Neurocomputing 298 (2018) 46–54

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Deep hybrid neural-kernel networks using random Fourier features

Siamak Mehrkanoon*, Johan A.K. Suykens

KU Leuