becomes

$$\left\{\begin{array}{rcl} \dot{z}_1 &=& z_{2,} \\ & \vdots & \\ \dot{z}_{d-1} &=& z_{d,} \\ \dot{z}_d &=& v, \\ y &=& z_1, \end{array}\right.$$

which is a controllable linear system. Noting that; in the proof of Theorem 7.8, we set $a(z) = L_f^d h(\Phi^{-1}(z))$ and $b(z) = L_g L_f^{d-1} h(\Phi^{-1}(z))u$; and that here $d = n$; these developments imply that if we apply the state feedback law

$$u(t) = \frac{-L_f^n h(x)}{L_g L_f^{n-1} h(x)} + \frac{v(t)}{L_g L_f^{n-1} h(x)}$$

to a control-affine system of relative degree n in a neighborhood of x_0, the closed loop system is linear and controllable. Once we have made the system linear, we can use linear control methods to make sure that the system meets the desired performance requirements.

This procedure relies on the fact that the relative degree of the system is equal to n. If the system has a relative degree d strictly less than n at the point x_0, the previous result does not hold. However, some weaker results are available. The first one relies on "replacing" the output function $h(x)$ by a new output function $m(x)$ that gives the system a relative degree of n. The resulting system will have a linear dynamics but a nonlinear output function.

Proposition 7.9 (Isidori). *Consider a control-affine system (7.6) of relative degree $d < n$ at x_0. There exists an output function $m(x)$ such that the system*

$$\begin{array}{rcl} \dot{x} &=& f(x) + g(x)u \\ y &=& m(x) \end{array}$$

has relative degree n at x_0 (the system is therefore feedback linearizable at x_0) if and only if

1. *The matrix $\left[g(x_0), \mathrm{ad}_f g(x_0), \ldots, \mathrm{ad}_f^{n-1} g(x_0)\right]$ has rank n.*

2. *The distribution $\mathcal{G}(x) = \mathrm{span}\left\{g(x), \mathrm{ad}_f g(x), \ldots, \mathrm{ad}_f^{n-2} g(x)\right\}$ is involutive in some neighborhood $\mathcal{U}$ of x_0.*

The change of coordinates becomes $\Phi(x) = \left[h(x), L_f h(x), \ldots, L_f^{n-1} h(x)\right]^T$ and the system under state feedback takes the form

$$\begin{cases} \dot{z}_1 &= z_{2,} \\ & \vdots \\ \dot{z}_{d-1} &= z_{d,} \\ \dot{z}_d &= v, \\ y &= h(\Phi^{-1}(z)). \end{cases}$$

We see that the dynamics of the closed-loop system is linear and controllable, although the output becomes a nonlinear function of the state. Another possible treatment of the case where the relative degree of the system is lower than n is to renounce to making the dynamics of the system completely linear. However, the input–output behavior of the system can be made linear by using the proposed state feedback (7.28) regardless of the fact that the relative degree is less than n. The resulting canonical form is

$$\begin{cases} \dot{z}_1 &= z_{2,} \\ & \vdots \\ \dot{z}_{d-1} &= z_{d,} \\ \dot{z}_d &= v, \\ \dot{z}_{d+1} &= q_{d+1}(z), \\ & \vdots \\ \dot{z}_n &= q_n(z), \\ y &= z_1. \end{cases}$$

We note here that the input–output dynamics is independent from the state variables $z_{d+1}, \ldots, z_n$ while the dynamics of the state variables $z_1, \ldots, z_d$ is linear and controllable. An approach similar to this one was applied to the input–output linearization of Hopfield-type networks by Delgado [29]. However, the treatment was limited to low-dimensional neural networks because of the difficulties involved in the computation of the iterated Lie derivatives $L_f^k h(x)$.

Except for the work of Delgado, the work of Nikolaou and Hanagandi [112] (who used a neural network to identify a stirred tank reactor of relative degree one and then input–output linearized the neural network model), and the work of Liu and Chen [79] (who used neural networks only as function approximators) not much seems to have been done on the feedback linearization of neural networks. The main obstacle appears again to be the difficulty to compute the necessary iterated Lie derivatives. The special structure proposed by Kambhapati to find the inverse of a Hopfield network makes the necessary computations possible, but they also make the direct computation of the inverse possible.

7.8 Conclusion

We have reviewed the theory of differential-geometric control and its applications to neural networks. We first introduced two important mathematical objects: the Lie derivative of a real-valued function with respect to a vector field and the Lie bracket of two vector fields. We then linked the controllability of a system to the span of the repeated Lie brackets of the control vectors. The fundamental result was Frobenius's theorem, which analyzes the geometry of the surfaces reachable from a given point. In terms of controllability, this result led to Hermann's controllability rank condition [56]. Conditions for observability were obtained dually and led to the observability rank condition. The application of these criteria to neural networks turned out to be unsuccessful because of computational difficulties.

We then turned to the study of inverses of neural networks. We first introduced the notion of the relative degree of a control-affine system. Using a result of Hirschorn [60], it was possible to obtain conditions under which a Hopfield-type network achieves a given relative degree and possesses an inverse [67]. As we had paid a particular attention to changes of coordinates in Chapters 4 and 6, we also studied a change of coordinates that brings a nonlinear system locally into a canonical form. This study also led to the notion of the zero dynamics of a nonlinear system and we saw that we could design a particular neural network architecture based on this canonical form.

Our observation is that the results available on the application of the methods from differential-geometric control to neural networks are still fragmentary and often hold only under stringent restrictions. The main obstacle was the difficulty to compute the iterated Lie derivatives and analytically solve the resulting conditions. To overcome this obstacle, it was necessary to put strong restrictions on the structure of the neural network. Unfortunately, these restrictions made that we lost the guarantee that the neural network model is able to approximate any control-affine system arbitrarily well, which usually justifies the use of neural networks in the first place. Although this property may still hold, we have at this point no proof of it.

In contrast, the methods of Chapter 5—which gave practical conditions for observability, controllability, and identifiability of neural networks—appeared more successful. Although the idea of applying the techniques from nonlinear control theory to neural networks appeared most attractive at first, significant progress is still necessary before it can become a mature direction of research.

Chapter 8

Integration of ordinary differential equations by composition methods

All the results presented so far dealt with analytical properties of neural networks. In the coming two chapters, we move towards applications and consider numerical questions relating to the use of Lie-algebraic methods for the integration of ODEs and the modeling of nonlinear systems. In this chapter, we develop the tools necessary to build the new modeling techniques that we present in the next chapter.

8.1 Exponential maps and the Baker–Campbell–Hausdorff formula

A notational difficulty appears here. In the coming two chapters, we shall write the ODE as $\dot{x} = A(x)$ and not as $\dot{x} = f(x)$ (as we did in the previous chapters) to follow the conventions of Lie algebra theory. This notation is somewhat unnatural, but it has the positive effect of emphasizing the likeness between linear and nonlinear systems (although this similarity can sometimes be misleading). The main example of this likeness arises when we write a solution to the ODE. So far, we have written a solution as a flow (as is standard in the dynamical system literature [63])

$$x(t) = \Phi(x_0, t).$$

Now, we also write it as what we call an exponential map, which is the standard notation in the Lie algebra literature [113],

$$x(t) = e^{tA}(x_0).$$

This notation means nothing more than: *"$x(t)$ is the image after time t of the initial condition x_0 under the flow of $\dot{x} = A(x)$"*. It is an extension of the case where the vector field is linear and the exponential of the matrix is the solution; hence the name of exponential map. But we have to stress that the exponential e^{tA} is—in general—a nonlinear map from the state-space into itself.

The notation is also coherent with the basic properties of the exponential. For example (using the dot "$\cdot$" to denote the composition of maps), we have

$$e^{(t+s)\,A} = e^{sA} \cdot e^{tA}, \tag{8.1}$$

which is nothing but a restatement of the group property of the flow: $\Phi(x_0, t+s) = \Phi(\Phi(x_0, t), s)$ [63]. This property states that, for any initial condition, evolving according to the differential equation $\dot{x} = A(x)$ for $t+s$ units of time is equivalent to: First, evolving for t units of time; then starting from the state where the system had arrived, evolving for another s units of time. This fact has an important consequence. Namely, by setting $s = -t$ in (8.1), we have $e^{-tA} \cdot e^{tA} = I$. Thus, e^{tA} is an invertible map. More precisely, because of the smoothness of the vector field, e^{tA} is a diffeomorphism (i.e., a smooth map with a smooth inverse) from the state-space onto some subset of itself [63].

We made the assumption that an ODE was the appropriate model for the dynamics underlying the system. However, when we integrate an ODE numerically, we look for the value of the solution at discrete time intervals Δt only. In the last chapters, these discrete time intervals will correspond to the instants at which we can measure the state or the output of the system. Thus, our continuous-time system turns into a discrete-time one:

$$x(k+1) = F_{\mathrm{c}}(x(k)).$$

This discrete-time system cannot, however, be completely arbitrary as it arises from the sampling of an ODE (hence the notation with a subscript c to emphasize that this relation arises from a continuous-time system). From the definition of the exponential and from the fact that the recurrence relation F_{c} is the transformation arising from the sampling of the ODE at Δt, we have

$$F_{\mathrm{c}}(.) = e^{\Delta t A}(.).$$

Therefore, approximating the recurrence relation F_{c} is equivalent to finding an approximation for $e^{\Delta t A}$. Also, since the exponential map is invertible, this equivalence proves that the recurrence relation is invertible.

The Lie-algebraic tools that we have developed in the previous chapter will now prove useful to analyze these exponential maps. Not only are these

methods an essential part of nonlinear system theory [64] as we have seen in the previous chapter, but they also have a prominent position in the area of classical mechanics [11] and partial differential equations [113]. We refer to the textbook of Arnold [11] for a more detailed presentation of Lie algebras and their applications to dynamical system theory. We only need here some basic concepts, which we present in this section, to develop our argument.

Just as in Section 7.1, we consider here the vector space of all smooth vector fields as a Lie algebra. In the case where we define the ODE on $\mathbb{R}^n$, the expression for the bracket was

$$[A, B] = B^T \nabla A - A^T \nabla B \,,$$

where ∇A and ∇B are the Jacobians of A and B. Recall also that the Lie bracket of two vector fields is again a vector field. In the more general case of an ODE defined on some manifold, we can define the following vector field as the Lie bracket of the vector fields A and B:

$$[A, B] = \left.\frac{\partial^2}{\partial s \partial t}\right|_{t=s=0} e^{-sB} \cdot e^{-tA} \cdot e^{sB} \cdot e^{tA} \,.$$

The bracket $[A, B]$ is called the commutator of the vector fields, as it measures the degree of noncommutativity of the flows of the vector fields ($[A, B] = 0 \Leftrightarrow e^{tA} e^{tB} = e^{tB} e^{tA}$).

The last mathematical tool we need is the Baker–Campbell–Hausdorff formula (BCH). This formula is the central piece of the rest of our presentation. It gives an expansion for the product of two exponentials of elements of the Lie algebra [69]:

$$e^{\Delta t A} e^{\Delta t B} = e^{\left(\begin{array}{l} \Delta t (A + B) + (\Delta t^2/2)[A, B] \\ +(\Delta t^3/12)([A, [A, B]] + [B, [B, A]]) \\ +(\Delta t^4/24)[A, [B, [B, A]]] + \cdots \end{array} \right)} . \tag{8.2}$$

It should be interpreted as follows. Letting the flow of $\dot{x} = B(x)$ act on the initial condition of the system for Δt, and then—from where we have arrived—letting the flow of $\dot{x} = A(x)$ act for another Δt *is equivalent* to letting the flow of

$$\dot{x} = ((A + B) + \frac{\Delta t}{2}[A, B] + \frac{\Delta t^2}{12}([A, [A, B]] + [B, [B, A]]) + \cdots)(x)$$

act on the initial condition for Δt.

8.2 Composition methods

We first look at how to solve ODEs using compositions. In the next chapter, we shall see how, using the techniques we develop here, the prediction of an

unknown system comes down to finding an ODE whose solution best matches the observed data. Suppose we want to solve an ODE with vector field X,

$$\dot{x}(t) = X(x(t)), \tag{8.3}$$

for a time step of Δt. The problem then becomes that of building an approximation for $e^{\Delta t X}$ as we have that

$$x(\Delta t) = e^{\Delta t X}(x_0)\,.$$

This problem has recently been the focus of much attention in the field of numerical analysis, especially for the integration of Hamiltonian differential equations [69, 85]. The *guiding principle* behind our work is that, *if you can split the vector field* X into elementary parts for which you can solve the differential equation directly, then *you can compose these solutions to approximate the solution* of the more complex system.

Suppose that the vector field X is of the following form:

$$X = A + B,$$

where you can integrate A and B analytically or much more easily than X. Then we can use the BCH formula to produce a first-order approximation to the exponential map:

$$\textbf{BCH:} \qquad e^{\Delta t X} = e^{\Delta t A} e^{\Delta t B} + o(\Delta t^2). \tag{8.4}$$

We check this relation by multiplying the left-hand sides of Equation 8.2 on the left by $e^{-\Delta t X}\,(= e^{-\Delta t (A+B)})$, expanding the expression using the BCH formula itself (8.2) and simplifying it using the properties of the Lie bracket. This computation gives

$$\begin{aligned}
& e^{-\Delta t X}\left(e^{\Delta t A} e^{\Delta t B}\right) \\
&\quad = e^{-\Delta t (A+B)}\left(e^{\Delta t (A+B) + \frac{\Delta t^2}{2}[A,B] + \frac{\Delta t^3}{12}([A,[A,B]]+[B,[B,A]]) + o(\Delta t^4)}\right) \\
&\quad = e^{\left(\begin{array}{l} \Delta t.(0) + \frac{\Delta t^2}{2}[A,B] + \frac{\Delta t^3}{12}([A,[A,B]]+[B,[B,A]]) + o(\Delta t^4) \\ \quad + \frac{\Delta t^2}{2}\left[-(A+B),(A+B) + \frac{\Delta t}{2}[A,B] + \frac{\Delta t^2}{12}([A,[A,B]]+[B,[B,A]]) + o(\Delta t^3)\right] \\ \qquad + o(\Delta t^4) \end{array}\right)} \\
&\quad = e^{\left(\begin{array}{l} \frac{\Delta t^2}{2}[A,B] + \frac{\Delta t^3}{12}([A,[A,B]]+[B,[B,A]]) + o(\Delta t^4) \\ \quad - \frac{\Delta t^3}{4}([A,[A,B]]-[B,[B,A]]) + o(\Delta t^4) \end{array}\right)} \\
&\quad = e^{\frac{\Delta t^2}{2}[A,B] - \frac{\Delta t^3}{6}[A,[A,B]] + \frac{\Delta t^3}{3}[B,[B,A]] + o(\Delta t^4)} \\
&\quad = e^{o(\Delta t^2)}.
\end{aligned} \tag{8.5}$$

Since the right-hand side is equal to $I + o(\Delta t^2)$ by a Taylor series expansion, where I is the identity map, we get (8.4). The relation of first-order approximation (8.4) between the solution of A and B, and the solution of X is the

essence of the method since it shows that we can approximate an exponential map (that is the mapping arising from the solution of an ODE) by composing simpler maps (Fig. 8.1).

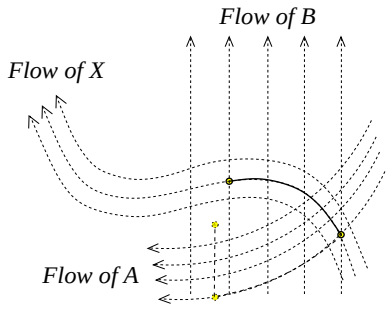


Figure 8.1: First-order approximation by composition. The vector field X of the system $\dot{x} = X(x)$ is the sum of two vector fields A and B. The Baker–Campbell–Hausdorff formula gives a first-order approximation with $e^{\Delta t X}(x_0) \approx e^{\Delta t B} e^{\Delta t A}(x_0)$. The bold solid line represents the evolution of the system between 0 and Δt from the initial condition x_0. The dashed lines represent the evolution from x_0 to $e^{\Delta t A}(x_0)$ and from $e^{\Delta t A}(x_0)$ to $e^{\Delta t B} e^{\Delta t A}(x_0)$.

We use the BCH formula to eliminate higher-order terms as we did for the first-order approximation, but this time for the composition of three terms. This way, we can show that the following symmetric-leapfrog scheme is second order:

$$\textbf{Leapfrog:} \qquad \begin{aligned} e^{\Delta t X} &= e^{\frac{\Delta t}{2}A} e^{\Delta t B} e^{\frac{\Delta t}{2}A} + o(\Delta t^3), \\ &= S(\Delta t) + o(\Delta t^3). \end{aligned} \tag{8.6}$$

Indeed,

$$\begin{aligned}
& e^{\frac{\Delta t}{2}A} e^{\Delta t B} e^{\frac{\Delta t}{2}A} \\
&= e^{\Delta t\left((\frac{A}{2}+B)+\frac{\Delta t}{2}[\frac{A}{2},B]+\frac{\Delta t^2}{12}([\frac{A}{2},[\frac{A}{2},B]]+[B,[B,\frac{A}{2}]])+o(\Delta t^3)\right)} e^{\frac{\Delta t}{2}A} \\
&= e^{\left(\begin{array}{l} \Delta t\left((A+B)+\frac{\Delta t}{4}[A,B]+\frac{\Delta t^2}{12}(\frac{1}{4}[A,[A,B]]+\frac{1}{2}[B,[B,A]])+o(\Delta t^3)\right) \\ \quad +\frac{\Delta t^2}{2}\left([(\frac{A}{2}+B)+\frac{\Delta t}{4}[A,B]+o(\Delta t^2),\frac{A}{2}]\right) \\ \quad\quad +\frac{\Delta t^3}{12}\left([(\frac{A}{2}+B)+o(\Delta t),[(\frac{A}{2}+B)+o(\Delta t),\frac{A}{2}]]+[\frac{A}{2},[\frac{A}{2},(\frac{A}{2}+B)+o(\Delta t)]]\right) \end{array}\right)} \\
&= e^{\left(\begin{array}{l} \Delta t(A+B)+\frac{\Delta t^2}{4}[A,B]+\frac{\Delta t^3}{48}[A,[A,B]]+\frac{\Delta t^3}{24}[B,[B,A]]+o(\Delta t^4) \\ \quad -\frac{\Delta t^2}{4}[A,B]-\frac{\Delta t^3}{16}[A,[A,B]]+o(\Delta t^4) \\ \quad\quad +\frac{\Delta t^3}{24}[B,[B,A]]+o(\Delta t^4) \end{array}\right)} \\
&= e^{\Delta t(A+B)+\Delta t^2.(0)-\frac{\Delta t^3}{24}[A,[A,B]]+\frac{\Delta t^3}{12}[B,[B,A]]+o(\Delta t^4)}.
\end{aligned}$$

Completing the argument by multiplying the left- and right-hand sides by $e^{-\Delta t X}$ as for the first-order scheme, we obtain that the leapfrog scheme is

second-order (8.6). Using this leapfrog scheme $S(\Delta t)$ as a basis element, we can build a fourth-order scheme:

$$\begin{aligned} \textbf{Fourth-order:} \qquad e^{\Delta t X} &= S(c\Delta t)S(d\Delta t)S(c\Delta t) + o(\Delta t^5), \\ &= SS(\Delta t) + o(\Delta t^5), \end{aligned}$$

with $c = -2^{1/3}/(2-2^{1/3})$ and $d = 1/(2-2^{1/3})$. Repeating the leapfrog strategy, Yoshida [149] showed that it is possible to produce an approximation to $e^{\Delta t X}$ up to any order:

$$\begin{aligned} &\textbf{Arbitrary order :} \qquad \exists k, \exists w_1, v_1, \ldots, w_k, v_k : \\ &e^{\Delta t X} = e^{w_1 \Delta t A} e^{v_1 \Delta t B} \ldots e^{w_k \Delta t A} e^{v_k \Delta t B} + o(\Delta t^{p+1}). \end{aligned} \tag{8.7}$$

There exist other composition schemes than the repeated leapfrog [85] as we will see in Section 8.4 and some of them are more efficient. Forest and Ruth [37] also showed that we can build approximations for more than two vector fields.

Example

We briefly explicit the essence of the technique by looking at the Van der Pol oscillator. This system is described by the following second-order nonlinear differential equation:

$$\ddot{x} - \alpha(1 - x^2)\dot{x} + x = 0. \tag{8.8}$$

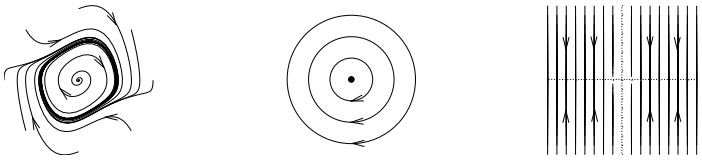
We can write it as a first-order system in state-space form and see that the vector field is a linear combination of two vector fields A, B which we can solve analytically:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} y \\ -x \end{pmatrix} + \alpha \begin{pmatrix} 0 \\ (1 - x^2)y \end{pmatrix} = A + \alpha B. \tag{8.9}$$

We present the flows of the vector fields X, A and B in the following figure (Fig. 8.2). The composition of the solutions of A and B in a first-order integrator was presented in Figure 8.1.

8.3 Control of the ball and beam

We show now that we can apply the composition method to nontrivial problems. In [90], we have studied how we could use a composition method to integrate the equations describing a controller for the ball and beam. We do not describe the theoretical background for this method here, but we briefly present its implementation. We illustrate the point that finding appropriate splittings for a composition method is not an unrealistic task.



flow of $X = A + \alpha.B$ flow of A flow of B

Figure 8.2: Flow of the Van der Pol oscillator and of its splittings. The flow of the oscillator possesses a stable periodic orbit. The vector field X of the oscillator is the superposition of two elementary parts A and B. The linear part A is a pure rotation (a linear system with an eigenvalue with zero real part). The nonlinear part B has trajectories that are all parallel to the y axis, as $\dot{x} = 0$. Because of this degeneracy, we can solve it explicitly.

The equations of the ball and beam are the following:

$$\left\{ \begin{array}{rcl} 0 & = & \ddot{r} + G\sin\theta + \beta\dot{r} - r\dot{\theta}^2, \\ \tau & = & (r^2+1)\ddot{\theta} + 2r\dot{r}\dot{\theta} + Gr\cos\theta, \end{array} \right.$$

where r is the position of the ball, θ is the angle of the beam, τ is the torque applied to the beam (which is here the control variable), G is the gravity ($G = 9.81m/s^2$), and β is the viscous friction constant ($\beta = 0.1s^{-1}$ in the simulations). Applying the feedback transformation (using a linear controller on θ and $\dot{\theta}$ with parameters k_1 and k_2)

$$\tau = 2r\dot{r}\dot{\theta} + Gr\cos\theta + k_1\theta + k_2\dot{\theta} + (r^2+1)u$$

and defining $z_1 = r, z_2 = \dot{r}, \xi_1 = \theta$, $\xi_2 = \dot{\theta}$, we obtain the following state-space equations for the ball and beam together with its linear controller:

$$\begin{array}{rcl} \dot{z}_1 & = & z_2, \\ \dot{z}_2 & = & -\beta z_2 - G\sin\xi_1 + z_1\xi_2^2, \\ \dot{\xi}_1 & = & \xi_2, \\ \dot{\xi}_2 & = & -k_1\xi_1 - k_2\xi_2 + u. \end{array}$$

The system as described so far is not asymptotically stable. It is characterized by the fact that the ball tends to first roll far away from the equilibrium point and then come back slowly. We can significantly improve the performance of the controller by using a method from constructive Lyapunov control called forwarding [128]. At every instant where we take a control action (typically at $t = k\Delta t$, if we have a digital control system), we have to compute the following control law [90]

$$u = -\frac{\partial\Psi}{\partial\xi_2} - \xi_2,$$

where we obtain the term $\frac{\partial \Psi}{\partial \xi_2}$ by integrating the following set of differential equations for an interval of time that is long compared to the time constant of the system ($T = 10\,s$):

$$\begin{array}{rcll}
\frac{d}{ds}\tilde{z}_1 & = & \tilde{z}_2, & \tilde{z}_1(0) = z_1, \\
\frac{d}{ds}\tilde{z}_2 & = & -\beta\tilde{z}_2 - G\sin\tilde{\xi}_1 + \tilde{z}_1\tilde{\xi}_2^2, & \tilde{z}_2(0) = z_2, \\
\frac{d}{ds}\nu_1 & = & \nu_2, & \nu_1(0) = 0, \\
\frac{d}{ds}\nu_2 & = & -\beta\nu_2 - G\cos\tilde{\xi}_1 e^{As}_{(12)} + \nu_1\tilde{\xi}_2^2 + 2\tilde{z}_1\tilde{\xi}_2 e^{As}_{(22)}, & \nu_2(0) = 0, \\
\frac{d}{ds}\frac{\partial\Psi}{\partial\xi_2} & = & (\beta\nu_1 + 2\nu_2)C_1 + (\beta\tilde{z}_1 + 2\tilde{z}_2)C_2, & \frac{\partial\Psi}{\partial\xi_2}(0) = 0,
\end{array}$$

where $A = \begin{bmatrix} 0 & 1 \\ -k_1 & -k_2 \end{bmatrix}$, $\tilde{\xi}(s) = e^{A\tau}\xi$, $C_1(\tilde{z}_1, \tilde{\xi}_1, \tilde{\xi}_2) = -G\sin\tilde{\xi}_1 + \tilde{z}_1\tilde{\xi}_2^2$, and $C_2(\tilde{z}_1, \tilde{\nu}_1, \tilde{\xi}_1, \tilde{\xi}_2) = -G\cos\tilde{\xi}_1 e^{A\tau}_{(12)} + \nu_1\tilde{\xi}_2^2 + 2\tilde{z}_1\tilde{\xi}_2 e^{A\tau}_{(22)}$. For the computer simulations, we have placed both eigenvalues of A at -2 with $k_1 = 4, k_2 = 4$. Based on the decay rate associated to these eigenvalues, we have set $T = 10$ seconds. We look at the closed-loop response of the system starting at the initial condition (1,0,-1.57,0); this position corresponds to setting the beam upright ($\theta = -\pi/2$) with the ball at 1 meter of the pivot of the ball and beam.

To integrate the variational equations by composition, we need to

- Find a splitting of the equations
- Choose an integrator
- Choose a time step Δt

We choose the following splitting:

$$\begin{pmatrix} \frac{d}{ds}\tilde{z}_1 \\ \frac{d}{ds}\tilde{z}_2 \\ \frac{d}{ds}\nu_1 \\ \frac{d}{ds}\nu_2 \\ \frac{d}{ds}\frac{\partial\Psi}{\partial\xi_2} \end{pmatrix} = \begin{pmatrix} \tilde{z}_2 \\ -\beta\tilde{z}_2 \\ \nu_2 \\ -\beta\nu_2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ C_1(\tilde{z}_1, \tilde{\xi}_1, \tilde{\xi}_2) \\ 0 \\ C_2(\tilde{z}_1, \tilde{\nu}_1, \tilde{\xi}_1, \tilde{\xi}_2) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ (\beta\nu_1 + 2\nu_2)C_1 \\ +(\beta\tilde{z}_1 + 2\tilde{z}_2)C_2 \end{pmatrix}.$$

First, we split among the evolution of $\frac{\partial\Psi}{\partial\xi_2}$(it is independent of itself), the linear part, and the nonlinear part. Condensing the notation, we can write $\frac{d}{ds}\zeta(s) = B\zeta(s) + C(\zeta(s)) + D(\zeta(s))$. Recall that the evolution of $\tilde{\xi}$ is given by exponentiation of the matrix A. So, the solution for the linear part is found by exponentiation of the matrix $e^{sB}(\zeta_0) = e^{sB}\cdot\zeta_0$. The solution for the nonlinear parts are trivial, since C_1 and C_2 depend on variables that all remain constant: $e^{sC}(\zeta_0) = \zeta_0 + sC$ and $e^{sD}(\zeta_0) = \zeta_0 + sD$. Second, we choose the following second-order leapfrog integrator:

$$\zeta(\Delta t) \approx e^{\Delta t/2B}e^{\Delta t/2C}e^{\Delta tD}e^{\Delta t/2C}e^{\Delta t/2B}(\zeta_0).$$

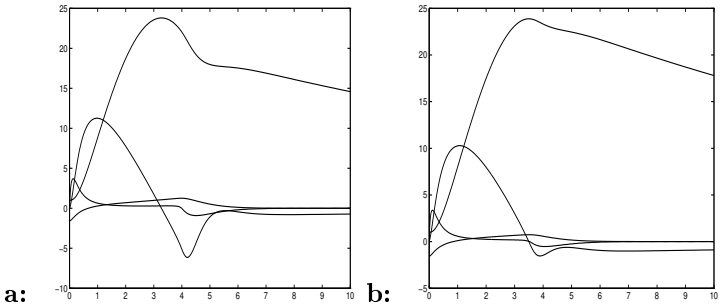


Figure 8.3: Evolution of the ball and beam. **a:** Using a Runge–Kutta method. **b:** Using the leapfrog composition integrator.

Third, we choose an integration step $\Delta t = 0.2$. The integration using the second-order leapfrog integrator gives the results presented in Figure 8.3). The composition integrator is able to control the ball and beam just as when we use a Runge–Kutta integrator. The performance of the composition method is slightly worse, but the speed of the integration is multiplied by a factor of 4 (while part of the Runge–Kutta code is compiled, as opposed to the composition method which is implemented as an interpreted routine in Matlab). By reducing the step size of the composition method, we recover the same performance as the Runge–Kutta method, but the advantage in speed is then lost.

Thus, we see that we are able to find appropriate splittings for some non-trivial problems, and that the composition method is efficient. Splittings will not always exist, but there are many interesting problems where composition methods are a viable alternative to standard integration schemes.

8.4 Formulas for composition integrators

We present a short-list of different composition integrators (Table 8.1) for the case where the vector field can be split into two parts: $X = A + B$. We can then find composition integrators for the case of more than two vector fields by using these schemes recursively. For example, in the case $X = A + B + C$, we can write $X = A+D$, where $D = B+C$. We then choose a composition method for D, which we use as basis step for the integration of X. The error constant associated to each method is the effective error constant defined in [85]. This measure of error accounts for the fact that, for a given order, methods with more substeps may be more precise, but they also require more computational work. This tradeoff would permit methods composed of fewer substeps to use a smaller step size, thereby increasing their precision. For a given order, even

Order	Substeps	Error	Formula
2	3	0.070	$S(\Delta t) = e^{(\Delta t/2)A} e^{\Delta t B} e^{(\Delta t/2)A}$
2	5	0.026	$e^{a_1 \Delta t A} e^{b_1 \Delta t B} e^{a_2 \Delta t A} e^{b_1 \Delta t B} e^{a_1 \Delta t A}$ $a_1 = 0.1932$, $b_1 = 0.5$, $a_2 = 1 - 2a_1$
4	7	0.098	$S(w_1 \Delta t) S(w_2 \Delta t) S(w_1 \Delta t)$ $w_1 = 1.3512$, $w_2 = 1 - 2w_1$
4	11	0.033	$S(w_1 \Delta t) S(w_2 \Delta t) S(w_3 \Delta t) S(w_2 \Delta t) S(w_1 \Delta t)$ $w_1 = 0.28$, $w_2 = 0.6254$, $w_3 = 1 - 2w_1 - 2w_2$
4	11	0.0046	$e^{a_1 \Delta t A} e^{b_1 \Delta t B} e^{a_2 \Delta t A} e^{b_2 \Delta t B} e^{a_3 \Delta t A} e^{b_3 \Delta t B}$ $\times e^{a_3 \Delta t A} e^{b_2 \Delta t B} e^{a_2 \Delta t A} e^{b_1 \Delta t B} e^{a_1 \Delta t A}$ $a_1 = 0.0893$, $b_1 = 0.4$, $a_2 = -0.0973$, $b_2 = -0.1$, $a_3 = 1 - 2a_1 - 2a_2$, $b_3 = 1 - 2b_1 - 2b_2$

Table 8.1: Formulas for the composition methods.

if the effective error constant decreases as the number of substeps increases, it may not be advantageous to use the method with the most substeps for finite time steps. This choice will depend on the system we are integrating, and on the error we require.

We present two types of methods: the symmetric methods and the symmetric methods composed of symmetric steps. The symmetric methods are presented directly as a product of exponentials, while the symmetric methods composed of symmetric steps are presented as a product of symmetric-leapfrog steps $S(\Delta t)$. The leapfrog is the simplest method to implement, while the fourth-order symmetric method (error=0.0046) is the most efficient. We therefore recommend the leapfrog for preliminary experiments and the fourth-order symmetric method (error=0.0046) for final implementations. For the formulas for higher-order integrators, see [85]. A remark of fundamental importance is that these formulas are valid even without the analytical solutions of $e^{\Delta t A}$ and $e^{\Delta t B}$. Indeed, we can replace these solutions in the following symmetric methods by any first-order integrator (e.g., $I + \Delta t A = e^{\Delta t A} + o(\Delta t^2)$) and still guarantee the same order of approximation. Similarly, in the symmetric methods composed of symmetric steps, we can replace the leapfrog $S(\Delta t)$ by any second-order symmetric integrator.

8.5 Convergence of decompositions of exponential operators

At the end of Section 8.2, we noted that we could find composition integrators of arbitrarily high order, but that these integrators converge only for small enough Δt. In contrast, Suzuki has shown [138] that it is possible to build schemes that converge for any Δt. We define the first-order integrator for $\dot{x} = X(x) = A_1(x) + A_2(x) + \cdots + A_m(x)$ as

$$Q_1(\Delta t) = e^{\Delta t A_1} e^{\Delta t A_2} \ldots e^{\Delta t A_m} = e^{\Delta t X} + o(\Delta t^2)$$

and the second-order leapfrog as

$$\begin{aligned} Q_2(\Delta t) = S(\Delta t) &= e^{(\Delta t/2) A_1} \ldots e^{(\Delta t/2) A_{m-1}} e^{\Delta t A_m} e^{(\Delta t/2) A_{m-1}} \ldots e^{(\Delta t/2) A_1} \\ &= e^{\Delta t X} + o(\Delta t^3). \end{aligned}$$

It is then possible to build integrators of increasingly high order by proceeding similarly to the approach of Yoshida. We define recursively

$$\begin{aligned} Q_{2n}(\Delta t) &= Q_{2n-2}(p_{n,1}\Delta t) Q_{2n-2}(p_{n,2}\Delta t) \ldots Q_{2n-2}(p_{n,r}\Delta t) \\ &= e^{\Delta t X} + o(\Delta t^{2n+1}), \end{aligned}$$

where the parameters satisfy the conditions

$$p_{n,1} + \cdots + p_{n,r} = 1 \text{ and } p_{n,1}^{2m-1} + \cdots + p_{n,r}^{2m-1} = 0.$$

Suzuki showed that such a decomposition converges to the solution of the ODE for *any* Δt. We have that

$$\lim_{n \to \infty} ||Q_{2n}(\Delta t) - e^{\Delta t X}|| = 0, \quad \forall \Delta t,$$

where the norm is the L_2 norm. As an illustration, if we consider the case $r = 3$, a solution is the repeated leapfrog:

$$\begin{aligned} S_2(\Delta t) &= S(\Delta t), \\ S_{2n}(\Delta t) &= S_{2n-2}(p_n \Delta t) S_{2n-2}((1 - 2p_n)\Delta t) S_{2n-2}(p_n \Delta t), \end{aligned}$$

with $p_n = 1/(2 - 2^{1/3})$; which is thus guaranteed to converge to the solution of the ODE for any Δt. The result of Suzuki is, in fact, much more general. As presented above, the higher-order approximations are derived recursively from the second-order leapfrog $S(\Delta t)$. We can however use any other symmetric second-order scheme, for example, the symmetric composition of the linearizations of the vector fields $Q_2(t) = (1 + \frac{\Delta t}{2} A_1) \ldots (1 + \frac{\Delta t}{2} A_{m-1})(1 + \Delta t A_m)(1 + \frac{\Delta t}{2} A_{m-1}) \ldots (1 + \frac{\Delta t}{2}) A_1$. Moreover, we can make the conditions on the parameters $p_{n,r}$ more general. Also, this result applies to any kind of operator on a Banach space, which means that the composition methods are applicable not only to ODEs, but to delay differential equations and partial differential equations. Behind the mathematics, we see that composition is an extremely versatile tool for the approximation of solutions of ODEs.

8.6 Simulation of arrays of Chua's circuits

We now tackle a problem of larger scale with the simulation of large arrays of nonlinear circuits. To show the potential relevance of composition methods to the field of neural networks, we present their application to the integration of the differential equations describing cellular neural networks (CNN) made out of an array of Chua's circuits. CNNs are effective for parallel signal processing and real-time simulation of phenomena described by partial differential equations. They have therefore been used for image processing and for the study of nonlinear spatiotemporal behaviors [105]. Their main interest from the applied viewpoint is that we can integrate a large array of such circuits into a monolithic VLSI chip.

We apply composition methods first to a single Chua's circuit and then to an array of coupled Chua's circuits. Of course, we can apply the standard schemes, such as the Euler method or the Runge–Kutta method to the integration of arrays of Chua's circuits. We therefore compare the performance of composition methods with those of standard methods: fixed time step explicit Euler and adaptive fourth-order Runge–Kutta. We shall see that composition methods not only provide a new viewpoint on the dynamics of arrays of Chua's circuits, but that they also provide enhanced performance for their simulation.

8.6.1 Splitting

Chua's circuit is a nonlinear electrical circuit. We can write it as a state-space model or ODE of the form

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{bmatrix} -a & a & 0 \\ b & -b & 1 \\ 0 & -c & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} g(x) \\ 0 \\ 0 \end{pmatrix},$$

with $g(x)$ a piecewise-linear function: $g(x) = -(a/2)[(s_1+s_2)x+(s_0-s_1)(|x-\beta_1|-|\beta_1|)+(s_2-s_0)(|x-\beta_2|-|\beta_2|)]$ (see Fig. 8.6.1). This system can exhibit chaotic behavior for appropriate parameter values—for example, $a = 6.3$, $b = 0.7$, $c = 7$, $s_0 = -1.14286$, $s_1 = s_2 = -0.71429$, $\beta_1 = -1$, and $\beta_2 = 1$.

If we split the vector field into its linear part and its nonlinear part, we see that we can solve both parts explicitly. If we call A the matrix containing the parameters a, b, c; the solution for the linear part is $x(\Delta t) = e^{\Delta t A} \cdot x_0$, where the exponential is the classical matrix exponential. The solution for the nonlinear part is $x(\Delta t) = e^{tg}(x_0)$. In the simple case where $\beta_1 < 0 < \beta_2$, we

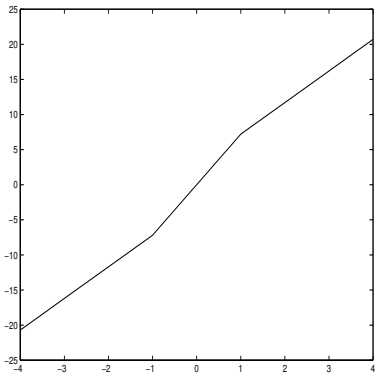


Figure 8.4: The piecewise-linear nonlinearity of Chua's circuit. It has three linear segments determined by two breakpoints. The exact expression of the nonlinearity and its parameters are in the text.

can compute the positive part of the solution as follows:

$$\tau(x_0) = \begin{cases} \frac{1}{s_2}\ln(\frac{x_0-\delta}{\beta_2-\delta}), & x > \beta_2, \\ \frac{1}{s_0}\ln(\frac{x_0}{\beta_2}), & 0 < x < \beta_2; \end{cases}$$

$$x(\Delta t) = \begin{cases} e^{s_2(\tau(x_0)+\Delta t)}(\beta_2 - \delta), & \tau(x_0) + \Delta t > 0, \\ e^{s_0(\tau(x_0)+\Delta t)}\beta_2, & \tau(x_0) + \Delta t \leq 0; \end{cases}$$

where $\delta = \beta_2(1 - \frac{s_0}{s_2})$. As $x = 0$ is a fixed point, trajectories starting at positive initial conditions always remain positive. For negative initial conditions, we have the same formula except that β_1 takes the place of β_2, s_1 takes the place of s_2, and all signs are reversed. Therefore, we can integrate the equations of Chua's circuit with a composition method.

Moreover, we can consider general CNNs. For example, we can have a CNN composed by an array of Chua's circuits coupled together by some constant linear coupling. If we have n circuits having each three state variables, the equations describing the evolution of the system would be

$$\begin{pmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \end{pmatrix} = \begin{bmatrix} -a & a & 0 \\ b & -b & 1 \\ 0 & -c & 0 \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \begin{pmatrix} g(x_i) \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} \rho\Sigma_{j=1}^{n}\xi_{ij}x_j \\ 0 \\ 0 \end{pmatrix}.$$

with $i = 1, \ldots, n$ and where the coupling term ξ_{ij} is equal to one if circuit i is connected to circuit j, and is equal to zero else. What we can remark is that, however high the number of circuits is, the system is made of a large linear part (the first and third terms) and a nonlinear part that consists of

n identical decoupled piecewise-linear equations. This splitting means that once again, we can solve both the linear and the nonlinear part explicitly, and combine them to find the solution of this large system of ODEs. If we call $x_{\rm tot}$ the total state of the system: $x_{\rm tot} = (x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n)^T$, A the matrix representing the linear part of the dynamics of the circuits and the linear interconnections, and Γ the nonlinear part of the dynamics; we have the following ODE:

$$\dot{x}_{\rm tot} = Ax_{\rm tot} + \Gamma(x_{\rm tot}).$$

As a first toy problem, we present a simulation of the solution of such a CNN for the case of a two-by-two array of Chua's circuits with the following topology of interconnection (Fig. 8.5) and interconnection strength $\rho = 0.1$. We realize

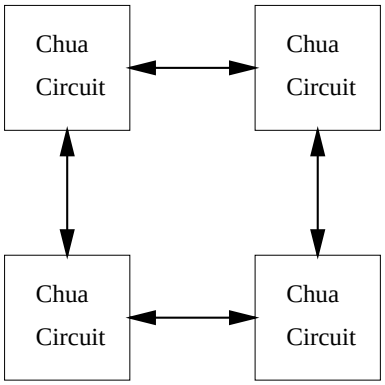


Figure 8.5: Cellular neural network. A 2 × 2 array of Chua's circuits.

the simulation in the Matlab environment. We use a second-order integrator: $x_{\rm tot}(\Delta t) = e^{(\Delta t/2)A} e^{\Delta t \Gamma} e^{(\Delta t/2)A}(x_{\rm tot})$. The time step for the integration is $\Delta t = 0.1$. In Figure 8.6, we see the evolution of the state of the four circuits.

An important point to consider is how to solve efficiently the one-dimensional nonlinear ODEs arising through the splitting of the vector field of the array of Chua's circuits. An analytical solution might not be difficult to compute for this piecewise linear ODE, but its implementation might be rather inefficient. Therefore, we present a method for computing the solution of a one-dimensional ODE using a lookup table.

8.6.2 Integration of one-dimensional ODEs

The main observation is that the behavior of a one-dimensional ODE is monotone between two fixed points. Let us write the ODE as $\dot{x} = A(x)$ with $x \in \mathbb{R}$, and its solution as $x(t) = \Phi(x_0, t)$. Suppose first that the ODE has no fixed

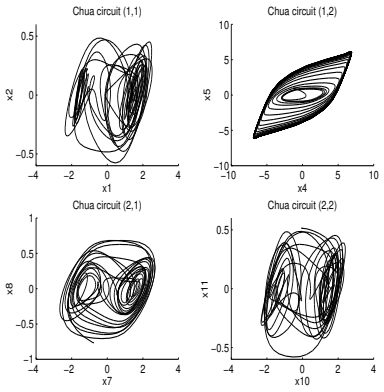


Figure 8.6: Evolution of the state of the CNN array. Simulation by a leapfrog composition method. We plot the evolution of the x and y variables of each of the four circuits.

points. Then, the trajectory passing through $x_0 = 0$ will cover the whole of $\mathbb{R}$. Define this trajectory as the function $\psi(t) = \Phi(0, t)$. Because of the group property of the flow $(\Phi(\Phi(x_0, t), s) = \Phi(x_0, t + s))$, we have that $\Phi(x_0, t) = \Phi(\Phi(0, \psi^{-1}(x_0)), t) = \Phi(0, \psi^{-1}(x_0) + t)$. So, $\Phi(x_0, t) = \psi(\psi^{-1}(x_0) + t)$. Thus, by storing the solution ψ of $\dot{x} = A(x)$ with $x(0) = 0$, and its inverse in a lookup table, we are able to compute the solution of the one-dimensional ODE in just two table lookups.

Suppose now that the ODE has k fixed points: $z_1, \ldots, z_k$. Set also $z_0 = -\infty$ and $z_{k+1} = \infty$. Define $v_0 = z_1 - 1, v_1 = (z_1 + z_2)/2, \ldots, v_k = (z_{k-1} + z_k)/2, v_{k+1} = z_k + 1$. Define $\psi_i(t) = \Phi(v_i, t)$, then we know that this trajectory $\psi_i(t)$ will cover the whole of the interval $]z_i, z_{i+1}[$. Thus, for all $x_0 \in]z_i, z_{i+1}[$: $\Phi(x_0, t) = \psi_i(\psi_i^{-1}(x_0) + t)$ for all i $= 0, \ldots,$ k. We can therefore store the ψ_i^{-1} in a lookup table and the ψ_i in another lookup table consisting of k subtables (one per interval). This procedure gives us a powerful way to compute repeated integrations of a one-dimensional ODE.

8.6.3 Numerical results

To show that the composition method is an efficient integration method, we run a simulation of a spiral wave on an array of 50×50 Chua's circuits as shown in [116] and [105]. The appropriate set of parameters is

$$\begin{array}{lll} a = 10, & b = 1, & c = 0.3014987, \\ s_1 = 0.078573, & s_2 = 55, & s_0 = -1.25719, \\ \beta_1 = -1, & \beta_2 = 0.023744. & \end{array}$$

These parameters correspond to a bistable cell with two stable equilibria: $P^- = [-1.238, 1.238]$, $P^+ = [0.02385, -0.02385]$. The appropriate initial condition is described as follows. All the x and y variables are in the rest state except for a front of excited cells. The z variable has a circular gradient of excitation, which is smoothly decreasing. This initial condition is described in details in [116].

We compare the composition method to a Runge–Kutta adaptive time step method and a simple fixed-step Euler method. We choose a fourth-order composition method with different time steps ($\Delta t = 0.01, \Delta t = 0.02, \Delta t = 0.05$), a first-order Euler method with different time steps ($\Delta t = 0.001, \Delta t = 0.002, \Delta t = 0.005$), and a fourth-order Runge–Kutta method (the time step is chosen automatically and adaptively). In a previous experiment [105], a fixed-step Euler method with time step $\Delta t = 0.0005$ had been used.

It is possible to increase the time step of the Euler method to $\Delta t = 0.002$ while keeping sufficient accuracy. We found that the method was then about twice as fast as the adaptive Runge–Kutta method. We show in Figure 8.7 snapshots of the activity of the x variable on a grid of 50×50 Chua's circuits at times $t_0 = 0, t_1 = 20, t_2 = 40, t_3 = 60, t_4 = 80, t_5 = 100$ for the Euler method. We should not forget that there are two other variables y and z for each Chua's circuit that also evolve over time.

If we try to increase the time step of the Euler method, we see that instabilities appear for $\Delta t = 0.005$, which make the integration unreliable as we can see in Figure 8.9 where we show snapshots of the activity of the x variable at $t = 10$.

In contrast, we can use a much larger time step for the composition method because of its good conservation properties. This larger time step comes at the price of a higher number of operations per iteration since the method contains a number of substeps. But as a whole, the fourth-order composition method with a time step of $\Delta t = 0.01$ is already as fast as the Euler method with a time step of $\Delta t = 0.001$ for the same accuracy. At $\Delta t = 0.02$ for the composition method, the integration still proceeds normally and the method is two times as fast as the Euler method. These simulations are displayed in Figure 8.8.

At $\Delta t = 0.05$, instabilities appear and they are too severe to make the integration acceptable. We can see this instability in Figure 8.10. Although, the two integration methods give results that are numerically not completely identical, all the phenomena that have been described for the spiral wave in an array of Chua's circuits are preserved. The differences are mainly the result of the truncation errors because we have tried to speed up both methods to a maximum. For smaller time steps, both methods again coincide.

8.7 Conclusion

In this chapter, we introduced composition methods for the integration of ODEs into the fields of neural networks and of control. Starting from the notion of the bracket of two vector fields, which we had defined in the previous chapter, we studied the solution of ODEs in terms of exponential maps instead of the usual notion of trajectories. Our main tool was the BCH formula that allowed us to compute products of exponential maps. This computation of products of exponentials allowed us in turn to approximate the solution of ODEs by splitting them into simpler parts and computing an appropriate composition of the solution of these parts.

To demonstrate the effectiveness of the composition methods, we first used them to implement a forwarding control law for the ball and beam. We found the necessary splitting of the vector field without difficulty and the resulting method outperformed Runge–Kutta methods. We then used composition methods to simulate arrays of Chua's circuits. Again, we found the splitting without difficulty as we isolated the nonlinear part from the linear part, which resulted in a linear system and a fully decoupled nonlinear system. The simulation of a 50×50 array of Chua's circuits demonstrated the better performance of the composition method when compared to Runge–Kutta methods.

The introduction of composition methods into the fields of neural networks and of control theory—together with the demonstration of their usefulness for the forwarding control of the ball and beam, and for the simulation of arrays of Chua's circuits—is an important contribution of this dissertation.

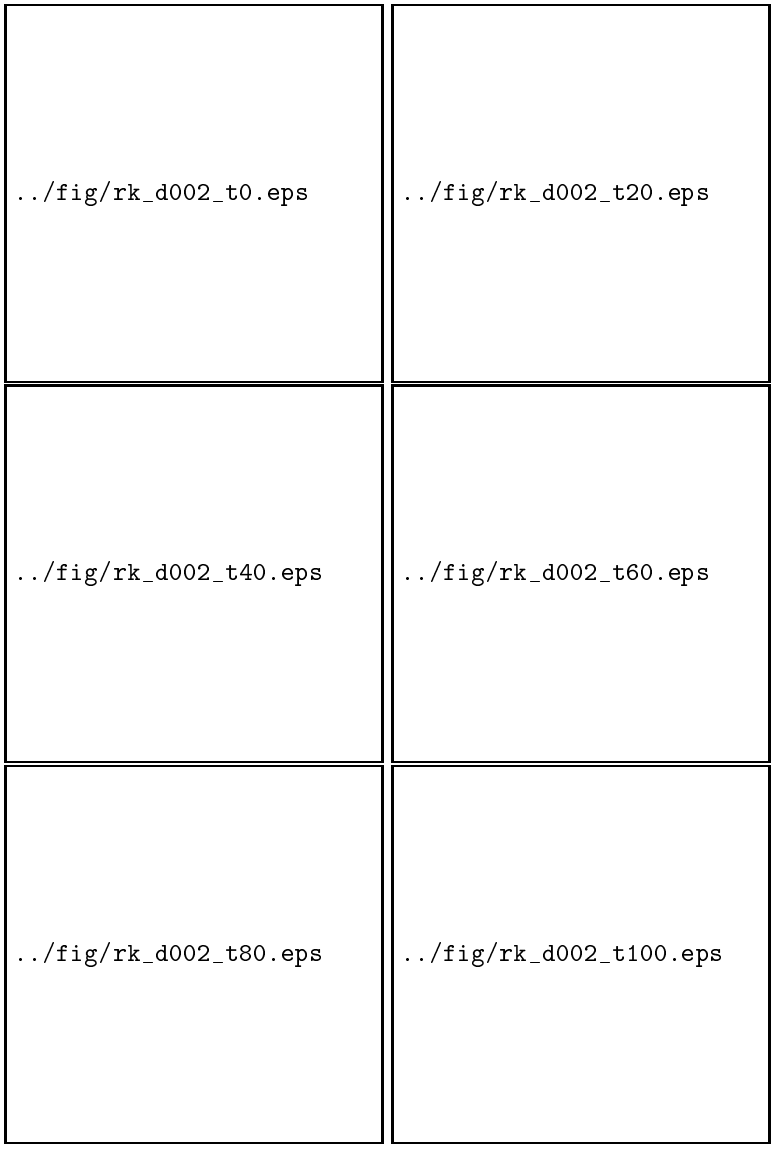


Figure 8.7: Snapshot of the spiral wave for the Euler method ($t_0 = 0, t_1 = 20, t_2 = 40, t_3 = 60, t_4 = 80, t_5 = 100$). The picture represents the activity of the x variable on a 50×50 grid of Chua's circuits. We do not present the activity of the y and z variables.

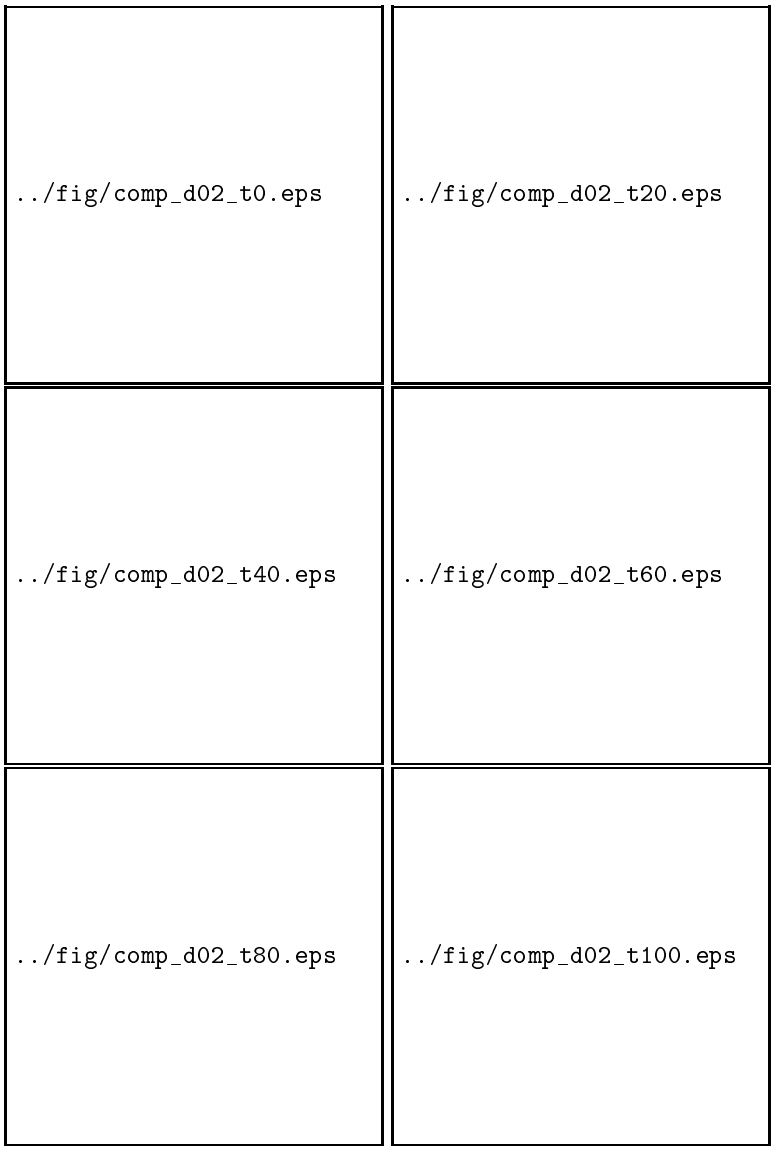


Figure 8.8: Snapshot of the spiral wave for the composition method ($t_0 = 0, t_1 = 20, t_2 = 40, t_3 = 60, t_4 = 80, t_5 = 100$). The composition method uses a time step much larger than the Euler method.



Figure 8.9: Snapshot of the integration for the Euler method with $\Delta t = 0.005$ at time $t = 10$. With this step size, the method becomes unstable and cannot give accurate results.

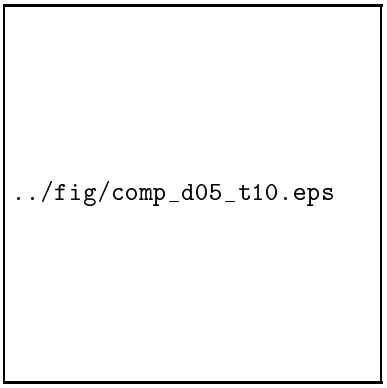


Figure 8.10: Snapshot of the integration for the composition method with $\Delta t = 0.05$ at time $t = 10$. With this time step, the composition method becomes unstable and does not give accurate results.

Chapter 9

Composition networks for system modeling

The composition methods for the integration of ODEs that we have presented in the previous chapter will now form the basis of a novel technique for the modeling of nonlinear systems: the *composition network*. The rationale behind the work we present now lies in the following questions: "What are the features of continuous-time systems that make their identification in discrete time different from the identification of arbitrary discrete-time systems? And can we use these properties to engineer better identification techniques for such systems"? It is the investigation of the logical consequences of these questions that eventually led us to the study of Lie-algebraic methods, which form the foundation of this chapter and of the previous two.

9.1 Compositions in system modeling

We consider continuous-time systems

$$\left\{ \begin{array}{rcl} \dot{x}(t) & = & f(x(t), u(t)), \\ y(t) & = & g(x(t), u(t)) \end{array} \right.$$

that we identify in discrete time (with a sampling period Δt). The data that is available for modeling the system is a discrete-time sequence of output samples $y(k) = g(x(k\Delta t), u(k\Delta t))$. We shall see that this sampling gives a particular structure to the resulting discrete-time system.

To study and use this structure easily, we have to impose restrictive conditions on our system. But we will relax these conditions and make our technique

general in Section 9.3. We consider first autonomous systems (no inputs) without noise:

$$\left\{ \begin{array}{rcl} \dot{x}(t) & = & f(x(t)), \\ y(k) & = & g(x(k\Delta t)), \end{array} \right.$$

with $t \in \mathbb{R}$, $k \in \mathbb{Z}$, and $\Delta t \in \mathbb{R}$ fixed a priori. The notation is unusual but describes sampled systems accurately. The aspect that is particular to sampled continuous-time systems, and which we use as a priori information to develop more accurate modeling techniques, is the following. If we rewrite the system completely in discrete-time,

$$\left\{ \begin{array}{rcl} x(k+1) & = & F_{\mathrm{c}}(x(k)), \\ y(k) & = & g(x(k)), \end{array} \right.$$

the recurrence relation F_{c} has to be an *invertible* transformation from the state-space into itself (we use the subscript c to emphasize that the discrete-time recurrence is derived from a continuous-time system). We proved this property in the Section 8.1.

In a first step, we also make the extra assumption that we can directly observe all the state variables. The system then reduces to

$$x(k+1) = F_{\mathrm{c}}(x(k)). \tag{9.1}$$

So the problem we consider first is the estimation, from discrete-time samples, of the (invertible) discrete-time recurrence F_{c} associated to a multidimensional ODE:

$$\text{From } x(1), x(2), \ldots, x(N), \text{ estimate } \hat{x}(k+1) = \hat{F}(x(k)).$$

The solution of this problem is as such of limited practical use, because these assumptions are so restrictive. But, once we have analyzed this problem in depth, we will extend our solution to more general systems (with noise, with an output map, with inputs).

Given that the recurrence relation F_{c} is invertible, it is interesting to have a predictor $\hat{F}$ that is invertible too. Thus, the question of predicting sampled continuous-time systems becomes one of how to approximate the class of invertible functions which arises from the sampling of ODEs. The basic idea is that the set of all invertible functions forms a group; and that the natural operation among elements of a group is the *composition* of these elements. While the operation of linearly combining basis functions (fixed or adaptive) serves as the foundation to most standard approximation schemes (e.g., polynomials, single–hidden-layer perceptrons, radial basis functions), this operation does not preserve the property of being invertible. In contrast, composition does preserve this property, so we propose to build an approximation scheme based on

composing basis functions instead of adding them. We will show that the advantages of networks of compositions over standard neural network techniques are that we can use smaller training sets, improve the accuracy of the modeling, and speed up the training of the model.

9.2 Prediction of dynamical systems

In Chapter 8, we showed that we could obtain an approximation to the exponential map of an arbitrary vector field using a sequence of compositions. To obtain a general approximation result, we mention the following fact: multiplying a vector field by a scalar is equivalent to multiplying the integration time. More precisely, the solution to

$$\dot{x} = aA(x) \quad \text{is} \quad x(\Delta t) = e^{(a\Delta t)A} x_0. \tag{9.2}$$

An immediate consequence of this fact and Suzuki's result of on the decomposition of exponential operators (Section 8.5) for the approximation of dynamical systems (assuming we consider the approximation on some compact set Ω) is that, if the vector field is a linear combination of some simpler vector fields, we can approximate its solution by a sequence of compositions (remember that the product denotes the composition of maps):

$$\begin{gathered} X = \textstyle\sum_{i=1}^{m} a_i A_i \quad \Rightarrow \quad \forall \epsilon \in \mathbb{R}, \quad \exists n, \exists w_{ij} \text{ such that} \\ \forall x \in \Omega : \left| e^{\Delta t X}(x) - \prod_{j=1}^{n} \prod_{i=1}^{m} e^{w_{ij} \Delta t A_i} \right| < \epsilon. \end{gathered} \tag{9.3}$$

We now use this approximation result to create a new approximation technique for sampled dynamical systems. The transformation and implementation of such an approximation scheme into a multilayer network is straightforward, as illustrated in the following figure (Fig. 9.1). We call this architecture a *composition network*.

If we suppose that the vector field of the observed system is some unknown linear combination of the vector fields A and B, which we can integrate directly, and if we are given a set of measurements $x(1), \ldots, x(N)$ of the state of the system, we can solve the problem of predicting a system by minimizing the prediction error

$$E = \sum_{k=1}^{N} \|\hat{x}(k) - x(k)\|^2$$

of the model by tuning the weights w_{ij}. When compared to other neural network methods, it shows the advantage of incorporating a priori knowledge about the system. Neural network methods have no obvious way to incorporate a priori knowledge in the regression model, making the composition network an attractive alternative.

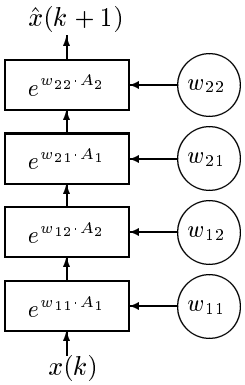


Figure 9.1: Implementation of the composition network. The network implements a nonlinear transformation that is a composition of invertible exponential maps. The weights w are free parameters that we adjust by optimization.

9.2.1 Computation of the gradients

The computation of the gradients (with respect to the weights) does not present any difficulty. We can therefore make use of any gradient-based optimization method to find the weights that minimize the prediction error. A general derivation of the gradient of composition networks is difficult since the choice of the architecture will depend on the problem at hand. We will therefore only present an illustrative example to show that the use of the chain rule allows us to compute all the necessary gradients.

Consider a generalized second-order composition network of the form

$$\hat{F}(x) = e^{w_3 A_\gamma} e^{w_2 B_\beta} e^{w_1 A_\alpha}(x).$$

We let the vector fields A and B depend on some extra nonlinear parameters α, β, γ. Unfortunately, the exponential notation does not offer the friendliest environment for the use of the chain rule. We have to reintroduce some more classical notation. Let

$$\Phi(x,t,p) = e^{tA_p}(x) \text{ and } \Psi(x,t,p) = e^{tB_p}(x).$$

The network then becomes

$$\hat{F}(x) = \Phi(\Psi(\Phi(x, w_1, \alpha), w_2, \beta), w_3, \gamma).$$

Hence, using the chain rule, we have

$$\frac{dF}{dw_3} = \frac{\partial \Phi}{\partial t}(\Psi(\Phi(x, w_1, \alpha), w_2, \beta), w_3, \gamma),$$

$$\frac{dF}{dw_2} = \frac{\partial \Phi}{\partial x}(\Psi(\Phi(x, w_1, \alpha), w_2, \beta), w_3, \gamma) \times \frac{\partial \Psi}{\partial t}(\Phi(x, w_1, \alpha), w_2, \beta),$$

$$\frac{dF}{dw_1} = \frac{\partial \Phi}{\partial x}(\Psi(\Phi(x, w_1, \alpha), w_2, \beta), w_3, \gamma) \times \frac{\partial \Psi}{\partial x}(\Phi(x, w_1, \alpha), w_2, \beta) \times \frac{\partial \Phi}{\partial t}(x, t, \alpha);$$

$$\frac{dF}{d\gamma} = \frac{\partial \Phi}{\partial p}(\Psi(\Phi(x, w_1, \alpha), w_2, \beta), w_3, \gamma),$$

$$\frac{dF}{d\beta} = \frac{\partial \Phi}{\partial x}(\Psi(\Phi(x, w_1, \alpha), w_2, \beta), w_3, \gamma) \times \frac{\partial \Psi}{\partial p}(\Phi(x, w_1, \alpha), w_2, \beta),$$

$$\frac{dF}{d\alpha} = \frac{\partial \Phi}{\partial x}(\Psi(\Phi(x, w_1, \alpha), w_2, \beta), w_3, \gamma) \times \frac{\partial \Psi}{\partial x}(\Phi(x, w_1, \alpha), w_2, \beta) \times \frac{\partial \Phi}{\partial p}(x, t, \alpha).$$

Also, given the definition of the solution of an ODE, we have $\frac{\partial \Phi}{\partial t}(., t, p) = A_p(.)$ and $\frac{\partial \Psi}{\partial t}(., t, p) = B_p(.)$. If we want to write these expression using the exponential notation, we get

$$\frac{dF}{dw_3} = A_\gamma e^{w_2 B_\beta} e^{w_1 A_\alpha}(x), \qquad \frac{dF}{d\gamma} = \frac{\partial e^{w_3 A_\gamma}}{\partial \gamma} e^{w_2 B_\beta} e^{w_1 A_\alpha}(x),$$

$$\frac{dF}{dw_2} = \frac{\partial e^{w_3 A_\gamma}}{\partial x} e^{w_2 B_\beta} e^{w_1 A_\alpha}(x) \times B_\beta e^{w_1 A_\alpha}(x), \qquad \frac{dF}{d\beta} = \frac{\partial e^{w_3 A_\gamma}}{\partial x} e^{w_2 B_\beta} e^{w_1 A_\alpha}(x) \times \frac{\partial e^{w_2 B_\beta}}{\partial \beta} e^{w_1 A_\alpha}(x),$$

$$\frac{dF}{dw_1} = \frac{\partial e^{w_3 A_\gamma}}{\partial x} e^{w_2 B_\beta} e^{w_1 A_\alpha}(x) \times \frac{\partial e^{w_2 B_\beta}}{\partial x} e^{w_1 A_\alpha}(x) \times A_\alpha(x); \qquad \frac{dF}{d\alpha} = \frac{\partial e^{w_3 A_\gamma}}{\partial x} e^{w_2 B_\beta} e^{w_1 A_\alpha}(x) \times \frac{\partial e^{w_2 B_\beta}}{\partial x} e^{w_1 A_\alpha}(x) \times \frac{\partial e^{w_1 A_\alpha}}{\partial \alpha}(x).$$

We can see that the computation of the gradients involves many common terms, and that we can backpropagate the information about the gradients through the network. These common terms allow us to compute the gradients in a staged way. Also, in the case where some parameters are constrained to be equal (e.g., $w_1 = w_3$ or $\alpha = \gamma$), the above gradients are only partial gradients and they must be summed up to obtain the total gradients. This situation would occur if we force the composition network to be symmetric, which uses the degrees of freedom of the network more efficiently in some cases.

9.2.2 Prediction of the Lorenz attractor

To demonstrate the efficiency of this method, we build a network and train it for the prediction of the Lorenz attractor. The Lorenz attractor is a system that is chaotic for some appropriate values of its parameters; its prediction has become a classical benchmark in the field of neural networks [145]. We first describe the setup of the experiment in some detail. Band-limited Gaussian white noise η is added to the dynamics of the Lorenz attractor. The system is then of the form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -\beta & 0 & 0 \\ 0 & -\sigma & \sigma \\ 0 & \rho & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} x_1 x_2 \\ 0 \\ -x_2 x_3 \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{bmatrix}$$

$$= C_{\beta,\rho,\sigma} + D + \eta.$$

We can solve the linear differential equation $\dot{x} = C_{\beta,\rho,\sigma}.x$ for any values of β, ρ, σ by exponentiation of the matrix. We can also solve the nonlinear differential equation $\dot{x} = D(x)$ analytically (observe that $\dot{x}_2 = 0$, hence x_2 is a constant). Although nothing guarantees that a splitting of the vector field exists, we often find them easily.

To build a predictor, we consider the parameters of the Lorenz attractor β, ρ, σ to be unknown. The predictor is a second-order composition network as implemented in Figure 9.2:

$$\hat{x}(k+1) = e^{\frac{\Delta t}{2} D} e^{\Delta t C_{\hat{\beta},\hat{\rho},\hat{\sigma}}} e^{\frac{\Delta t}{2} D} x(k).$$

We choose the following operating point as setup for the simulation, and we make comparisons around this operating point. The parameters β, ρ, σ have values equal to $8/3$, 28 and 10 respectively. The power of the noise is $\eta = 0.01$. The training data consists of a set of $N = 100$ data points, $x(k) = [x_1(k), x_2(k), x_3(k)]^T (k = 1, \ldots, N)$, on a trajectory sampled at time interval $\Delta t = 0.01$. The test data consists of $M = 100$ data points. The error criterion is the sum of all squared errors $SSE = \sum_{i=1}^{M} \|\hat{x}(i) - x(i)\|^2$. To help us evaluate the performance of the different models, we first perform the following sanity checks. The sum of all squared test samples is $SS = 78766$. The optimal predictor (in the least-square sense) is the noise-free Lorenz attractor. Its test error is $SSE_{\text{Lorenz}} = 64.17$. A trivial predictor would be to hold the value of the current sample as prediction for the next sample. The test error would then be $SSE_{\text{trivial}} = 120.42$. Interesting models therefore have to have an error at least smaller than SS_{trivial}; and their error is bounded below by SS_{Lorenz}.

The first aspect we want to examine is the influence of the order of the model. We compare a first-, second- and fourth-order method. We use a Levenberg–Marquardt optimization routine for the training of the network.

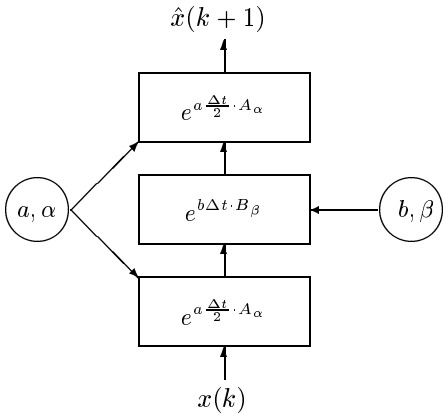


Figure 9.2: Implementation of a second-order composition network. This composition network incorporates the splitting of the Lorenz attractor.

We perform these experiments (and all subsequent ones) in the Matlab environment. We see that we can rapidly make the error SE_{test} close to the minimum error by increasing the order of the model (Table 9.1). The second

Order	SE_{test}	$\hat{\beta}$	$\hat{\sigma}$	$\hat{\rho}$
First	68.18	3.7232	9.7584	27.0935
Second	66.29	3.1825	10.9024	27.5680
Fourth	66.34	3.1685	10.8836	27.4256
Exact	64.17	2.6666	10.0000	28.0000

Table 9.1: Performance of the composition network w.r.t. the order of composition. The table gives the SE on the test set for different orders and for the true system. It also gives the value of the weights of the composition network.

aspect we want to consider is the influence of the noise level η. We look at a fourth-order network for increasing levels of noise. We see that the performance remains close to the optimum independently of the noise level (Table 9.2). The last variable we examine is the size of the training set. The major advantage of this approach is that we have incorporated the a priori knowledge we had about the system. This preservation of the structure of the problem ensures that the estimation is efficient even for small sizes of the training set (Fig. 9.3). Simultaneously, we compare the performance of our method with

η	SSE_{test}	SSE_{Lorenz}
0	3.10^{-6}	0
0.01	5.46	5.33
0.1	66.34	64.17
1	2776	2773

Table 9.2: Influence of the noise. For different noise levels, the table gives the SSE on the test set both for the composition network and the true system.

that of a single–hidden-layer perceptron (which is a particular case of multilayer perceptron (MLP)) with five hidden neurons (we chose the number of neurons that gave the neural network the best performances). With a training set of 100 points, the MLP fails to produce any meaningful solution. A MLP requires a minimum of 150 samples for training, in contrast with the composition network—which is consistently efficient for training sets having as few as 30 samples (Fig. 9.3). A last remark is that the floating-point–operation count

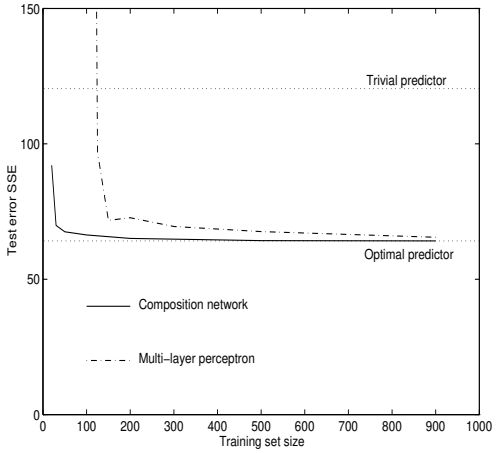


Figure 9.3: Composition network vs. MLP. The composition network, through an efficient of a priori knowledge, gives a much lower estimation error than the multilayer perceptron. It also works with a small number of samples.

of our method is 10 to 100 times lower than that of the MLP.

A method related to these composition networks is direct parameter estimation. Since we have assumed that we know the differential equation up to some unknown parameters, we could find a model by direct integration of the

equation (using, for example, a Runge–Kutta method) and then try to estimate what values of the parameters give a best fit. This approach is essentially the same as ours, but it is conceptually simpler since we would not have to worry about Lie algebras, BCH formulas and other differential geometric considerations. The main argument in favor of the composition network is that conventional integration methods are computationally intensive and that using them in an estimation procedure makes the estimation tragically slow. Also, the Lie algebraic approach sheds new light on the behavior of continuous-time systems observed in discrete-time.

9.2.3 Irregularly sampled systems

Another area where we can expect improvement from the composition network compared to other neural architectures is for the prediction of irregularly sampled systems. It is difficult to use discrete-time neural networks for the prediction of systems if the measurements are irregularly sampled. To use discrete-time networks, we would need to interpolate the measurements between the samples to obtain a series sampled at discrete time intervals, but this procedure might not be robust if noise is present in the system. Composition networks do not suffer from this drawback since their output is function of the sampling time Δt, which does not need to be fixed. In this case, composition networks provide an alternative to having to use continuous-time neural networks. We look again here at the Lorenz attractor with noise added on its dynamics and sampled at irregular time intervals. The data is produced by a Matlab–Simulink simulation. The average time step is $\mu(\Delta t) = 0.0533\,s$, the standard deviation of the time step is $\sigma(\Delta t) = 0.0172\,s$. Band-limited white noise (in Matlab) is added to the dynamics of the attractor, the noise has a sampling time of one second and power of 1000. There is a total of 1841 state samples covering a time window of 100 seconds. The model used for estimation is a fourth-order composition network similar to those presented before. After 68 training iterations using the Levenberg–Marquardt optimization algorithm, the reconstruction error is $SSE = 13246$. Although this error is large, we have to consider the fact that a large amount of noise was added to the dynamics. The following figure (Fig. 9.4) shows first, on the left, the evolution of the state in the system under the influence of the dynamics noise. We can see that the trajectory is seriously perturbed by the added noise since the trajectory is repeatedly crossing itself. Then we use the model estimated by training with $\hat{\beta} = 2.59$, $\hat{\sigma} = 5.35$, $\hat{\rho} = 31.64$ in a recurrent loop. We see in the same figure, on the right, that the model learned by training has recovered the chaotic attractor of the Lorenz system, although this system was irregularly sampled and a significant amount of dynamics noise was added.

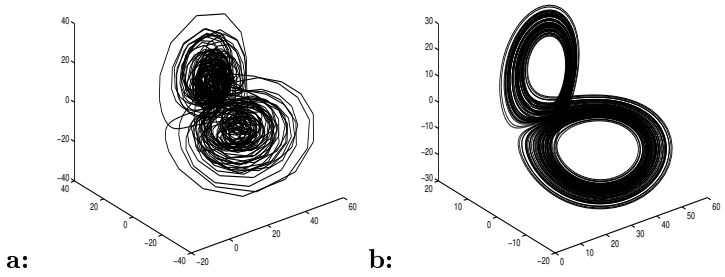


Figure 9.4: Irregularly sampled Lorenz attractor with noise. **a:** Original data. **b:** Reconstructed attractor. The reconstruction captures the behavior of the Lorenz attractor accurately.

9.3 System identification with recurrent composition networks

The feedforward composition network as we have presented it so far is not practically usable for identification or prediction. The problem is that the assumption that we can observe the full state of the system rarely holds in practice. If we want to create a useful tool, we therefore need to use an observer in combination with a composition network. Any observation strategy is possible. As an example, we develop here a recurrent composition network. We will suppose that we can observe a subset of all state variables. This assumption is not too stringent since we saw, in Chapter 6, that we can replace any nonlinear control system whose output is a nonlinear function of the state by some equivalent system that has a state of higher dimension and has a subset of the state variables as output. Thus, the systems that we consider have a d-dimensional state and an f-dimensional output:

$$\begin{cases} \dot{x}(t) = X(x(t)), \\ y(k) = (x_{i_1}(k.\Delta t), x_{i_2}(k.\Delta t), \ldots, x_{i_f}(k.\Delta t))^T, \end{cases}$$

where $\{i_1, , \ldots, i_f\}$ is a subset of $\{1, \ldots, d\}$ with cardinal equal to $f \leq d$.

We propose to use teacher forcing [142] to design the network:

$$\begin{aligned} (\hat{x}_1(k+1), \ldots, \hat{x}_d(k+1))^T &= \hat{F}((\tilde{x}_1(k), \ldots, \tilde{x}_d(k))^T), \\ \text{where} \quad \tilde{x}_i(k) &= \begin{cases} x_i(k), & \text{if } x \text{ is observed,} \\ \hat{x}_i(k), & \text{if } x \text{ is not observed.} \end{cases} \end{aligned}$$

9.3.1 Identification of the Lorenz attractor

We take the Lorenz attractor and assume that the output is equal to the second state variable y. A second-order recurrent composition network would look as follows (Fig. 9.5). The setup is identical to the feedforward case (Sec-

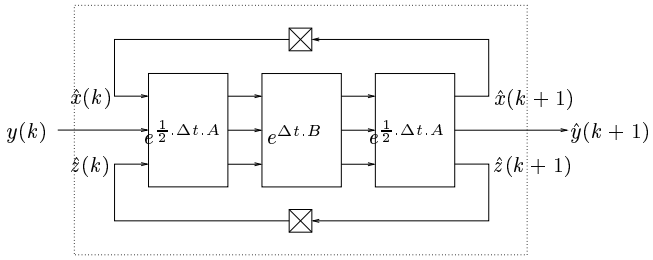


Figure 9.5: Recurrent composition network for the Lorenz attractor. The architecture uses teacher forcing for training: It feeds the observed measurement into the system, replacing the predicted value at each time step.

tion 9.2.2) except that the data is only a time series of the second state-variable $y(1), \ldots, y(1000)$. We estimate the parameters of the recurrent composition network by fitting our composition network to this time series. In the following figure (Fig. 9.6), the noise is band-limited Gaussian white noise with noise power $\eta = 0.01$. We then use the estimated composition network to simulate the Lorenz attractor. It does reproduce the behavior of the Lorenz attractor well. In a second experiment, we compare the prediction of the composition network to the prediction of the exact Lorenz attractor (teacher forcing architecture with the exact model instead of a composition network). We also compare it to the trivial prediction (predicting the next sample equal to the sample currently observed). We train the network on the first 900 samples and test it on the last 100 samples. The error criterion is the sum of squared error SSE. These comparisons are summarized in Table 9.3.1. The recurrent composition network achieves performances close to that of the Lorenz attractor with the same teacher forcing strategy. For high error levels ($\eta = 1$), we see that the teacher forcing strategy becomes inappropriate (results worse than trivial). This failure of teacher forcing as well for the composition network as for the ideal model means that other observation strategies are necessary in the case of high noise.

9.3.2 Identification of control-affine systems

In our presentation of recurrent composition networks, there is so far no way to deal with systems that have inputs. We can remove this restriction in the case

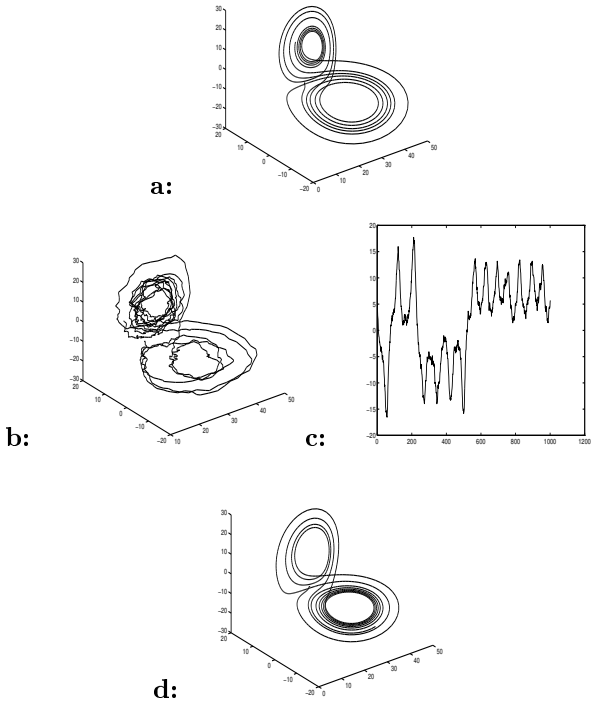


Figure 9.6: Reconstruction of the noisy Lorenz attractor. **a:** Noise-free Lorenz attractor. **b:** Lorenz attractor with additional noise. **c:** Time series used for reconstruction. **d:** Reconstructed attractor. The reconstructed attractor accurately captures the behavior of the noise-free Lorenz attractor, even in the presence of noise.

noise	SSE_{comp}	SSE_{Lorenz}	SSE_{trivial}
0	1.75e-6	2.65e-13	8.07
0.01	2.32	2.21	23.2
0.1	25.0	24.54	27.69
1	347	361	290

Table 9.3: Performance of the recurrent composition network. For different noise levels, the table gives the SSE on the test set for the composition network, the true system, and the trivial sample-and-hold predictor.

of affine controls as follows. These systems are described by an affine input-driven differential equation: $\dot{x} = X(x(t)) + \Sigma_{j=1}^{m} Y_j(x(t))u_j(t)$. The system has thus an m-dimensional input u. The central assumption here is that the control variable $u(t)$ remains constant during each time interval $[k\Delta t, (k+1)\Delta t[$. This is often the case in digital control systems. Thus, between two sampling times $k\Delta t$ and $(k+1)\Delta t$, the behavior of the system is given by $\dot{x} = X(x(t)) + \Sigma_{j=1}^{m} Y_j(x(t))u_j(k\Delta t)$ together with the initial condition $x_0 = x(k\Delta t)$. Note again that the inputs $u_j(k\Delta t)$ are constants. Assuming that we can split the vector fields X and Y_j into elementary parts: $X = \Sigma_{i=1}^{p} A_i$ and $Y_j = \Sigma_{k=1}^{q(j)} B_{jk}$, we have that the total vector field is a linear combination of elementary parts that we can solve separately. Thus, we are in the situation of Equation 9.3 and we can develop a composition integrator for this system. Now, once the system has evolved till $t = (k+1)\Delta t$, its behavior will change since the input will change from $u(k\Delta t)$ to $u((k+1)\Delta t)$. But the vector field remains a (different) linear combination of the same elementary vector fields. This fact implies that the structure of the composition network will not change, but that we need to update its coefficients proportionally to the new input value. To make this clear, we consider the following example.

We have again the Lorenz attractor where we observe the second state y, but this time the system has an additional input u on the first state x.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -\beta & 0 & 0 \\ 0 & -\sigma & \sigma \\ 0 & \rho & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} x_1 x_2 \\ 0 \\ -x_2 x_3 \end{bmatrix} + \begin{bmatrix} u \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{bmatrix}$$

$$= C_{\beta,\rho,\sigma} + D + U + \eta.$$

We develop a fourth-order composition integrator for this system by splitting it into two parts: the affine part (merging the effect of $C_{\beta,\rho,\sigma}$ and U into a single vector field) and the nonlinear part. We choose the same operating point as before with $\beta = 8/3$, $\sigma = 10$, and $\rho = 28$. The input is the sum of two sinusoids: the first with an amplitude of 10 and a frequency of 1 radian per second, the second with an amplitude of 3 and a frequency of 3.5 radians per second; both sinusoids have phase zero. The system is sampled at $\Delta t = 0.1$—which is a rather large sampling time—giving a time series of 1000 samples for identification. The input is sampled at the same frequency. The noise has a power of one and is band-limited Gaussian noise sampled at $\Delta t = 0.1$ also. For identification, we use only the input u and second state y as output. The error after training is $SSE = 705.0$ (this residual is essentially caused by the noise; in a noise free experiment we get a residual of $SSE = 6.0770$) and the parameters of the composition network are $\hat{\beta} = 2.8805$, $\hat{\sigma} = 8.9805$, and $\hat{\rho} = 27.2005$. The following plots (Fig. 9.7) present (1) the dynamics of the Lorenz attractor under the action of input and the effect of the sampling time $\Delta t = 0.1$, (2) the input u sum of two sinusoids, (3) the time series of 1000 samples of y used

for identification, and (4) the reconstruction of the Lorenz attractor we get when we use the composition network in a recurrent loop without any external input. We can see that the reconstruction is almost perfect, even with the large sampling time and the noise.

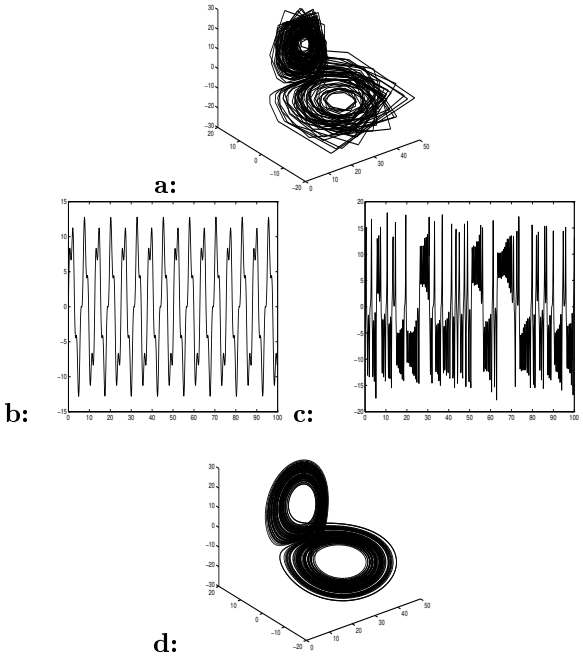


Figure 9.7: Lorenz attractor with input. **a:** trajectory in state space. **b:** Input (sum of sinusoids). **c:** Time series used for reconstruction. **d:** Autonomous behavior of the estimated system. It reproduces the dynamics of the Lorenz attractor.

9.4 MLPs in dynamics space

The method we have presented so far relies on some decomposition of the equations of the system. However, the knowledge of these equations might not always be available. It would therefore be interesting to take advantage of the universal approximation properties of neural networks. If we decide to approximate the vector field of our system by an MLP, we can derive a composition network to implement this approximation. The unknown differential equation

is (8.3). And we want to approximate it by $\dot{x}(t) = A(x(t))$, where the vector field is of the following form:

$$A(x) = \sum_{j=1}^{n} \vec{c}_j \sigma(\vec{b}_j x + b_{j_0}) = \sum_{j=1} A_j^{\vec{c}_j, \vec{b}_j}(x), \tag{9.4}$$

where $\sigma(x) = \tanh(x)$ and $\vec{c}_j, \vec{b}_j \in \mathbb{R}^{d+1}, j = 1, \ldots, n$.

To find the associated composition network, we have to solve the differential equation $\dot{x} = A^{\vec{c},\vec{b}}(x)$. To do this, we first solve the one-dimensional problem, which we use to derive the solution for the multidimensional case. Finally, we derive a composition network where each transformation is the solution of a sigmoidal ODE, and can thus approximate arbitrary dynamical systems.

9.4.1 One-dimensional sigmoidal ODEs

We want to solve $\dot{x} = \sigma(x), x \in \mathbb{R}$ with a hyperbolic tangent for the activation function σ. That is,

$$\dot{x} = \tanh x.$$

Using the change of variables $y = \tanh x$, we get

$$\dot{y} = (1 - y^2)y.$$

We solve by separation of variables:

$$\int_y^{y_0} \frac{dy}{(1-y^2)y} = t - t_0.$$

Assuming $y_0 > 0$, we integrate by splitting into simple fractions:

$$\ln\left|\frac{y}{y_0}\right| - \frac{1}{2}\ln\left|\frac{y-1}{y_0-1}\right| - \frac{1}{2}\ln\left|\frac{y+1}{y_0+1}\right| = t.$$

Taking the exponential of the opposite of both sides, we get

$$\sqrt{\frac{y^2-1}{y^2}}\sqrt{\frac{y_0^2}{y_0^2-1}} = e^{-t}.$$

Since we have assumed $y_0 > 0$ and given that the vector field has a fixed point at zero, y has to remain positive for all times. Using this fact and the previous equation, we can solve for y and get

$$y = \left(1 - (1 - \frac{1}{y_0^2})e^{-2t}\right)^{-1/2}.$$

Substituting back $x = \operatorname{atanh} y$, we define, for $x_0 > 0$, the function $h(x_0, t)$:

$$h(x_0, t) = \operatorname{atanh}\Big((1 + \frac{e^{-2t}}{\sinh^2 x_0})^{-1/2}\Big).$$

For $x_0 = 0$, we have $x(t) = 0$, for all t. For $x_0 < 0$, we can directly use the symmetry of the vector field $\tanh x = -\tanh(-x)$. We finally get

$$x(t) = \Phi(x_0, t) = \begin{cases} h(x_0, t), & \text{if } x_0 > 0, \\ 0, & \text{if } x_0 = 0, \\ -h(-x_0, t), & \text{if } x_0 < 0. \end{cases} \tag{9.5}$$

9.4.2 Multidimensional sigmoidal ODEs

We can now use the solution of the one-dimensional case to explicitly integrate the multidimensional system $\dot{x} = A^{\vec{c},\vec{b}}(x) = \vec{c}\sigma(\vec{b}^T x + b_0)$ for any value of the parameters $\vec{c}, \vec{b}$. To this end, we use the fact that the dynamics of a single neuron is essentially one-dimensional [96]. To show this, we use a transdimensional change of coordinates. We have shown in Chapter 6 that the following two systems are equivalent:

$$\begin{cases} \dot{x} = C\sigma(Bx + b_0), \\ x(0) = x_0 \end{cases} \iff \begin{cases} \dot{z} = \sigma(BCz + B(I - CC^\dagger)x_0 + b_0), \\ x = Cz + (I - CC^\dagger)x_0, \\ z(0) = C^\dagger x_0, \end{cases}$$

where $C^\dagger$ denotes the pseudoinverse of C. This equivalence means that in the present case, we want to solve the system

$$\begin{cases} \dot{z} &= \sigma(\vec{b}^T\vec{c}z + \vec{b}^T(I - \vec{c}\frac{\vec{c}^T}{\|\vec{c}\|^2})x_0 + b_0), \\ x &= \vec{c}z + (I - \vec{c}\frac{\vec{c}^T}{\|\vec{c}\|^2})x_0, \\ z(0) &= \frac{\vec{c}^T}{\|\vec{c}\|^2}x_0 \end{cases}$$

since $\vec{c}^\dagger = (\vec{c}\vec{c}^T)^{-1}\vec{c} = \frac{\vec{c}^T}{\|\vec{c}\|^2}$. Using the change of coordinates $y = \vec{b}^T\vec{c}z + \vec{b}^T(I - \vec{c}\frac{\vec{c}^T}{\|\vec{c}\|^2})x_0 + b_0$, and the rescaling of time $s = (\vec{b}^T\vec{c})t$, we get the equivalent system

$$\begin{cases} \frac{dy}{ds} &= \sigma(y), \\ x &= x_0 + \vec{c}\left(\frac{y - (\vec{b}^T x_0 + b_0)}{\vec{b}^T\vec{c}}\right)z, \\ y(0) &= \vec{b}^T x_0 + b_0, \end{cases}$$

if $\vec{b}^T\vec{c} \neq 0$. If $\vec{b}^T\vec{c} = 0$, we get the system

$$\begin{cases} \dot{z} &= \sigma(\vec{b}^T x_0 + b_0), \\ x &= \vec{c}z + (I - \vec{c}\frac{\vec{c}^T}{\|\vec{c}\|^2})x_0, \\ z(0) &= \frac{\vec{c}^T}{\|\vec{c}\|^2}x_0. \end{cases}$$

Using the solution Φ of the one-dimensional sigmoid (9.5), we can express the solution of the multidimensional sigmoid as

$$x(t) = e^{tA^{\vec{c}_j,\vec{b}_j}} = \begin{cases} x_0 + \left(\frac{\Phi(z_0, dt) - z_0}{d}\right)\vec{c}, & \text{if } d_1 \neq 0, \\ x_0 + (t\sigma(z_0))\vec{c}, & \text{if } d_1 = 0; \end{cases}$$

$$\text{with } \begin{cases} d = \vec{b}^T\vec{c}, \\ z_0 = \vec{b}^T x_0 + b_0, \end{cases}$$

9.4.3 Universal approximation by compositions

Therefore, we can design a network of the following form:

$$\hat{x}(k+1) = \hat{F}(x(k)) = e^{A^{\vec{c}_k,\vec{b}_k}} \dots e^{A^{\vec{c}_1,\vec{b}_1}}(x(k)). \tag{9.6}$$

We call this architecture an "MLP in dynamics space" as an MLP is implicitly used to parameterize the vector field. The MLP possesses universal approximation properties in the space of vector fields [28]. This property of MLPs means that, on some compact subset Ω of $\mathbb{R}^n$, we have the following. For any vector field X and for any approximation error ε, there exists an order of composition n and parameters $c_j, b_j, b_{j_0}, j = 1, \dots, n$, with $A(x) = \sum_{j=1}^n \vec{c}_j.\sigma(\vec{b}_j.x + b_{j_0}) = \sum_{j=1} A_j^{\vec{c}_j,\vec{b}_j}(x)$ such that $|X(x) - A(x)| < \varepsilon$, for any $x \in \Omega$.

As we showed in Chapter 6 (using an elementary application of the Gronwall inequality [42]) that universal approximation of the vector field guarantees universal approximation of the dynamical system for any finite time. This is the case here, as we work with a finite sampling-time Δt. Furthermore, Equation 9.3 shows that the "MLP in dynamics space" can approximate the solution of the dynamical equation parameterized by the MLP arbitrarily well. This parameterization of the vector field by a MLP gives the "MLP in dynamics space" a universal approximation property in the group of all time-Δt maps of smooth dynamical systems. This property implies that, for any system defined by (8.3) with recurrence map defined by (9.1), we have

$$\exists k, \exists c_j, b_j, j = 1, \dots, k, \text{ such that } |F_c(x) - \hat{F}(x)| < \varepsilon, \qquad \forall x \in \Omega.$$

9.5 Compositions and the problem of motion-planning

To bridge the gap between the topics discussed in Chapter 7 and the use of compositions to approximate dynamical systems, we present a problem that links them naturally: the computation of an open-loop control law that steers some initial condition of a control system into a desired final state. Controllability is enough to guarantee that such a control law exists, but it does not say anything about what this control is. More specifically, we examine the motion-planning problem for controllable systems without drift as discussed by Lafferiere and Sussman [72]. We look at systems of the form

$$\dot{x} = \sum_{i=1}^{m} A_i(x) u_i, \tag{9.7}$$

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ under the assumption that $A_1(x), \ldots, A_m(x)$ are analytic vector fields and the system (9.7) is controllable. By the controllability rank condition applied to analytic affine systems, we know that the system is controllable if the dimension of the involutive closure $\mathcal{G}$ of the distribution $\mathcal{G}_0(x) = \operatorname{span}\{A_1(x), \ldots, A_m(x)\}$ is equal to n for all $x \in \mathbb{R}^n$. We call $\mathcal{G}$ the controllability Lie algebra of (9.7) and denote it from now on by $\mathcal{L}(A_1, \ldots, A_m)$. The motion-planning problem—which we also call the steering problem—is, given an initial condition x_0 and a final condition x_1, to find a control $u(t) = (u_1(t), \ldots, u_m(t))$ such that there exists a finite time T after which the state of the system, starting at x_0, has reached x_1:

$$\forall x_0, x_1 \in \mathbb{R}^n, \text{ find } T \in \mathbb{R}, u : [0,T] \to \mathbb{R}^n : \begin{cases} \dot{x}(t) = \sum_{i=1}^{m} A_i(x(t)) u_i(t), \\ x(0) = x_0, \\ x(T) = x_1. \end{cases}$$

The final time T is arbitrary because the system is control affine; if $u(t)$ is such that $x(0) = x_0$ and $x(T) = x_1$, then $v(t) = u(t)/\alpha$ with $\alpha \neq 0$ is such that $x(0) = x_0$ and $x(\alpha T) = x_1$. There are several approaches to the solution of this problem [54, 107, 124], but we discuss an approach that makes extensive use of exponential maps. The additional ingredients are the notion of nilpotent and nilpotentizable systems and the Hall basis of a Lie algebra, which we define shortly. The proposed approach works exactly for nilpotent and nilpotentizable systems; for general systems, we use successive approximations that converge to an exact solution. We abandon the notion of a distribution to consider the equivalent concept of a Lie algebra of vector fields. As discussed in Chapter 7, we can view an involutive distribution as a Lie algebra (as it is closed under the bracketing of two of its vector fields). We now review the notions of a nilpotent system and of a Hall basis of a Lie algebra.

9.5.1 Nilpotent Lie algebras and Hall bases

Definition 9.1. A Lie algebra $\mathcal{L}$ is *nilpotent of order k* if k is the smallest positive integer for which all the direct Lie brackets of order k are identically equal to zero:

$$\text{For all } A_1, \dots, A_{k+1} \in \mathcal{L}, \text{ we have } [A_1, [A_2, [\dots, [A_k, A_{k+1}]] \cdots]] = 0. \quad (9.8)$$

A system (9.7) is *nilpotent of order k* if its controllability Lie algebra is *nilpotent of order k*.

Note that Condition 9.8 is equivalent to the condition that any bracket of order $k+1$ be equal to zero. The idea is now to consider an extended system similar to (9.7) but where we add a number of control vector fields that belong to the controllability algebra $\mathcal{L}(A_1, \dots, A_m)$ of the original system. If the controllability Lie algebra is nilpotent of order k, we denote it by $\mathcal{L}_k(A_1, \dots, A_m)$. We then solve the steering problem for this extended system, thereby getting a steering control $v(t)$; we then use this control $v(t)$ to produce a steering control $u(t)$ for the motion-planning problem on the original system.

The extended system is of the form

$$\dot{z} = A_1(z)v_1 + \cdots + A_m(z)v_m + A_{m+1}(z)v_m + \cdots + A_r(z)v_r, \quad (9.9)$$

where $z \in \mathbb{R}^n$ (just as x), $A_1, \dots, A_m$ are identical to the control vector fields of (9.7) and $A_{m+1}, \dots, A_r$ belong to the controllability algebra $\mathcal{L}_k(A_1, \dots, A_m)$ and we choose them such that $\operatorname{span}\{A_1(x), \dots, A_r(x)\} = \mathbb{R}^n$ for all $x \in \mathbb{R}^n$. We first solve the steering problem for (9.9). We choose an arbitrary time T and an arbitrary C^1 curve $w(t) : [0, T] \to \mathbb{R}^n$ such that $w(0) = x_0$ and $w(T) = x_1$ (for example, a straight line between x_0 and x_1). We then write the vector $\dot{w}(t)$ tangent to this curve as $\dot{w}(t) = A_1(w(t))v_1(t) + \cdots + A_m(w(t))v_m(t) + A_{m+1}(w(t))v_{m+1}(t) + \cdots + A_r(w(t))v_r(t)$. We can actually solve this relation for $v(t)$ as we have $w(t), A_1(w(t)), \dots, A_r(w(t))$ already available. This solution always exists since $\operatorname{span}\{A_1(x), \dots, A_r(x)\} = \mathbb{R}^n$ for all $x \in \mathbb{R}^n$. We find the solution at time t by computing a pseudoinverse of the matrix $[A_1(w(t)), \dots, A_r(w(t))]$ and multiplying it by $\dot{w}(t)$; in this way we obtain $v(t)$.

Now, we want to use this steering control $v(t)$ of the extended system to determine a steering control $u(t)$ for the original system (9.7). We proceed by first requiring that the vector fields $A_{m+1}, \dots, A_r$ that we use to construct the extended system be Lie brackets of a particular type. Not only must they belong to the controllability Lie algebra $\mathcal{L}_k(A_1, \dots, A_m)$ (which contains all possible Lie brackets of $A_1, \dots, A_m$) but we require further that they be elements of a Hall basis of this Lie algebra. We will then make formal computations on the Lie algebra $\mathcal{L}_k(A_1, \dots, A_m)$ using the Hall basis, and we will deduce the steering control $u(t)$ from these formal computations.

The necessity of a basis for a Lie algebra arises when we try to represent all the elements of the algebra in terms of a number of elementary brackets. The problem is the following. Suppose we have only two vector fields $A(x)$ and $B(x)$, and that they form a nilpotent algebra of order 3 (which implies that all third-order brackets, such as $[A,[A,[A,B]]]$, are equal to zero). We therefore know that we can write any vector field belonging to $\mathcal{L}_3(A,B)$ as a linear combination of Lie brackets of order two and lower. However, the elementary Lie brackets are related by antisymmetry and by the Jacobi identity (which imply $[A,B] = -[B,A]$ and $[A,[A,B]] = -[A,[B,A]]$), so the linear combination is not unique and we want to pick only a subset of the elementary brackets to build our representations. We are thus looking for a basis $\mathcal{B}$ of the Lie algebra; in the case of $\mathcal{L}_3(A,B)$, we can choose $\mathcal{B} = \{A, B, [A,B], [A,[A,B]], [B,[A,B]]\}$. Finding such a basis is not trivial in general because the number of elementary brackets grows combinatorially with k and exponentially with m; the Hall basis gives a systematic method for finding such a basis. For this procedure, we consider first the set $\mathcal{F}_m$ of all the expressions that are given by taking the Lie brackets of $A_1, \ldots, A_m$ up to any order. It is important to note that the elements of the set $\mathcal{F}_m$ are expressions (such as $[A_1,[A_2,A_3]]$ and $[[A_3,A_2],A_1]$) and not the functions these brackets represent (since $[A_1,[A_2,A_3]]$ is equal to $[[A_3,A_2],A_1]$). We therefore call $\mathcal{F}_m$ the set of formal Lie monomials in $A_1, \ldots, A_m$. If we denote by order(F) the bracketing order of an element F of $\mathcal{F}_m$ (for example, order$([[A_1,A_2],[A_2,A_3]]) = 3$), we are now in position to define a Hall basis.

Definition 9.2. A *Hall basis* of the controllability algebra $\mathcal{L}(A_1, \ldots, A_m)$ is a subset $\mathcal{B}$ of $\mathcal{F}_m$ such that

1. The vector fields $A_1, \ldots, A_m$ belong to $\mathcal{B}$.
2. If $F \in \mathcal{F}_m \setminus \{A_1, \ldots, A_m\}$, then $F \in \mathcal{B}$ if and only if either there exist $G, H \in \mathcal{B}$ such that $F = [G,H]$ with order$(G) <$ order(H) and $H \in \{A_1, \ldots, A_m\}$, or there exist $G, H, K \in \mathcal{B}$ such that $F = [G,[H,K]]$ with order$(K) \leq$ order(G).

If the controllability algebra is nilpotent of order k, we get a Hall basis $\mathcal{B}_k$ by discarding all elements of order superior to k.

9.5.2 Hall coordinates and compositions of exponential maps

Now, we use this Hall basis to define our extended system. Consider again the system

$$\dot{x} = A_1(x)u_1 + \cdots + A_m(x)u_m \tag{9.10}$$

and assume it has a nilpotent controllability Lie algebra of order k. Choose the new vector fields as elements of the Hall basis $(\{A_{m+1}, \ldots, A_r\} \subset \mathcal{B}_k \setminus$

$\{A_1, \ldots, A_m\}$) under the condition that span $\{A_1(x), \ldots, A_r(x)\} = \mathbb{R}^n$ for all $x \in \mathbb{R}^n$. The extended system then becomes

$$\dot{z} = A_1(z)v_1 + \cdots + A_m(z)v_m + A_{m+1}(z)v_{m+1} + \cdots + A_r(z)v_r. \tag{9.11}$$

Now, we consider the set $\mathcal{E}_k$ consisting of the exponential maps e^X of the elements of the nilpotent controllability Lie algebra (we write the exponential maps before as $e^{\Delta t X}$, so we consider here the case $\Delta t = 1$):

$$\mathcal{E}_k(A_1, \ldots, A_m) = \left\{ e^A : \mathbb{R}^n \to \mathbb{R}^n \text{ for which } A \in \mathcal{L}_k(A_1, \ldots, A_m) \right\}.$$

The following facts are then known about the elements of this set [72]. For any $X \in \mathcal{E}_k(A_1, \ldots, A_m)$, with $\mathcal{B} = \{A_1, \ldots, A_s\}$ a Hall basis of $\mathcal{L}_k(A_1, \ldots, A_m)$, there exists a *unique* vector $h \in \mathbb{R}^r$ such that

$$X = e^{h_1 A_1} e^{h_2 A_2} \ldots e^{h_s A_s}, \tag{9.12}$$

and another unique vector $l \in \mathbb{R}^r$ such that

$$X = e^{l_s A_s} e^{l_{s-1} A_{s-1}} \ldots e^{l_1 A_1}. \tag{9.13}$$

Note that X is a map from $\mathbb{R}^n$ to $\mathbb{R}^n$ if our vector fields live in $\mathbb{R}^n$ and the previous expressions define compositions of invertible maps. Moreover, the vector h (and the vector l as well) define a coordinate system on our set $\mathcal{E}_k(A_1, \ldots, A_m)$ of invertible maps, which we can actually show to be a group. We call h the forward Hall coordinates of the map X and we call l the backward coordinates of X. Since we have a coordinate system on $\mathcal{E}_k(A_1, \ldots, A_m)$, we can actually define ODEs on the subset $\mathcal{C} \subset \mathbb{R}^r$ of possible coordinates.

We solve the following ODE on $\mathcal{E}_k(A_1, \ldots, A_m)$, which we call the formal extended equation:

$$\begin{array}{rcl} \dot{X}(t) & = & A_1(X(t))v_1(t) + \cdots + A_r(X(t))v_r(t), \\ X(0) & = & I. \end{array} \tag{9.14}$$

Note that this differential equation represents the evolution over time of a *map* from $\mathbb{R}^n$ to $\mathbb{R}^n$, hence the initial condition $X(0) = I$, where I denotes the identity map. This object is somewhat different from the ODEs we have considered so far, but it guarantees that each state evolves according to $\dot{x}(t) = A_1(x(t))v_1(t) + \cdots + A_r(x(t))v_r(t)$. Moreover, we fall back onto an ODE by using the fact that the set $\mathcal{E}$ of exponential maps is here a finite-dimensional manifold.

We posit a solution to the formal extended differential equation (9.14) in terms of backward Hall coordinates:

$$X(t) = e^{l_s(t) A_s} e^{l_{s-1}(t) A_{s-1}} \ldots e^{l_1(t) A_1}. \tag{9.15}$$

Although we cannot describe the necessary steps explicitly here, for lack of space, the use of backward Hall coordinates allows us to compute the vector function $l(t)$ easily by solving some s-dimensional ODE in l driven by $v(t)$. To find this ODE, we need to differentiate Equation 9.15 with respect to time (here are the steps that require some care) and then identify the resulting coefficients with the formal extended equation. Once we have this ODE in l driven by $v(t)$, we use the fact that we have already computed $v(t)$ in the first step of our discussion by solving a system of linear equations. Although we cannot prove here that we can always find $l(t)$ as the solution to some ODE and refer to [134] for the idea of the proof, we illustrate the application of the method by an example.

Suppose we have two vector fields A and B on $\mathbb{R}^2$ ($m = 2$) such that their controllability Lie algebra is nilpotent of order 3 ($k = 3$). A Hall basis of $\mathcal{E}_3(A, B)$ is $\mathcal{B} = \{A, B, [A, B], [A, [A, B]], [B, [A, B]]\}$ ($s = 5$). Suppose further that $\operatorname{span}\{A(x), B(x), [A, B](x), [A, [A, B]](x)\} = \mathbb{R}^2$ for all $x \in \mathbb{R}^2$ ($r = 4$). We then set $A_1 = A, A_2 = B, A_3 = [A, B], A_4 = [A, [A, B]]$ for the extended system and $A_5 = [B, [A, B]]$ for the rest of the Hall basis. We can write any exponential map $X \in \mathcal{E}_3(A, B)$ uniquely in backward Hall coordinates as $X = e^{l_5 A_5} \dots e^{l_1 A_1}$. We can then compute the backward Hall coordinates by solving the system of differential equations (although we do not discuss here the details of how to obtain this system)

$$\begin{aligned}
\dot{l}_1 &= v_1, \\
\dot{l}_2 &= v_2, \\
\dot{l}_3 &= l_1 v_2 + v_3, \\
\dot{l}_4 &= (1/2) l_1^2 v_2 + l_1 v_3 + v_4, \\
\dot{l}_5 &= l_2 v_3 + l_1 l_2 v_2, \\
l(0) &= 0.
\end{aligned}$$

Now, there is a simple algebraic transformation that will produce the solution in forward Hall coordinates $h(t)$ from the solution in backward Hall coordinates.

However, we still need to find the steering control $u(t)$. We do this by solving again a formal equation similar to (9.14) on the group of exponential maps, except that the equation is not extended anymore:

$$\begin{aligned}
\dot{Y}(t) &= Y(t)(A_1 u_1(t) + \dots + A_m u_m(t)), \\
Y(0) &= I.
\end{aligned}$$

The question is thus how to deduce $u(t)$ from the knowledge of $v(t)$ and $h(t)$. We know that $v(t)$ steers x_0 into x_1 at time T. So, we have that $X(T)(x_0) = x_1$ (remember that $X(t)$ is a map from $\mathbb{R}^n$ to $\mathbb{R}^n$). But we know the forward Hall

coordinates of $X(T)$: We have that $X(T) = e^{h_1(T)A_1}e^{h_2(T)A_2}\dots e^{h_s(T)A_s}$. We can reinterpret this formula as follows: The control sequence

$$\bar{h}(t) = \begin{cases} (h_1(T), 0, \dots, 0), & \text{for } t \in [0,1[, \\ (0, h_2(T), 0, \dots, 0), & \text{for } t \in [1,2[, \\ \vdots & \\ (0, \dots, 0, h_s(T)), & \text{for } t \in [s-1, s], \end{cases}$$

steers the extended system (9.10) from x_0 into x_1 at time $t = s$. We can deduce the steering control $u(t)$ for the original system (9.10) from this control sequence by using the properties of nilpotency.

It is easier to discuss the procedure at the hand of an example. Suppose that we have a nilpotent system of order 3 with two vector fields A and B; and suppose $X(T) = e^{\alpha A}e^{\beta B}e^{\gamma[A,B]}e^{\delta[A,[A,B]]}e^{\epsilon[B,[A,B]]}$. If we use a constant control $u(t) = (1,0,0,0,0)$ (for the original system (9.7)) between 0 and 1 unit of time, the resulting exponential map is e^A. Similarly, using a constant control $u(t) = (\alpha,0,0,0,0)$ between 0 and 1, and a constant control $u(t) = (0,\beta,0,0,0)$ between 1 and 2, the resulting exponential map is $e^{\alpha A}e^{\beta B}$. To make the notation lighter, we denote such a control by $\alpha A \# \beta B$ (we have a concatenation of two controls, one between zero and one, and another one between one and two). Now, $\sqrt{\gamma}A\#\sqrt{\gamma}B\#(-\sqrt{\gamma})B\#(-\sqrt{\gamma})A$ gives almost rise to $e^{\gamma[A,B]}$ by the use of the Baker–Campbell–Hausdorff formula and using the fact that higher-order brackets vanish because of nilpotency. More precisely, we have $e^{\sqrt{\gamma}A}e^{\sqrt{\gamma}B}e^{-\sqrt{\gamma}A}e^{-\sqrt{\gamma}B} = e^{\gamma[A,B]}e^{\rho[A,[A,B]]}e^{-\rho[B,[A,B]]}$, where $\rho = (1/2)\gamma^{(3/2)}$. The control sequence $\alpha A\#\beta B\#\sqrt{\gamma}A\#\sqrt{\gamma}B\#(-\sqrt{\gamma})B\#(-\sqrt{\gamma})A$ therefore gives rise to $e^{\alpha A}e^{\beta B}e^{\gamma[A,B]}e^{\rho[A,[A,B]]}e^{-\rho[B,[A,B]]}$. With some further computations, it is possible to show that the following control $u(t)$, which uses only the controls A and B, steers the system exactly from x_0 to x_1,

$$\begin{aligned} &\alpha A\#\beta B\#\sqrt{\gamma}A\#\sqrt{\gamma}B\#(-\sqrt{\gamma})B\#(-\sqrt{\gamma})A \\ &\#\rho A\#\rho A\#\rho B\# - \rho A\# - \rho B\# - \rho A\#\rho B\#\rho A\# - \rho B\# - \rho A \\ &\#\tau B\#\tau A\#\tau B\# - \tau A\# - \tau B\# - \tau B\#\tau B\#\tau A\# - \tau B\# - \tau A, \end{aligned} \tag{9.16}$$

where $\rho = (\delta - \gamma)^{(1/3)}$ and $\tau = (\delta + \gamma)^{(1/3)}$. The control reduces to 22 steps (four steps cancel out). The following theorem guarantees that the following procedure will work as long as the vector fields $A_1(x), \dots, A_m(x)$ are complete—that is, the solution of the differential equation that they determine exists for all times.

Theorem 9.1 (Lafferiere and Sussman). *If the control-affine system without drift (9.7) is nilpotent of order k and the vector fields $A_1(x), \dots, A_m(x)$ are complete, it is always possible to compute an input $u(t)$ that steers x_0 into x_1 exactly by the above method.*

The method presented here is valid only if the system is nilpotent. This will not be the case in general. A first extension is to the case where the system is *feedback nilpotentizable*, which means that there exists a feedback transformation $u(t) = \sum_{i=1}^{m} b_i(x)w_i(t)$ of the control-affine system without drift (9.7) for which the resulting closed-loop system is nilpotent. In this case, we can repeat the procedure outlined above after the feedback transformation, see [72]. Another extension is in the general case where no nilpotency property holds. In this case, it is possible to develop an iterated version of the algorithm. It is a repetition of the above algorithm used as if the system was nilpotent, but where we do not allow the system to move further than some critical distance within a single iteration. It is possible to show that the procedure gives a control for which the system converges to the final condition, see [72].

9.5.3 Links between the motion-planning problem and approximation by compositions

We can easily see that the problem of approximation by composition, which we discussed in the first part of this chapter, is similar to the motion planning problem. First of all, the solution proposed by Lafferiere and Sussman produces also a sequence of compositions. But we can make the analogy stronger by looking at our maps as points in a function space. We consider, for simplicity, an ODE $\dot{x}(t) = X(x(t))$ whose solution we try to approximate using the solutions of two other differential equations $\dot{x}(t) = A(x(t))$ and $\dot{x}(t) = B(x(t))$. Suppose that the time step Δt we considered for approximation is equal to one. We thus try to approximate an exponential map e^X by a sequence of compositions of the form $e^{a_1 A}e^{b_1 B} \ldots e^{a_n A}e^{b_n B}$. Ideally, the approximation would be exact and we would have $e^X = e^{a_1 A}e^{b_1 B} \ldots e^{a_n A}e^{b_n B}$. Of course, a main difference is that the motion-planning is about steering a single point from some initial condition to some final condition,

$$x_1 = e^X(x_1) = e^{a_1 A}e^{b_1 B} \ldots e^{a_n A}e^{b_n B}(x_0),$$

while the approximation looks at a transformation occurring to all initial conditions taken together. However, the two problems can coincide—at least formally—if we consider invertible maps as points in some function space (here, the group of all invertible maps arising from the sampling of some ODE having a complete vector field). Indeed, we can then rewrite the sequence of compositions as

$$Y_1 = e^X(I) = e^{a_1 A}e^{b_1 B} \ldots e^{a_n A}e^{b_n B}(I),$$

where X, A, and B are vector fields while Y, all the exponentials, and the identity I are maps. We then define a motion-planning problem on this group of

invertible maps as follows: Steer the initial condition I into the final condition $Y(1)$ by driving the system

$$\dot{Y}(t) = u_1(t)A(Y(t)) + u_2(t)B(Y(t))$$

with an appropriate control $u_1(t), u_2(t)$. The expressions $\dot{Y}(t), A(Y(t))$, and $B(Y(t))$ do make sense. Indeed, we can interpret an expression such as $\dot{Y}(t) = A(Y(t)); Y(0) = I$ as having e^{tA} for solution (we are looking at the ODE $\dot{x}(t) = A(x(t))$ but instead of looking at a single trajectory, we consider the evolution of the time-t map for all initial conditions *together*). Applying the method of Lafferiere and Sussman formally, we then obtain a solution of the form $Y_1 = e^{a_1 A} e^{b_1 B} \dots e^{a_n A} e^{b_n B}(I)$, which is precisely the form of our approximation by composition. The main difficulty here is to make the above statements precise because differential geometry in infinite-dimensional spaces is much more difficult than in finite-dimensional ones (and the group of all diffeomorphisms is indeed infinite dimensional). However, in the case where A and B form a nilpotent Lie algebra and Y_1 belongs to $\mathcal{E}_k(A, B)$, the motion-planning problem almost coincides with the approximation problem. Indeed, we know that there exist backward Hall coordinates l for Y_1:

$$Y_1 = e^{l_s A_s} e^{l_{s-1} A_{s-1}} \dots e^{l_1 A_1},$$

where A_1=A, $A_2 = B$, and $A_3, \dots, A_s$ are the other elements of a Hall basis of the nilpotent Lie algebra. Moreover, following the same reasoning as for the motion-planning problem, we know that we can rewrite this sequence of compositions as a sequence involving only e^A and e^B. This reasoning means that, in a nilpotent Lie algebra, we can always build approximations by compositions; moreover, these approximations can be exact. The result is technically identical to the solution of the motion-planning problem but we regard it through a stronger interpretation. Of course, the same argument holds when the nilpotent algebra is generated by more than two vector fields.

9.6 Conclusion

We conclude now the last chapter of this dissertation. We studied here the role that compositions could play in the approximation of dynamical systems. The rationale behind our approach was that standard approximation methods—which use linear combinations of basis functions for approximation—did have the drawback that they could not guarantee to approximate an invertible map by an invertible approximator. An important case where such maps arose was for the discrete-time sampling of continuous-time dynamical systems. We therefore proposed a new approximation scheme based on compositions of invertible transformations: the composition network. We built this architecture

in a fashion similar to the composition methods we had developed in Chapter 8 and within the same framework of Lie algebra theory.

We saw that the composition network could be an efficient grey-box modeling technique in the case where we knew the general form of the system we wanted to approximate but some parameters remained unknown. Comparisons of the new method with MLPs on the problem of identifying a Lorenz attractor gave an advantage to composition networks: The incorporation of a priori knowledge translated into the efficient use smaller training samples and into lower errors at equal sample size. Moreover, the lower complexity required from the composition network resulted into a lower computational burden. Furthermore, composition networks could also treat systems that were sampled at irregular intervals, which is a notoriously difficult problem for MLP architectures. These results were valid for feedforward as well as recurrent prediction of systems with or without inputs. We also proposed a general architecture for black-box modeling called the MLP in dynamics space that combines both MLPs and composition networks, and we showed that such networks could universally approximate dynamical systems.

Finally, we laid a link between approximation by compositions, integration of ODEs, and control theory by reviewing the motion-planning problem. We saw that a solution proposed by Lafferiere and Sussman [72] to the problem of steering the state of a control-affine system without drift (similar to the systems discussed in Chapter 7) from a given initial state into a given final state had a structure similar to the problem of approximation by composition. We first reviewed the notion of nilpotent Lie algebras, Hall bases of a Lie algebra, and of Hall coordinates of an exponential map. We then saw that the solution to the steering problem then amounted to finding a sequence of compositions of exponential maps that steers the initial condition into the desired final condition. Replacing the state of system by some map in the group of invertible maps, we noticed the similarity between the solution of this problem, our composition network, and composition methods for the solution of differential equations.

The composition network and its application to system modeling—as well as the link between composition methods for the approximation of invertible transformations, the solution of differential equations, and the motion-planning problem—form a major contribution of our work.

Chapter 10

General conclusions

The goal of this dissertation was to show that dynamical neural networks and composition networks are versatile models of nonlinear systems. We argued that changes of coordinates are powerful tools for the analysis of neural networks. We also argued that we can model nonlinear systems by sequences of compositions and that Lie algebra theory provides the necessary tools to analyze such compositions. On the whole, we accumulated evidence that neural networks now form a well-structured research area of systems and control theory.

10.1 Contributions

We reviewed the approximation capabilities of feedforward single–hidden-layer perceptrons in Chapter 2. We put a personal emphasis on demystifying the results on the approximation capabilities of neural networks. To achieve this objective, we foregrounded a classical result from analysis (the Weierstrass theorem) in the proof that neural networks have the universal approximation property. Moreover, we carefully discussed the implications of the theorems on approximation and rates of convergence that we presented.

We then moved from static neural networks to dynamical neural networks in Chapter 3. We reviewed the well-known theorems on the convergence and global asymptotic stability of dynamical neural networks. Most of these results relied on finding appropriate Lyapunov functions to prove convergence or stability.

The first main contribution [88] of our work was the study, in Chapter 4, of a representation of neural networks as Lotka–Volterra systems; these systems are simplified quadratic systems that are extensively studied in theoretical biology and dynamical system theory. We showed how to transform different types of neural networks into quasi-monomial forms (which are dynamical systems

where products of powers of the variables play a special role). From there, we could use a known result to transform neural networks into Lotka–Volterra systems.

Moving away from dynamical systems, we added inputs to our dynamical neural networks to enter the realm of control theory, in Chapter 5. We reviewed results on learning in continuous-time neural networks, observability and controllability of neural networks, and on the uniqueness of their weights. We also reviewed control schemes for discrete-time neural networks because practical control systems most often use discrete-time models.

We then studied, in Chapter 6, the capability of dynamical neural networks to approximate dynamical systems, possibly with inputs. Another main contribution [96, 101] of this dissertation was a general theorem on linear changes of coordinates between systems of different dimensions, for a class of generalized neural networks. Our theorem is a significant extension of a previously known result. With this theorem, we revisited results on dynamical system approximation by neural networks.

In Chapter 7, we reviewed the techniques from differential-geometric control and considered their application to neural networks. In particular, we considered the rank conditions for controllability and observability of dynamical neural networks, and the use of Hirschorn's inversion theorem for the computation of inverses of Hopfield-type networks. This chapter appears to be the first thorough and critical review of the application of differential-geometric control to neural networks.

Moving back to dynamical systems, we presented another contribution [90, 94, 98, 100, 102] of this dissertation, which is the introduction of composition methods in the field of neural networks. These methods are efficient for the integration of ODEs and we demonstrated this new approach with two applications. First, we presented a fast implementation of a forwarding control law for the ball and beam. Second, we presented simulations of arrays of Chua's circuits. In our experiments, composition methods appeared significantly faster than traditional methods.

In the last chapter, we developed an original approach [91, 92, 93, 95, 97, 99] to the modeling of nonlinear systems: the composition network. By optimizing the (previously fixed) parameters of the composition methods to match measurements, we obtained a method for grey-box identification. When compared to multilayer perceptrons in the task of predicting the Lorenz attractor, our method required less samples for an accurate identification, was more robust to noise, and was faster to train. Another contribution was the combination of multilayer perceptrons and composition networks for black-box modeling. We proved that the MLP in dynamics space that results from this combination can approximate arbitrary dynamical systems. A final contribution in this chapter

was the discovery of a link between the motion-planning problem, approximation by composition, and composition methods: In the case of nilpotent systems, they share the same formalism.

In the appendix, we presented our contributions [1, 2, 22, 23, 89, 103, 104] to the development of a prototype for the detection of fraud in mobile communications. We contributed to the design of the general architecture of the system and to the design of the features representing the behavior of a user. We proposed the database system and developed the monitoring environment. Finally, we designed the supervised neural network system. Our colleague, Herman Verrelst, did much of the implementation and we acknowledge his work here. Alongside from the presentation of our prototype using the supervised neural network, we also briefly discussed a rule-based prototype and an unsupervised neural network developed by our partners to point out the similarities between the different approaches.

10.2 Suggestions for further research

As we come to the end of this dissertation, we must consider the new possibilities it opened up. The most promising direction of research that emerged from our work is certainly the representation of neural networks as Lotka–Volterra systems. Indeed, this transformation opens the door to all the material available on the analysis of Lotka–Volterra systems, which is more developed than the analysis of neural networks. We expect that we will shortly be able to present new criteria for the convergence and stability of dynamical neural networks. As limited work on the analysis of periodic and chaotic behaviors in Lotka–Volterra systems has already been done [140], we can hope to derive similar results for neural networks.

We saw that changes of variables are an efficient method for studying dynamical neural networks. We presented a transformation that brings a neural network into a Lotka–Volterra form, but it is not clear that this transformation is unique. We may hope for smaller Lotka–Volterra representations than the one we presented here (where the Lotka–Volterra system grows quadratically with the original neural network). It is also not clear that the Lotka–Volterra form is the only possible form for which we can tap into a rich body of existing literature. Therefore, we must further investigate the role of changes of coordinates in neural networks not limiting ourselves to a particular kind of dynamical neural network or equivalent system.

The link between the use of compositions for the approximation of dynamical systems and the motion-planning problem also opened a new path for research. It appears that nilpotency is an assumption that makes the analysis of such systems much more tractable. In our work, we make no such assumption to achieve greater generality; but the Lie algebras that we consider are infinite

dimensional, implying that few theoretical results are available. We expect that a much sharper analysis of approximation by composition will be possible under the assumption of nilpotency. Furthermore, this assumption opens the way to a fruitful exchange between the domain of motion planning and the domain of composition methods. We think that it is possible to design powerful composition methods under the assumption of nilpotency. It may even be that the notion of feedback nilpotentizability, briefly introduced in Section 9.5.3, is useful for this problem.

Our most general object of interest is the use of one-to-one transformations in system modeling. We have certainly not covered all their possible applications in our work. The most important problem remaining is to find simple parameterizations for nonlinear manifolds. The difficulty is that such parameterizations must be easy to manipulate, so that we can use them in optimization procedures, while they must be sufficiently flexible to allow varying topologies. A typical application would be the determination of the manifold of the states of a nonlinear system in the space of delay vectors. We can consider the use of compositions for such parameterizations, but we should also consider other architectures. Another application of compositions that we will investigate is their use for the parameterization of Lyapunov functions that guarantee global asymptotic stability. Indeed, we can interpret such a sequence of compositions as a nonlinear change of coordinates and—by taking the norm of the resulting function—we obtain a Lyapunov function equivalent to a quadratic Lyapunov function by our change of coordinates. The use of a sequence of compositions gives us much more flexibility than quadratic Lyapunov functions while still guaranteeing global asymptotic stability.

For the detection of fraud, the next step will be the integration of the three prototypes into a single, more powerful tool. After this integration, the use of this new prototype in daily operations will allow the collection of more reliable data for the detection of fraud. This data of better quality will permit the redesign of the supervised neural network. We expect that the resulting neural network will achieve superior performance.

Appendix A

Detection of mobile phone fraud using neural networks

During our doctoral work, we had the chance to work on a practical application of neural networks—namely, the detection of fraud in mobile communications. The European Community is accelerating the deployment of advanced communications infrastructures and services through its Advanced Communications Technologies and Services (ACTS) program. In the framework of this program, the Advanced Security for Personal Communication Technologies (ASPeCT) project addresses the issues of security in mobile communications in a future mobile communication system, and during the migration from the current digital system. Within this project, a workpackage is dedicated to the detection and management of fraud in mobile communication networks using advanced ("intelligent") decision tools. We report here on the work the different partners of the ASPeCT project performed to develop three prototypes of fraud detection systems. Two mobile phone companies—Vodafone, England, and Panafon, Greece—provided their expertise on the telecommunication industry and, in particular, on the technical aspects of fraud. They also provided the data necessary to build the prototypes. Three research groups—the Department of Computer Science at Royal Holloway University of London, the Computer Security Group at Siemens Corporate Research, Münich, and the Department of Electrical Engineering at the Katholieke Universiteit Leuven—developed the fraud detection systems. In particular, we contributed by developing a supervised neural network that performs the classification of calling patterns as fraudulent or nonfraudulent. The two other prototypes are a rule-based system, developed by Siemens and an unsupervised neural network, developed by Royal Holloway. We review these prototypes in this chapter.

The mobile communication industry suffers major losses from fraud. In the United States alone, the industry estimates its loss of revenue to over 650

million dollars a year. Part of this loss of revenue is virtual, as people who make fraudulent calls would not make these calls if they had to pay regular fees. Yet, a significant part of the loss of revenue corresponds to an actual loss, because many fraudulent calls are made to destinations outside the network of the mobile phone operator (which means that the operator has to pay charges to another phone operator when a fraudulent call is made). Moreover, the operator has to dimension its infrastructure to account for these supplementary calls (in the most fraudulent areas, such as Manhattan, NY, up to 25% of the calls have been known to be fraudulent). Because of the direct financial consequences of fraud for mobile phone operators, the prevention and detection of fraud is a priority for the industry.

A.1 Fraud scenarios and fraud indicators

The operation of a mobile network is complex, and fraudsters invest a lot of energy to find and exploit every weakness of the system. The first step of our developments is thus to analyze the operation of the network, and identify and categorize the vast range of possible scenarios of fraud. Several parties are involved in the fraud chain:

- The network operator, which provides the infrastructure for mobile communications.
- The service provider, which is responsible for the management of the customers. It buys access time on the network of the operator and is responsible for all charges incurred by its customers. In some cases, the network operator is at the same time the service provider.
- The mobile phone user, who suffers major inconveniences as a result of fraud, as he has to deny that he placed the fraudulent calls and as he needs to negotiate the cancellation of the charges resulting from fraud.
- The fraudster, who gets access to the network and usually resells this access at a discounted rate to an unscrupulous customer.
- The unscrupulous customer, who has two main advantages when using the fraudulent service. First, the fraudulent service is much cheaper than the regular service, which is particularly attractive for foreigners calling their families at home. Second, the fraudulent service is anonymous, which is important for criminals who do not want the police to trace their phones.

Fraudsters often organize their operations on a fairly large scale with multiple fraudulent lines at once. This scheme is often dubbed a *call-selling bureau*.

There are many ways for a fraudster to unlawfully access a mobile phone network. To discuss these different fraud scenarios, we first need to note that the mobile phone technology available to customers varies widely from operator to operator and from country to country, and that the technology used on a mobile phone network has a direct influence on the type of fraud to which it is vulnerable. The easiest categorization of the different standards is to divide them into generations.

The first generation is the group of analogue mobile phones. This early technology provides poor security features. The major weakness lies in the identification procedure. Each time a user places a call, his mobile phone reaches the network through the closest cellular switch and identifies itself to gain access to the network. This identification takes place through a unique identification code sent over the air waves. Bandits can intercept such codes with a radio–scanner and plug them into reprogrammed phones. They can then access the mobile network with the identity of the unsuspecting user and start charging call to this user's account. This kind of fraud is called *cloning fraud* and it is a severe problem for analogue phones because the hardware necessary to commit this type of fraud is readily available.

The second generation of phones are digital systems, such as the global system for mobile communications (GSM). These systems provide enhanced security features through an *authentication algorithm*. When a user places a call, the phone again contacts the closest switch but, instead of sending its private identification code directly, it only claims its identity. The switch then sends a random number back to the phone, which in turn encrypts this random number using its private identification key and an encryption algorithm. The phone then sends only the encrypted number back to the switch. The identification code is also available to the computer at the switch; the computer performs the same calculation as the phone and checks that the answer provided by the phone is correct. If the answer is correct, which proves that the phone knew the private key and was indeed who it claimed it was, the switch lets the user place a call. Using this challenge-and-response procedure, the phone never needs to send its private key over the air waves. The fraudster can only intercept question–answer pairs. Deducing the private key from a sequence of questions and answers is difficult because it amounts to breaking an encryption algorithm. The computing power necessary to break these algorithms still puts this approach out of daily fraud practices, although many experts think that this situation might change in the near future.

Authentication is not in essence limited to second-generation phones, but the cost of implementing it on first-generation networks and the need for the users to purchase authentication-capable phones, makes operators prefer to look for alternatives to authentication for first-generation systems. When first and second-generation systems coexist (such as in the United Kingdom

and in the United States), the main target of fraudsters remains the first-generation phones as the fraudsters always attack the most vulnerable prey. However, as the industry migrates further into second-generation and future third-generation systems (high-bandwidth digital systems, such as the Universal Mobile Telephony System (UMTS)), the industry expects fraud to follow this migration.

Moreover, authentication is not enough to prevent fraud. One simple technique to defeat authentication is *subscription fraud*. The fraudster subscribes to the services of a network under a false identity, possibly with the help of unscrupulous personnel at the service-provider level to bypass the security checks. The fraudster accumulates a large number of expensive calls, but vanishes before it is possible to collect the bill. Another important type of fraud is *private automatic branch exchange (PABX) fraud*. Large companies have a computerized management of their phone lines through a PABX. These systems offer the possibility to call them and place an outside call through their fixed lines by using a special identification code. Fraudsters use mobile phones to call the PABX repeatedly while automatically trying random codes until they can guess the right one. Although the victim is not the operator nor one of the operator's customers, PABX fraud remains high on the agenda of network operators. We have only given here a sample of fraud strategies. Many more are possible, especially in the case where operator personnel is involved.

After identifying the possible fraud scenarios, we look at the possible indicators of fraud that we can extract from the information available on the network. These are some the fraud indicators. An excessive number of international calls to countries from different continents is indicative of a call-selling bureau. These calls also take place one right after the other ("back-to-back"), so short intervals between calls is indicative of fraud. Fraudsters set up these call-selling bureaus mostly in shabby neighborhoods of large cities, so repeated calls from some "hot cells" are also indicative of a call-selling bureau. Unscrupulous users call some numbers often (for example, sex lines), so calls to some "hot numbers" are an indicator. For PABX fraud, the attack takes place by repeatedly calling the PABX shortly to try different access codes, so a high number of short back-to-back calls is indicative of PABX fraud. The task of the fraud detection system is to compute parameters representative of these behaviors, weight these parameters appropriately, and produce an alarm if the suspicious activity rises above a given threshold.

The information about the activity on the network is encoded in the toll tickets of all the calls placed on the network. A toll ticket is a bill issued by the network after each call, which contains all relevant information about that call. The information we use is

1. The International Mobile Subscriber Identity (IMSI), which identifies a user uniquely

2. The starting date of the call
3. The starting time of the call
4. The duration of the call
5. The number that was called (called the B-number)
6. The type of call (national, international, or supplementary services)

More information is available from the toll ticket, but we consider only these six characteristics at this point. The limited complexity of the system comes from the constraint that it must process massive amounts of information: A network operator can have over a million subscribers, placing a few million calls a day. A major requirement is that the system must be able to analyze up to 50 calls per second.

A.2 Absolute and differential analyses

A first approach to fraud detection is to detect excessive activity directly. For example, we might want to raise an alarm if a user is making more than two hours of calls from Europe to South America on any single day. We call this approach *absolute analysis*, because the fraud detection system raises alarms on the basis of activity, independently of the user who is placing the calls. With this approach, users with a high level of activity raise alarms, since high activity is characteristic of fraud. However, if these users happen not to be fraudsters but legitimate users, they are in fact the best customers of the operator, as they spend the most. Hence, the operator wants to avoid troubling these legitimate high-end users. To prevent this problem, it is necessary to estimate what the normal activity of any given user is. The system must raise alarms on the basis of a sudden departure of the behavior of a user from his normal behavior. We call this strategy *differential analysis*. Note that it is not possible to build a whole system on the basis of differential analysis only. Indeed, in the case of subscription fraud, the mobile phone is always used by the fraudster, so there is never any departure from a normal behavior and any differential analysis would only learn to recognize the fraudulent behavior as the usual one. The fraud detection system will therefore have to combine these two types of analysis to produce its alarms.

It appears that the use of differential analysis for the detection of fraud is new [1, 36]. Many of the currently used systems use a combination of the following two methods. First, a set of simple rules raises an alarm when the absolute activity of the user surpasses a certain threshold. Second, the system matches the calls of a user against one another to detect *collisions* or *velocity violations*. A collision happens when two calls of a single user take place at

the same time. A velocity violation results when two calls are placed shortly after one another from two distant places. Experience shows that the tuning of velocity-based systems requires much care because of synchronization problems between the clocks of different switches (resulting in false collisions) and because of radio-wave propagation (in some landscapes and weather conditions, particular radio-wave–propagation behaviors can cause a phone to connect to a far away switch instead of to the closest one).

A.3 Architecture of the fraud detection tools

We describe now the general architecture that implements the absolute and differential analyses. This architecture is common to the three prototypes we describe in Sections A.6 to A.9. We keep this framework common on purpose, so that an integration of the three prototypes into a single system, allying the advantages of each prototype, remains possible.

We build a large database where we store, *for each user*, a set of relevant parameters (called *user profile*) representative of his behavior. Because of the high number of users stored in the database (250,000 in most of our experiments, over 1,000,000 in an industrial environment) and because of the resulting requirement of over 50 database updates per second, conventional transactional databases cannot achieve the required speed. We therefore use a simple database indexed only by user. It is thus impossible to formulate queries, such as requesting directly from the database the identity of all the users who made more than two hours of international calls during the previous day. We use the GNU gdbm database driven by C or C++ calls and achieve the required speed without difficulty.

When a user finishes a call, the network produces a toll ticket, which it transmits to the system responsible for billing the customer. This transmission can happen in real time or the switch can transmit all its toll tickets in one batch during the night. Most operators are now moving to real time billing; we therefore develop our prototypes under the assumption that we have to handle the toll tickets in real time and in a continuous flow. We thus assume that the fraud detection system receives a copy of the toll ticket produced by the switch. The fraud detection tool then retrieves the profile of the user who made the call from the database. Using the information contained in the toll ticket (in our case, the six fields we mentioned in the previous section), the system updates the parameters representative of the behavior of the user. The system passes these parameters to the decision engine that checks if the activity of the user warrants an alarm. It also stores the updated user profile back into the database.

We implement the differential analysis by splitting the user profile into a short-term (*current user profile*) and a long-term (*user profile history*) representation of the user's behavior; and we base the analysis on a comparison of the two. We perform the absolute analysis by treating the current user profile without reference to the user profile history. Figure A.1 depicts the general architecture of the system. The decision engine transmits its alarms to a monitoring tool that gathers relevant information about suspicious users and transmits this information to a fraud analyst who decides whether or not to take action against the suspicious user—for example, by calling him to check his identity or by disconnecting him from the network.

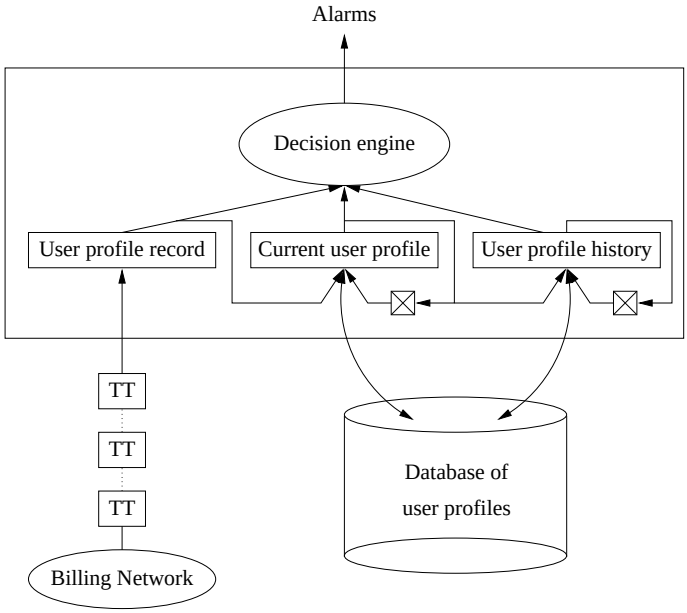


Figure A.1: Architecture of the fraud detection tool. The tool handles the toll tickets from different users. It swaps the profile of the users to and from the database of user profiles at each toll ticket. It represents the information about the toll ticket in the user profile record. The current user profile represents the short-term behavior of a user while the user profile history represents the long-term behavior of a user. The tool updates the value of the current user profile with the user profile record and the value of the user profile history with the current user profile. The decision engine uses all this information to produce alarms.

A.4 User monitoring

We must stress that the fraud detection system does not decide whether a user is fraudulent or not; it only suggests suspicious users and leaves the ultimate decision to the fraud analyst. The reasons for this attitude are (1) the financial stakes involved with misclassifying a high-end user as a fraudster, (2) the risk that the fraud detection system could run amok and shut down all users, (3) the ability of the fraud analyst to perform other checks, and (4) the higher accuracy of the decision of the fraud analyst. What the fraud analyst cannot do is process the data of all the users manually, but given all the information about a suspicious user his decision will be more reliable than that of any fraud engine (be it only because he can try to contact the user). This situation will have an influence on the performance criteria we use to evaluate our system, which we discuss in Section A.7.

The function of the monitoring tool is to collect information about suspicious users in a way that is most helpful and easiest to handle for the analyst. In our system, the monitoring tool compiles a daily list of all the users who raised an alarm on that day. It also gathers the toll tickets of all the calls made by a suspicious user during the two days before he first raised an alarm and all the calls made until he raises no alarm for three days. Collecting all the calls made by a suspicious user from the moment he raises an alarm presents no difficulty. However, the production of an alarm implies that the recent accumulated activity (current user profile) is abnormal (absolutely or with respect to the user profile history); this accumulation means that the suspicious activity has been taking place for some time before the system raised an alarm. It is therefore of importance to the analyst to know what the calls of the suspicious user were before he raised an alarm.

Because the system cannot anticipate which users will raise an alarm, so we need to keep a list of all the calls made by all users and search this list when a user raises an alarm. We cannot keep such a list for long because of the amounts of data the system processes (just under 100 megabytes a day in our experiments). Moreover, searching such long lists of calls for all the calls of a single user is time consuming. We therefore adopt the following strategy. We build a list of all the calls made on a single day but keep this list sorted by user. (We implement the sorting procedure incrementally by collecting small batches of calls, sorting them, and merging the sorted batch with the already sorted list; proceeding this way, we can keep the computational burden of the sorting procedure under control.) We keep such a list for the current day and the day before; at midnight, we trash the list of the day before, move the current list to replace the old list, and start a new list for the new day. When a user raises an alarm, we can go through these two sorted list and collect all the calls of the last two days. We also perform the same procedure for the profiles of the suspicious users. The monitoring tool makes this list of suspicious users

and the list of calls and profiles of these users available to the analyst. The analyst can then request the calls and profiles of a specific suspicious user and also visualize this information graphically. This information is also kept as evidence in an audit trail to support later prosecution, if necessary.

Legal aspects of fraud detection are an important problem. It is not clear what value a court would give to the information gathered by the fraud detection tool, as computer records carry much less weight in court than written records. Also, serious privacy issues arise when processing data pertaining to personal communications. In Europe, the personal-data–protection directive and the data-protection directive of the European community offer many provisions against abusive processing of personal information. In our experiments, the network operators encrypted part of the IMSI of the user placing the call and the number called by the user to protect the privacy of their customers. However, we must note that this encryption is not, in itself, enough to satisfy the provisions of the European directives as their definition of personal data encompass all data about an identified or identifiable person (the user is still identifiable after encryption since someone possesses the key necessary for decryption). It appears that processing call data for fraud detection is probably acceptable as long as (1) the data is used only for this purpose, (2) the system uses only the minimum information necessary for this task and does not keep the data longer than necessary, and (3) the service provider properly informs its customers about this practice.

A.5 Description of the user profiles for the supervised and the rule-based systems

The profiling performed by the supervised neural network and the rule-based system is similar. We therefore discuss it in this section while we leave the profiling of the unsupervised neural network for the next section. The main part of the work went into designing a set of parameters or features that adequately captures the behavior of a user. Because the supervised network and rule-based system are static classifiers (not having any internal states containing information about the previous calls of the user), the features themselves contain this information about the previous calls. Alternatively, we can view—for each user—the decision engine together with the current user profile and the user profile history as a dynamic system whose state is driven by the sequence of toll tickets.

The features contain short-term estimates (for the current user profile) and long-term estimates (for the user profile history). The system estimates the means and standard deviations of the duration and number of calls or interval between calls. It computes these estimates both for national and international calls (we leave supplementary services, such as calls to a voice mail, aside

because this information is not available for the examples of fraud we have). The total number of features is thus $(2 \times 2 \times 2 \times 2 =)16$. These features are updated for each user after he makes a call. It is important to update and manage these feature profiles for all users in real time. This requirement imposes severe constraints on the design of the features.

Two variants of the features are possible: instantaneous features and daily features. For the instantaneous features, we try to compute, for example, the average duration of the national calls of a user both on a short-term and on a long-term basis. We can estimate this quantity through a simple and efficient filtering strategy: We use a bank of first-order filters to compute a short-term mean, and feed the means in a second bank of first-order filters to produce a long-term mean. If we denote the duration of the kth national call of a user by $d_{\mathrm{n}}(k)$, we can estimate the mean duration μ_{n} of the national calls by the short-term estimate $m_{\mathrm{n}}(k)$ and the long-term estimate $n_{\mathrm{n}}(k)$ as follows:

$$\begin{aligned} m_{\mathrm{n}}(k) &= \alpha m_{\mathrm{n}}(k-1) + (1-\alpha) d_{\mathrm{n}}(k), \\ n_{\mathrm{n}}(k) &= \beta n_{\mathrm{n}}(k-1) + (1-\beta) m_{\mathrm{n}}(k), \end{aligned}$$

where we choose the parameters $0 < \alpha < 1$ and $0 < \beta < 1$ appropriately. We note that $m_{\mathrm{n}}(k)$ is a first-order low-pass filter on the call duration that favors the continuous component of the signal (which is its mean) over the higher frequencies. Furthermore, $n_{\mathrm{n}}(k)$ is a second-order low-pass filter that is sharper in the frequency domain than the first-order filter and is therefore appropriate to estimate a long-term mean. Finally, we note that the difference between these two filters $m_{\mathrm{n}}(k) - n_{\mathrm{n}}(k)$ is a band-pass filter that filters out the lower frequencies (the long-term natural evolution of the behavior of a user) and the higher frequencies (the differences between a few successive calls) to focus on the middle frequencies (which indicate a significant change of behavior on the scale of a few tens of calls).

We can also estimate the standard deviation of the duration of the calls by filtering the square of the duration of the calls. We can estimate the standard deviation of the duration σ_{n} of national calls with the short-term estimate $s_{\mathrm{n}}(k)$ and the long-term estimate $r_{\mathrm{n}}(k)$. Assuming that the duration of the calls is an independent and identically-distributed random variable, and using the relations

$$\begin{aligned} \mu &= E(d), \\ \sigma^2 &= E((d-\mu^2)) = E(d^2) - \mu^2, \end{aligned}$$

we can compute the estimates with the following filters:

$$\begin{aligned} m_{\mathrm{n}}(k) &= \alpha m_{\mathrm{n}}(k-1) + (1-\alpha) d_{\mathrm{n}}(k), \\ p_{\mathrm{n}}(k) &= \alpha p_{\mathrm{n}}(k-1) + (1-\alpha) d_{\mathrm{n}}^2(k), \\ s_{\mathrm{n}}(k) &= \sqrt{p_{\mathrm{n}}(k) - m_{\mathrm{n}}^2(k)}, \\ r_{\mathrm{n}}(k) &= \beta r_{\mathrm{n}}(k-1) + (1-\beta) s_{\mathrm{n}}(k). \end{aligned}$$

We can perform a similar procedure to compute the frequency of the calls. We apply then the same filtering technique to the inverse of the interval between two successive calls. A problem here is that calls with a small interval produce a large inverse that tends to destabilize the filter (we can remedy to this difficulty by biasing the signal towards its mean). Another difficulty is to choose the parameters of the filter correctly. Two philosophies conflict here. We may want the filters to adapt naturally as a function of the number of calls placed by the user; or we may want to set the parameters of the filters according to the call frequency of this user so that the estimates adapt as a function of the time span elapsed (independently of the number of calls). Filters that adapt over a fixed time span make it easier for the analyst to interpret the difference between short-term and long-term behaviors, but they are more difficult to implement.

The other variant for the estimation of the features are daily estimates. Here, we estimate quantities such as the average of the total duration of all the national calls of a user on a day and the average number of a user's national calls. All we do is add up the duration of all the calls of a user in the first case and count each new call in the other. We use a sequence of two or three of these daily values as the short-term estimates and filter these daily values to obtain long-term estimates. A first problem with this approach is that the estimate for today is incomplete until midnight, so its comparison with yesterday's value can be misleading. Another problem is that we reset the estimate at midnight, which means that the sequence of feature vectors follows a discontinuous path (this resetting is a problem if the user's behavior is approaching the alarm boundary, because the resetting pushes the behavior away from the boundary and the behavior will need some time to return and produce an alarm). These difficulties imply that these features react more slowly to fraud than instantaneous features. Both choices of features thus have drawbacks. In practice, it appears that daily features are somewhat easier to handle both for the decision engine and the fraud analyst, even if they are a bit more difficult to program.

A.6 Supervised neural network for fraud detection

Fraud detection systems using neural networks are currently being developed worldwide for a variety of applications [44, 35], among which fraud detection in mobile communications [27, 14], but literature on the subject is scarce. Because we put much care in the design of the user profiles, the neural network can now solve a well-defined classification problem. The essential property of neural networks here is that they are nonlinear classifiers. The user profile (current user profile plus user profile history) that we compute after each call is nothing more than a vector in a 16-dimensional vector space. We have two classes:

profiles that corresponds to fraud events and profiles that correspond to normal behavior. As neural networks are nonlinear classifiers they can implement a boundary between these two classes that is a curved surface (or even the union of such surfaces) instead of a hyperplane. This property is important because the fraudulent behaviors are not confined to a single region of the parameter space (for example, call-selling fraud may appear through repeated long international calls while PABX fraud may appear through much higher frequencies of short calls).

The collection of data is a difficult problem as the design of the business processes in the mobile communication industry (for example, split responsibility between the service provider and network operator) conflict with an efficient collection of the data. The data we use for training consists of a list of toll tickets produced by 300 new users of a GSM network over a six-week period, and of 104 cases of fraud from a first-generation network over a period of six months. For all these users, we clean up all the toll tickets to extract the six fields of information necessary to produce sequences of user profiles. The training and evaluation of the prototype are difficult tasks because of the messy nature of the data. There are numerous problem with the data:

- Different users make vastly different number of calls.
- The call histories cover different periods of time.
- There is no guarantee that none of the new users are not fraudsters.
- There is no guarantee that the call history of fraudsters contains enough information for detection using an analysis of call duration and call frequency patterns.
- We do not know, for a case of fraud, when the fraud exactly started and when it stopped.

For these reasons, we must select a subset of the user profiles for which the information is qualitatively adequate and we must label them manually into suspicious and nonsuspicious. The features we use are the 16-dimensional instantaneous features described in Section A.5. The architecture of the neural network we developed is a multilayer perceptron with 16 inputs, a single hidden layer of 5 neurons with the hyperbolic tangent as the activation function, and a linear output unit. The training is performed on half of the data, using the conjugate-gradient algorithm.

A.7 Evaluation of the supervised neural network

For testing, we used data consisting of the activity of 1974 new users during a two-month period, and of the call history of another 104 fraudsters. The testing

data was collected so as to minimize the problems mentioned above, but they are far from being completely alleviated. Squared error criteria on the test set containing both fraudulent and new users are essentially meaningless because the proportion of fraudulent cases to normal cases is not representative of the true proportion. This proportion is in fact unknown. We therefore settled for criteria based on percentage of "correct" classification (fraudsters classified as suspicious and new users classified as nonsuspicious) on the two classes separately. Further, if we want the system to be used in practice, the analyst needs to in a position to control the number of alarms generated by the system. Another constraint is that the ratio of correct detections to false alarms needs to be high enough, so that the analyst takes the alarms seriously. We deal with the first constraint by using a variable alarm threshold. We then compute the percentage of fraudsters detected by the system, and the percentage of new users who raise an alarm. We can represent these proportions through a diagram called a receiver operating characteristic (ROC). We can see such a curve on Figure A.8. To give an idea of the performance, it is possible to classify 90% of the fraudsters correctly, while misclassifying 10% of the new users. Such a performance, while being of interest, is unusable in practice because it would generate a flood of false alarms. This problem stems from the fact that a network contains hundreds of thousands of users. However, if we accept to detect only 50% of the fraudsters, we misclassify only 0.3% of the new users. Although we cannot answer the question of what the ratio between correct detection and false alarms is, since we do not know the exact proportion of fraud on the network, we expect that such a percentage of misclassification can keep the number of alarms under control.

A.8 Rule-based system

The rule-based system is a modification of Siemens's protocol-data analysis (PDAT) tool. The Computer Security Group previously developed this system for the detection of intrusions on network of UNIX workstations. Its architecture consists of a number of analyzing units, which perform data analysis on-line according to some set of rules, and of a controlling unit, which centralizes the results and manages the analyzing units. An analyst can manage rules and visualize alarms via a graphical user interface. The programming of the rules takes place through a special protocol-data analysis language.

The main modification that is necessary to adapt PDAT from intrusion detection to fraud detection is the development of a front end (which is part of the analyzing unit) that produces the features representative of a user's behavior and manages the database of user profiles. The rule-based system uses the daily features presented in Section A.5. A major advantage of the PDAT system is that the analyst can add or remove rules and see these rules

take effect immediately. The controlling unit transfers the rules to the analyzing unit and collects all the alarms from it. The controlling unit then displays the alarms on the graphical user interface.

The system implements three types of rules based on total call duration and number of calls: absolute analysis, differential analysis, and a combination of the two that uses lower thresholds. Each rule has a different threshold. To use ROC curves, it is necessary to have only one alarm threshold. To achieve a unique threshold, we normalize the activity of each rule by its individual threshold. We use as a measure of activity the maximum of all the normalized activities. It is this meta-rule that we use to raise an alarm. When the threshold is set to one, this procedure is equivalent to the original behavior of raising an alarm if any of the rules rises above its threshold. When we adapt the threshold, the procedure is equivalent to adapting all the individual thresholds in the same proportion.

After tuning the individual thresholds manually, the rule-based system achieves excellent performance. The data used for development and testing is similar to the data used by the supervised neural network. The system can achieve 95% of correct detection for 5% misclassification, which is again impractical because it would produce too many false alarms. But at 0.1% misclassification, it achieves 73.5% detection. Figure A.8 presents these results. The different curves correspond to different time constants of the filters that produce the long-term estimates. Relatively fast adaptation ($\beta = 0.96$) led to the best results.

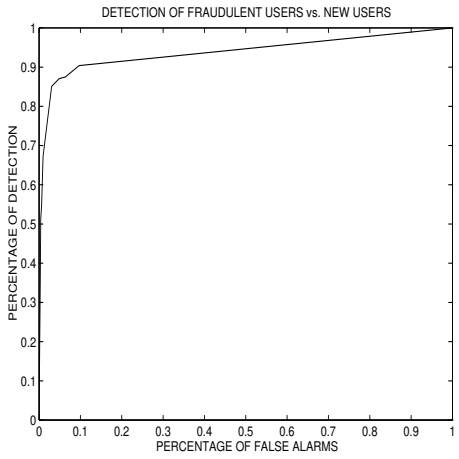


Figure A.2: Performance of the supervised neural network. The receiver operating characteristic displays the tradeoff between correct detection (y axis) and false alarms (x axis). The further the ROC curve reaches toward the upper left corner, the better the general performance of the system.

A.9 Unsupervised neural network

The last of the prototypes is the unsupervised neural network developed by the Department of Computer Science at Royal Holloway University of London. It adopts a strategy different from the preceding two prototypes. The idea is to represent the behavior of a user by a probability distribution. An example would be the distribution of the time of the call of a user. We assume that this probability distribution is representative of the user and that we should associate a completely different probability distribution to another user. For example, business users have a probability distribution highly concentrated into business hours while private customers tend to call in the evenings. We therefore build short-term and long-term estimates of such probability distributions. If the difference between the short-term and the long-term behavior becomes too high, the user must have changed his behavior in some way.

The strategy adopted here does not focus directly on detecting fraud but more on detecting unusual changes of behavior. We can expect that this approach is somewhat less sharp than the previous ones, but the main objective here is to obtain a system that detects fraud patterns for which the other systems have not been designed. Indeed, through their mechanism of computing features representatives of a user's behavior and tuning their parameters, the previous systems optimize their behavior to detect the existing fraud patterns. But fraudsters move to new fraud strategies as soon as their old strategies get defeated. There is thus a definite need for some form of monitoring of new abnormal patterns. This flexibility is the goal of the unsupervised neural network.

We do not limit ourselves to one-dimensional probability distributions. The recursive estimation of a multidimensional probability distribution is a difficult problem. The method we adopt here is to discretize such a probability distribution. For a multidimensional distribution, a discretization at points on a regular grid is inefficient because many of the bins created in such a way may remain empty. An optimal representation requires to discretize irregularly or, more precisely, to quantize the random variable in a way that maximizes the entropy of the discrete distribution. Put simply, the best discretization is the one where each bin has a zone of influence such that the probabilities of all bins are equal. We do not present a justification of this fact here and only state that there exist algorithms that, starting from a random distribution of the bins (prototypes), evolve towards an optimal distribution of the prototypes. Figure A.9 presents such an optimal distribution of 50 prototypes for the two-dimensional probability distribution of the starting time of a national call versus its duration. We plot the individual calls on the left while we plot the prototypes on the right. We can visualize the continuous probability distribution as the limit of the density of points on the diagram on the left. We can visualize the discretization of this distribution as a discrete distribution

over the 50 prototypes. We associate any given call on the left to its closest prototype. The spatial distribution of prototypes is such that their individual region of influence are equiprobable.

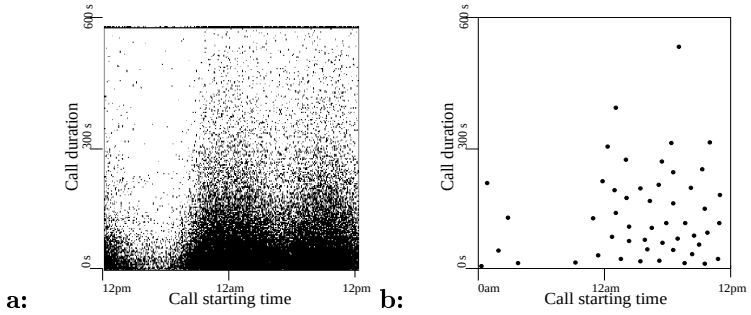


Figure A.3: Prototypes of the unsupervised neural network. **a:** Probability distribution of the calls. The dots give the distribution of the time of the call (x axis) versus the duration of the call (y axis). **b:** Distribution of the prototypes. These prototypes give a discretization of the probability distribution of the calls. All the prototypes cover a surface that is approximately equiprobable.

The unsupervised neural network uses this strategy for the distribution of call starting time versus call duration for national calls, international calls, and supplementary services. It maintains distributions on these three sets of 50 discrete variables using the same filtering strategy as the other prototypes. Indeed, if we represent each incoming national call by a 50-dimensional vector whose components are all zero except the one corresponding to the prototype to which we associate the incoming call, the average of such vectors is equal to the probability distribution over its components. As before, we obtain the short-term averages through first-order filters and the long-term averages by feeding the short-term averages through a second bank of first-order filters. We then need a criterion to measure the difference between the short-term and long-term probability distributions. The unsupervised neural network employs the Hellinger distance to measure this difference. We compute the Hellinger distance between two probability distributions $p(i)$ and $q(i)$ with $i = 1, \ldots, k$, where k is the number of values that the discrete distribution takes (50 in our case), as follows:

$$d = \sum_{i=1}^{k} \left(\sqrt{p(i)} - \sqrt{q_i}\right)^2.$$

We find the threshold to use for fraud detection by running the unsupervised neural network through the data of normal users. We use the maximal value observed in this data as our reference threshold. We vary the actual threshold

around this reference value to estimate the tradeoff between detection of fraud and false alarms. The unsupervised system reports its performance using the same ROC curve as before, with a typical tradeoff of 40% detection for 1% false alarm, as we can see on Figure A.9.

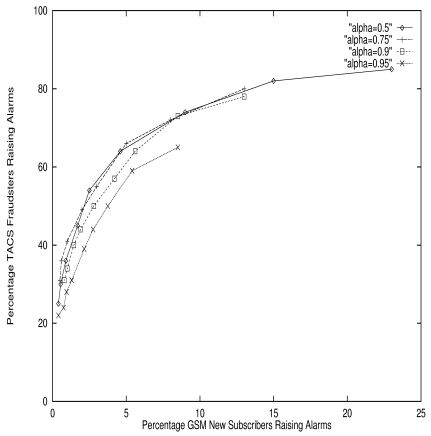


Figure A.4: Performance of the unsupervised neural network. The receiver operating characteristic displays the tradeoff between correct detection (y axis) and false alarms (x axis). The different curves give the trade-off for different values of the parameters of the filters.

A.10 Conclusion

The unsupervised neural network does not perform as well as the supervised neural network. But the rule-based system reports a typical tradeoff of 70% detection for 0.1% false alarm, which is slightly better than our tool. The reason behind this difference is that the data about fraudulent users is limited and messy, making learning difficult. But each tool has real qualities. The rule-based system is simple and efficient. In a production environment, we can use the rule-based system to collect high-quality data that makes the training of the supervised network efficient and enables the supervised network to surpass the rule-based system. In the case where the fraud strategies evolve over time—which is definitely the case if a good fraud detection system forces fraudsters to change their strategies—both the rule-based and supervised network cannot adapt without retuning or retraining. We can use the unsupervised neural network either to handle these new scenarios or at least to detect such shifts and signal the need for a redesign of the other systems.

As mentioned in Section A.6, literature on fraud detection is scarce. In [14], Barson and his colleagues report a performance of 92.5% detection of fraudulent users and 92.5% correct classification of normal users for a neural network prototype, on some synthetic data they had produced. Because 7.5% of false alarms is totally unacceptable, this report is not usable. However, Foster and Provost [36] recently described work on an adaptive rule-based system, which is of the highest quality. (Their report is actually posterior to our work within ASPeCT.) Their work deals with cloning fraud for analogue phones but it shares the same spirit as ours. We can interpret their method as a mix between our rule-based system and our supervised neural network. Foster and Provost generate potential rules automatically by looking for common patterns in the data (for example, a high proportion of the cases of fraud contain calls during the night). Their rules use differential analysis as our systems do. They then train a linear classifier on these rules. We can reinterpret the resulting architecture—although they obtain it in a completely different way—as a single–hidden-layer perceptron with step nonlinearities working on the same type of features as our rule-based system. Although it is not possible to compare the performance of their system with ours because of the difference in data and in experimental methodology, Foster and Provost report comparisons with other existing systems that show that their system is the state of the art. Their three main conclusions are that an efficient fraud detection system must (1) perform differential analysis, (2) optimize its working by training, and (3) adapt to new patterns automatically. Our systems address these problems: all perform differential analysis, the supervised neural network is optimized by training, and the unsupervised neural network adapts automatically.

By way of conclusion, this Chapter presented our work on the detection of fraud in mobile telecommunication networks. We presented the three prototypes developed in the framework of the ASPeCT project. The first, and most efficient prototype at this point, used a rule-based system. The second used a supervised neural network that we can optimize automatically, as long as enough relevant data is available. The third used an unsupervised neural network that monitors abnormal changes in the probability distribution of the activity of a user and offers the flexibility necessary to detect new fraud patterns. The performance of these systems brings them within range of state-of-the-art systems.

Bibliography

[1] ASPeCT ACTS AC095, "Definition of fraud detection concepts," ASPeCT AC095/ATEA/W22/DS/P/06/A, ACTS Programme of the European Community, 1996.

[2] ASPeCT ACTS AC095, "Fraud management tools: First prototypes," ASPeCT AC095/DOC/RUL/072/WP22/A, ACTS Programme of the European Community, 1997.

[3] R. A. Adams, *Sobolev spaces*. Academic Press, 1975.

[4] D. Aeyels, "Generic observability of differentiable systems," *SIAM Journal of Control and Optimization*, vol. 19, pp. 595–603, 1981.

[5] F. Albertini and P. Dai Pra, "Forward accessibility of recurrent neural networks," *IEEE Transactions on Automatic Control*, vol. 40, no. 11, pp. 1962–1968, 1995.

[6] F. Albertini and E. D. Sontag, "For neural networks, function determines form," *Neural Networks*, vol. 6, no. 7, pp. 975–990, 1993.

[7] F. Albertini and E. D. Sontag, "State observability in recurrent neural networks," *Systems and Control Letters*, vol. 22, pp. 235–244, 1994.

[8] F. Albertini, E. D. Sontag, and V. Maillot, "Uniqueness of weights for neural networks," in *Artificial Neural Networks with Applications in Speech and Vision* (R. Mammone, ed.), London: Chapman and Hall, 1993.

[9] J. Anderson, "Neural models for cognitive computation," *IEEE Transactions: Systems, Man, and Cybernetics*, vol. SMC-13, pp. 799–815, 1983.

[10] M. Arbib, ed., *The handbook of brain theory and neural networks*. Cambridge, MA: MIT Press, 1995.

[11] V. I. Arnold, *Mathematical methods of classical mechanics*. New York, NY: Springer-Verlag, 1989.

[12] A. R. Barron, "Neural net approximation," in *Proceedings of the Yale Workshop on Adaptive Learning Systems* (K. Narendra, ed.), Yale University, New Haven, CT, May 1992.

[13] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoïdal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

[14] P. Barson, S. Field, N. Davey, G. McAskie, and R. Frank, "The detection of fraud in mobile phone networks," *Neural Network World*, vol. 6, no. 4, pp. 477–484, 1996.

[15] R. E. Bellman, *Adaptive control processes*. Princeton, N. J.: Princeton University Press, 1961.

[16] E. Bienenstock and S. Geman, "Neural networks and the bias-variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[17] H. Bourlard and C. J. Wellekens, "Links between Markov models and multilayer perceptrons," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 12, pp. 1167–1178, 1990.

[18] S. Boyd, L. El Ghaoui, and E. Feron, *Linear matrix inequality in system and control theory*, vol. 15 of *Studies in Applied Mathematics*. Philadelphia, PA: SIAM, 1994.

[19] L. Breiman, "Hinging hyperplanes for regression, classification and function approximation," *IEEE Transactions on Information Theory*, vol. 3, pp. 999–1013, 1993.

[20] L. Breiman, "Discussion of neural networks and related methods for classification," *Journal of the Royal Statistical Society B*, vol. 56, no. 3, pp. 409–456, 1994.

[21] L. Brenig, "Complete factorization and analytic solutions of generalized Lotka-Volterra equations," *Physics Letters A*, vol. 133, no. 7–8, pp. 378–382, 1988.

[22] P. Burge, J. Shawe-Taylor, Y. Moreau, B. Preneel, C. Störmann, and C. Cooke, "Fraud detection and management in mobile telecommunications networks," in *Proceedings of the IEE European Conference on Security and Detection*, London, pp. 91–96, 1997.

[23] P. Burge, J. Shawe-Taylor, Y. Moreau, H. Verrelst, C. Störmann, and P. Gosset, "BRUTUS — A hybrid detection tool," in *Proceedings of the ACTS Mobile Telecommunications Summit 1997*, Aalborg, Denmark, October 1997.

[24] L. O. Chua, M. Hasler, J. Neirynck, and P. Verburgh, "Dynamics of a piecewise-linear resonant circuit," *IEEE Transactions on Circuits and Systems CAS*, vol. 29, pp. 535–547, 1982.

[25] L. O. Chua, T. Roska, and T. Kozek, "Cellular neural networks — A tutorial on programmable nonlinear dynamics in space," *Lecture Notes in Computer Science*, vol. 888, pp. 53–85, 1995.

[26] M. A. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 815–826, 1983.

[27] L. Connor, L. Brothers, and J. Alspector, "Neural network detection of fraudulent calling card patterns," in *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications*, pp. 362–370, Lawrence Erlbaum Associates, 1995.

[28] G. Cybenko, "Approximation by superpositions of a sigmoïdal function," *Math. Control Signals Syst.*, vol. 2, pp. 303–314, 1989.

[29] A. Delgado, *Input–output linearization of control affine systems using neural networks.* PhD thesis, The University of Reading, Department of Cybernetics, Reading, UK, 1996.

[30] A. Delgado, C. Kambhapati, and K. Warwick, "Inverse nonlinear control with a dynamic recurrent neural network," in *Proceedings of the International ICSC Symposia on Intelligent Industrial Automation and Soft Computing* (P. G. Anderson and K. Warwick, eds.), vol. A, University of Reading, UK, pp. 64–70, ICSC Press, March 26–28, 1996.

[31] R. A. DeVore, R. Howard, and C. Micchelli, "Optimal nonlinear approximation," *Manuscripta Mathematica*, vol. 63, pp. 469–478, 1989.

[32] J.-P. Draye, D. A. Pavisic, G. A. Cheron, and G. A. Libert, "Dynamic recurrent neural networks: A dynamical analysis," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-26, no. 5, pp. 692–706, 1996.

[33] L. Dupré, *BUGS in writing: A guide to debugging your prose.* Reading, MA: Addison-Wesley, 1995.

[34] B. Ermentrout and N. Kopell, "Learning of phase-lags in coupled neural oscillators," *Neural Computation*, vol. 6, no. 2, pp. 225–241, 1994.

[35] K. Fanning, K. Cogger, and R. Srivasta, "Detection of management fraud: A neural network approach," *International Journal of Intelligent Systems in Accounting, Finance, and Management*, vol. 4, pp. 113–126, 1995.

[36] T. Fawcett and F. Provost, "Adaptive fraud detection," *Data Mining and Knowledge Discovery*, vol. 3, no. 1, pp. 291–316, 1997.

[37] E. Forest and R. Ruth, "Fourth-order symplectic integration," *Physica D*, vol. 43, pp. 105–117, 1990.

[38] M. Forti and A. Tesi, "New conditions for global stability of neural networks with application to linear and quadratic programming problems," *IEEE Transactions on Circuits and Systems—I: Fundamental theory and applications*, vol. 42, no. 7, pp. 354–366, 1995.

[39] J. H. Friedman, "Multivariate adaptive regression splines (with discussion)," *Annals of Statistics*, vol. 19, pp. 1–141, 1991.

[40] J. H. Friedman, "An overview of predictive learning and function approximation," in *From Statistics to Neural Networks, Theory and Pattern Recognition Applications* (V. Cherkassky, J. H. Friedman, and H. Wechsler, eds.), vol. 136 of *NATO ASI Series F*, pp. 1–61, Springer, 1994.

[41] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–192, 1988.

[42] K.-I. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Networks*, vol. 6, no. 6, pp. 801–806, 1993.

[43] A. R. Gallant and H. White, "There exists a neural network that does not make avoidable mistakes," in *Proceedings of the IEEE Second International Conference on Neural Networks*, vol. 1, San Diego, CA, pp. 657–664, SOS Printing, 1988.

[44] S. Ghosh and D. Reilly, "Credit card fraud detection with a neural network," in *Proceedings of the 27th Annual Hawaii International Conference on System Science*, vol. 3, pp. 621–630, IEEE Computer Society Press, 1994.

[45] F. Girosi, "An equivalence between sparse approximation and support vector machines," Tech. rep. A.I. Memo 1606, Artificial Intelligence Laboratory, M.I.T, 1997.

[46] F. Girosi and G. Anzellotti, "Convergence rates of approximation by translates," Tech. rep. 1288, M.I.T., Artificial Intelligence Laboratory, Cambridge, MA, 1992.

[47] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural network architectures," *Neural Computation*, vol. 7, pp. 219–269, 1995.

[48] G. Golub and C. Van Loan, *Matrix computations*. Baltimore, MD: The Johns Hopkins University Press, 1989.

[49] J.-L. Gouzé, "Transformation of polynomial differential systems in the positive orthant," Tech. rep. INRIA-1308, Institut National de Recherche en Informatique et en Automatique: Unité de Recherche INRIA-Sophia Antipolis, France, 1990.

[50] S. Grossberg, "Competition, decision, and consensus," *Journal of Mathematical Analysis and Applications*, vol. 66, pp. 470–493, 1978.

[51] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Networks*, vol. 1, no. 1, pp. 17–61, 1988.

[52] J. Guckenheimer, A. Back, M. Myers, F. Wicklin, and P. Worfolk, "Dstool: Computer assisted exploration of dynamical systems," *Notices of the American Mathematical Society*, vol. 39, pp. 303–309, 1992.

[53] J. Guckenheimer and P. Holmes, *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*. No. 42 in Applied Mathematical Sciences, New York: Springer-Verlag, 1983.

[54] J. Hauser, S. Sastry, and P. Kokotovic, "Nonlinear control via approximate input-output linearization: The ball and beam example," in *Proceedings of the 28th IEEE Conference on Decision and Control*, Tampa, FL, pp. 1987–1993, December 1989.

[55] H. O. Hebb, *The organization of behavior: A neuropsychological theory*. New York: Wiley, 1949.

[56] R. Hermann and A. J. Krener, "Nonlinear controllability and observability," *IEEE Transactions on Automatic Control*, vol. 22, pp. 728–740, 1977.

[57] N. J. Higham, *Handbook of writing for the mathematical sciences*. Philadelphia: SIAM, 1993.

[58] M. W. Hirsch, "Convergent activation dynamics in continuous time networks," *Neural Networks*, vol. 2, pp. 331–349, 1989.

[59] M. W. Hirsch and S. Smale, *Differential equations, dynamical systems, and linear algebra*. Orlando, FL: Academic Press, 1974.

[60] R. M. Hirschorn, "Invertibility of nonlinear control systems," *SIAM Journal of Control and Optimization*, vol. 17, pp. 289–297, 1979.

[61] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Science USA*, vol. 79, pp. 2554–2558, 1984.

[62] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[63] M. C. Irwin, *Smooth dynamical systems*. New York, NY: Academic Press, 1980.

[64] A. Isidori, *Nonlinear control theory*. New York, NY: Springer-Verlag, 1989.

[65] L. K. Jones, "A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression," *Annals of Statistics*, vol. 20, pp. 608–613, 1992.

[66] P. C. Kainen, V. Kůrková, and A. Vogt, "Approximation by neural networks is not continuous," Technical Report, Georgetown University, Industrial Math, Washington, D.C., 1998.

[67] C. Kambhapati, S. Manchanda, A. Delgado, G. Green, K. Warwick, and M. Tham, "The relative order and inverses of recurrent networks," *Automatica*, vol. 32, no. 1, pp. 117–123, 1996.

[68] D. E. Knuth, T. Larrabee, and P. M. Roberts, *Mathematical writing*. No. 14 in MAA Notes, Washington, D. C.: The Mathematical Association of America, 1989.

[69] P.-V. Koseleff, *Calcul formel pour les méthodes de Lie en mécanique hamiltonienne*. PhD thesis, Ecole Polytechnique, Paris, 1993.

[70] S. Y. Kung, *Digital neural networks*. Englewood Cliffs: Prentice Hall, 1993.

[71] V. Kůrková, "Kolmogorov's theorem and multilayer neural networks," *Neural Networks*, vol. 5, pp. 501–506, 1992.

[72] G. Lafferiere and H. J. Sussman, "Motion planning for controllable systems without drift," in *Proceedings of the IEEE Conference on Robotics and Automation*, Scramento, CA, pp. 1148–1153, April 1991.

[73] J. P. LaSalle, "Stability theory of differential equations," *Journal of Differential Equations*, vol. 4, pp. 57–65, 1968.

[74] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[75] M. Leshno, V. Y. Lin, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, 1993.

[76] A. U. Levin and K. S. Narendra, "Control of nonlinear dynamical systems using neural networks: Controllability and stabilization," *IEEE Transactions on Neural Networks*, vol. 4, no. 2, pp. 192–206, 1993.

[77] A. U. Levin and K. S. Narendra, "Control of nonlinear dynamical systems using neural networks—Part II: Observability, identification, and control," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 30–42, 1996.

[78] R. Lippmann, "An introduction to neural computing," *IEEE Signal Processing Magazine*, vol. 4, no. 2, pp. 4–22, 1987.

[79] C.-C. Liu and F.-C. Chen, "Adaptive control of non-linear continuous-time systems using neural networks—general relative degree and mimo cases," *International Journal of Control*, vol. 58, no. 2, pp. 317–335, 1993.

[80] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of Atmospheric Science*, vol. 20, pp. 131–141, 1963.

[81] R. H. MacArthur, "Species packing, or what competition minimizes," *Proceedings of the National Academy of Sciences, USA*, vol. 54, pp. 1369–1375, 1969.

[82] R. M. May and W. J. Leonard, "Nonlinear aspects of competition between three species," *SIAM Journal of Applied Mathematics*, vol. 29, pp. 243–253, 1975.

[83] D. F. McCaffrey and A. R. Gallant, "Convergence rates for single hidden layer feedforward networks," Tech. rep., North Carolina State University, Department of Statistics, 1992.

[84] W. S. McCullogh and W. H. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Physiology*, vol. 5, pp. 113–133, 1943.

[85] R. I. McLachlan, "On the numerical integration of ordinary differential equations by symmetric composition methods," *SIAM J. Sci. Comput.*, vol. 16, no. 1, pp. 151–168, 1995.

[86] H. N. Mhaskar and C. A. Micchelli, "Approximation by superposition of a sigmoïdal function," *Advances in Applied Mathematics*, vol. 13, pp. 350–373, 1992.

[87] M. Minsky and S. Papert, *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[88] Y. Moreau, S. Louies, J. Vandewalle, and L. Brenig, "Embedding recurrent neural networks into predator-prey models," Tech. rep. ESAT-SISTA TR98-02, Katholieke Universiteit Leuven, Department of Electrical Engineering, 1998. Submitted to *Neural Networks*.

[89] Y. Moreau, B. Preneel, P. Burge, J. Shawe-Taylor, C. Stoermann, and C. Cooke, "Novel techniques for fraud detection in mobile telecommunications," in *ACTS Mobile Summit*, Granada, Spain, 1996.

[90] Y. Moreau, R. Sepulchre, and J. Vandewalle, "Accelerated implementation of forwarding control laws using composition methods," Tech. rep. ESAT-SISTA TR97-110, Katholieke Universiteit Leuven, 1997.

[91] Y. Moreau and J. Vandewalle, "A Lie algebraic approach to dynamical system prediction," in *Proceedings of the European Symposium on Artificial Neural Networks 96*, vol. 3, Atlanta, GA, pp. 182–185, May 1996.

[92] Y. Moreau and J. Vandewalle, "Prediction of dynamical systems with composition networks," in *Proceedings of the European Symposium on Artificial Neural Networks 96*, Brugge, Belgium, pp. 303–308, April 1996.

[93] Y. Moreau and J. Vandewalle, "System modeling using composition networks," in *Preprints of the Second IEEE Workshop on Computer Intensive Methods in Control and Signal Processing*, Prague, pp. 183–192, August 1996.

[94] Y. Moreau and J. Vandewalle, "Composition methods for the integration of dynamical neural networks," in *Proceedings of the European Symposium on Artificial Neural Networks 97*, Brugge, Belgium, pp. 303–308, April 1997.

[95] Y. Moreau and J. Vandewalle, "Dynamical system prediction: A Lie algebraic approach for a novel neural architecture," in *Mathematics of Neural Networks: Models, Algorithms and Applications* (S. W. Ellacott, J. C. Mason, and I. J. Anderson, eds.), Kluwer Academic Publisher, 1997.

[96] Y. Moreau and J. Vandewalle, "Equivalence between dynamic one-hidden-layer perceptrons and dynamic perceptrons without hidden layer," in *Proceedings of the European Control Conference*, Brussels, July 2–4, 1997.

[97] Y. Moreau and J. Vandewalle, "Modeling dynamical systems using compositions," Tech. rep. ESAT-SISTA TR97-79, Katholieke Universiteit Leuven, Department of Electrical Engineering, Leuven, Belgium, 1997. Submitted to *IEEE Transactions on Neural Networks*.

[98] Y. Moreau and J. Vandewalle, "Simulation of arrays of Chua's circuits by composition methods," Tech. rep. ESAT-SISTA TR97-115, Katholieke Universiteit Leuven, Department of Electrical Engineering, Leuven, Belgium, 1997. To appear in *Proceedings of the European Symposium on Artificial Neural Networks 98*.

[99] Y. Moreau and J. Vandewalle, "System modeling using composition networks," in *Computer Intensive Methods in Signal Processing and Control*, ch. 7, pp. 119–140, Boston: Birkhauser, 1997.

[100] Y. Moreau and J. Vandewalle, "Time-integration of dynamical neural networks by composition methods," Tech. rep. ESAT-SISTA TR97-43, Katholieke Universiteit Leuven, Belgium, 1997.

[101] Y. Moreau and J. Vandewalle, "When do dynamical neural networks with and without hidden layer have identical behavior? ," Tech. rep. ESAT-SISTA TR97-51, Katholieke Universiteit Leuven, Department of Electrical Engineering, Leuven, Belgium, 1997. Submitted to *Neurocomputing*.

[102] Y. Moreau and J. Vandewalle, "Composition methods for the simulation of arrays of Chua's circuits," Tech. rep. ESAT-SISTA TR98-05, Katholieke Universiteit Leuven, Department of Electrical Engineering, Leuven, Belgium, 1998. To appear in *International Journal of Bifurcations and Chaos*.

[103] Y. Moreau, H. Verrelst, and J. Vandewalle, "Detection of mobile phone fraud using supervised neural networks: A first prototype," in *Lecture Notes in Computer Science 1327* (W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, eds.), Berlin, pp. 1065–1070, Springer, 1997.

[104] Y. Moreau, H. Verrelst, and J. Vandewalle, "Fraud detection in mobile communications using supervised neural networks," in *SNN'97: Europe's Best Neural Networks Practice* (B. Kappen and S. Gielen, eds.), pp. 149–152, Singapore: World Scientific, 1998.

[105] A. Muñuzuri, V. Pérez-Muñuzuri, M. Gómez-Gesteira, L. Chua, and V. Pérez-Villar, "Spatiotemporal structures in discretely-coupled arrays of nonlinear circuits: A review," *International Journal of Bifurcations and Chaos*, vol. 5, no. 1, pp. 17–50, 1995.

[106] D. Mumford, "On the computational architecture of neocortex," *Biological Cybernetics*, vol. 65, pp. 135–145, 1991.

[107] R. W. Murray and S. S. Sastry, "Steering nonholonomic systems using sinusoids," in *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, HA, pp. 2097–2101, December 1990.

[108] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[109] K. S. Narendra and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 252–262, 1991.

[110] K. S. Narendra and J. H. Taylor, *Frequency domain criteria for absolute stability*. Academic Press, 1973.

[111] H. Nijmeijer and A. J. van der Schaft, *Nonlinear dynamical control systems*. New York: Springer-Verlag, 1990.

[112] M. Nikolaou and V. Haganandi, "Control of nonlinear dynamical systems modeled by recurrent neural networks," *AIChE*, vol. 39, pp. 1890–1894, 1993.

[113] P. J. Olver, *Applications of Lie groups to differential equations*. New York, NY: Springer-Verlag, 1986.

[114] B. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, pp. 263–269, 1989.

[115] M. Peschel and W. Mende, *The predator-prey model*. New York: Springer-Verlag, 1986.

[116] L. Pivka, "Autowaves and spatio-temporal chaos in CNNs—Part I: A tutorial," *IEEE Transactions on Circuits and Systems—I: Fundamental theory and applications*, vol. 42, no. 10, pp. 623–637, 1995.

[117] R. Redheffer and W. Walter, "Solution of the stability problem for a class of generalized Volterra prey-predator systems," *Journal of Differential Equations*, vol. 52, pp. 245–263, 1984.

[118] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge: Cambridge University Press, 1996.

[119] A. J. Robinson and F. Fallside, "Static and dynamic error propagation networks with application to speech coding," in *Proceedings of Neural Information Processing Systems* (D. Z. Anderson, ed.), American Institute of Physics, 1987.

[120] F. Rosenblatt, "The perceptron: A probabilistic model for information storage in the brain," *Psychological review*, vol. 65, pp. 386–408, 1958.

[121] W. Rudin, *Functional analysis*. New York: McGraw-Hill, 1973.

[122] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by backpropagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[123] M. Saerens and A. Soquet, "Neural controller based on backpropagation," in *IEE Proceedings F*, vol. 138, pp. 55–62, 1991.

[124] S. S. Sastry and Z. Li, "Robot motion planning with nonholonomic constraints," in *Proceedings of the 28th IEEE Conference on Decision and Control*, Tampa, FL, pp. 211–216, December 1989.

[125] M. Sato, "A learning algorithm to teach spatio-temporal patterns to recurrent neural networks," *Biological Cybernetics*, vol. 62, pp. 259–263, 1990.

[126] T. Sejnowski and C. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Systems*, vol. 1, pp. 145–168, 1987.

[127] T. J. Sejnowski, C. Koch, and P. S. Churchland, "Computational neuroscience," *Science*, vol. 241, pp. 1299–1306, 1988.

[128] R. Sepulchre, M. Jankovic, and V. Kokotovic, *Constructive nonlinear control*. London: Springer-Verlag, 1997.

[129] S. Smale, "On the differential equations of species in competition," *Journal of Mathematical Biology*, vol. 3, pp. 5–7, 1976.

[130] E. Sontag, *Mathematical control theory: Deterministic finite dimensional systems*. New York, NY: Springer Verlag, 1990.

[131] E. Sontag, "Neural nets as system models and controllers," in *Proc. 7th Yale Workshop on Adaptive and Learning Systems*, pp. 73–79, 1992.

[132] E. D. Sontag, "Personal communication," 1997.

[133] E. D. Sontag and H. J. Sussman, "Complete controllability of continuous-time recurrent neural networks," *Systems and Control Letters*, vol. 30, pp. 177–183, 1997.

[134] H. J. Sussman, "A product expansion for the chen series," in *Theory and Applications of Nonlinear Control Systems* (C. Byrnes and A. Lindquist, eds.), pp. 323–335, North-Holland, 1986.

[135] H. J. Sussman, "Uniqueness of the weights for minimal feedforward nets with a given input-output map," *Neural Networks*, vol. 5, pp. 589–593, 1992.

[136] J. Suykens, B. De Moor, and J. Vandewalle, *Artificial neural networks for modelling and control of nonlinear systems*. Boston: Kluwer Academic Publisher, 1997.

[137] J. Suykens and J. Vandewalle, "Absolute stability criterion for a Lur'e problem with multilayer perceptron nonlinearity," in *Proceedings of the International Workshop on Nonlinear Dynamics of Electronic Systems NDES 96*, Sevilla, Spain, pp. 117–122, June 1996.

[138] M. Suzuki, "Convergence of general decompositions of exponential operators," *Communications in Mathematical Physics*, vol. 163, pp. 491–508, 1994.

[139] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence* (D. Rand and L.-S. Young, eds.), vol. 898 of *Lecture Notes in Mathematics*, pp. 336–381, Springer-Verlag, 1981.

[140] Y. Takeuchi, *Global dynamical properties of Lotka-Volterra systems*. Singapore: World Scientific Publishing Company, 1996.

[141] R. Tibshirani, "A comparison of some error estimates for neural network models," *Neural Computation*, vol. 8, no. 1, pp. 152–163, 1996.

[142] N. B. Toomarian and J. Barhen, "Learning a trajectory using adjoint functions and teacher forcing," *Neural Networks*, vol. 5, no. 3, pp. 473–484, 1992.

[143] V. N. Vapnik, S. E. Golowich, and A. Smola, "Support vector method for function approximation, regression, estimation, and signal processing," in *Advances in Neural Information Processing Systems*, vol. 9, San Mateo, CA, pp. 281–287, Morgan Kaufmann, 1997.

[144] F. Verhulst, *Nonlinear differential equations and dynamical systems*. Berlin: Springer-Verlag, 1985.

[145] E. Wan, *Finite impulse response neural networks with applications in time series prediction*. PhD thesis, Stanford University, 1993.

[146] N. Wiener, *Cybernetics: On control and communication in the animal and the machine*. New York: The Technology Press and Wiley, 1948.

[147] J. H. Wilkinson, *The algebraic eigenvalue problem*. Oxford University Press, 1965.

[148] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[149] H. Yoshida, "Construction of higher order symplectic integrators," *Phys. Lett. A.*, vol. 150, pp. 262–269, 1990.

[150] R. W. Żbikowski, "Lie algebra of recurrent neural networks and identifiability," in *Proceedings of the American Control Conference*, San Francisco, pp. 2900–2901, 1993.

[151] R. W. Żbikowski, *Recurrent neural networks: Some control aspects*. PhD thesis, University of Glasgow, Department of Mechanical Engineering, UK, 1994.

Index

activation function
    properties of, 84
    sigmoidal, 18
    squashing, 20
Albertini, F., 36, 94, 100
approximation rate, 29
asymptotic stability, 47

backpropagation through time, 88
Baker–Campbell–Hausdorff
    formula, 165
ball and beam system, 168
Barron, A. R., 29
Brenig, L., 75

canonical form, 154
canonical neural network, 157
canonical subspace, 94
cellular neural network, 174
change of coordinates, 69, 75, 118,
        154
Chua's circuit, 174
codistribution, 143
Cohen, M. A., 49, 51
competitive dynamical system, 49
composition method, 165, 171
composition network, 185
conjugacy, 116
continuous selection, 32
control-affine system, 135
controllability, 96
    rank condition, 142
convergence, 46
convex hull, 30, 96
covector field, 138
curse of dimensionality, 33

decomposition of operator, 173
Delgado, A., 157
dense family, 19
DeVore, R. A., 31
diffeomorphism, 164
directional derivative, 136
discrete-time neural control, 104
discrete-time neural network, 103
distribution, 139
    integrable, 141
    involutive, 140
    nonsingular, 140
Draye, J.-P., 89
dual space, 138

equilibrium of Hopfield network, 62
equivalence, 116
exponential map, 164

feedback
    linearization, 159
    nilpotentization, 206
foliation, 141
forwarding control, 169
Fourier representation, 29
Frobenius's theorem, 141

global asymptotic stability, 63
gradient
    of composition network, 186
Grossberg, S., 49, 51

Hall basis, 202
Hamiltonian, 90
Hermann, R., 143, 144
Hirschorn, R. M., 147
Hopfield network, 45
Hopfield-type network, 60
Howard, R., 31

immersibility, 116
immersion, 115
independence property, 37, 93, 99
input–output equivalence, 100
integrability, 141
integral manifold, 140
invariant set, 48
invariant subspace
    for a pair of matrices, 94
inverse of a dynamical neural network, 148
invertibility, 164, 203
involutive closure, 142
irreducibility, 37
irreducible matrix $\mathcal{B}^{n\times m}$, 93, 96, 101
irregularly sampled system, 191
Isidori, A., 154, 160

Jacobi identity, 137
Jones, L. K., 30

Krener, A. J., 143, 144

Lafferriere, G., 200
LaSalle invariance principle, 48
leapfrog composition scheme, 167
learning algorithm, 88
Levin, A. U., 104
Lie algebra, 137
Lie bracket, 137
Lie derivative, 136
    of a covector field, 139
linear matrix inequality, 64
Lipschitz continuity, 46
local controllability, 96
Lotka–Volterra system, 75
Lur'e system, 59
Lyapunov diag. stable matrix, 62
Lyapunov function, 48, 51
    Lur'e–Postnikov, 59
    strict, 54
Lyapunov stability, 47
Lyapunov theorem, 48

Maillot, V., 36
McLachlan, R. I., 166
Micchelli, C., 31
MIMO system, 114
motion-planning problem, 200

Narendra, K. S., 104
NARMAX model, 105
NARX model, 104
neural computation, 2
nilpotent
    Lie algebra, 201
    system, 201
nonpolynomial activation, 24
notation, 15

observability, 94
    local weak, 144
    of neural network, 95
    rank condition, 144
optimal approximation, 31

P matrix, 61
parallel mode, 106
Parthasarathy, K., 104
perceptron
    dynamical, 45, 68
      multilayer, 45, 82
      single–hidden-layer, 45
      with self-connections, 45, 77
    static
      multilayer, 18
      single–hidden-layer, 18
Pontryagin minimum principle, 90

prediction of dynamical system, 185
product of powers of variables, 75
pseudoinverse, 69, 117

quasi-convergence, 47
quasi-monomial form, 70

rank condition
    controllability, 142
    observability, 144
rank degeneracy, 69, 121
rate of convergence, 29
real-time recurrent learning, 88
ridge decomposition, 24
Runge–Kutta integration, 171

sector condition, 62
series expansion, 34
series–parallel mode, 105
sigmoidal activation, 18
sign–permutation equivalence, 38, 100
singular value decomposition, 118
Sobolev space, 31
Sontag, E. D., 36, 94, 96, 100
steering, 200
Sussman, H. J., 36, 96, 200
symmetric interaction, 51
system identification, 192

uncontrollability and arctangent activation, 97
uniform convergence, 21
uniqueness of the weights
    of feedforward network, 35
uniqueness of weights
    of recurrent network, 100
universal approximation, 19

Weierstrass theorem, 21

Żbikowski, R. W., 144
zero dynamics, 156

Yves Moreau was born on July 8, 1970 in Haine-St-Paul, Belgium. In July 1992, he received the Electrical Engineering degree from the Faculté Polytechnique de Mons, Belgium. In May 1994, he was awarded the M.Sc. in Applied Mathematics from Brown University, Providence, RI, USA. From August 1992 to May 1993, he held a Fulbright grant at the Division of Applied Mathematics of Brown University. Since October 1994, he has been a research assistant of the F.W.O. - Vlaanderen (Belgian Fund for Scientific Research - Flanders) working towards his Ph.D. at the Department of Electrical Engineering of the Katholieke Universiteit Leuven.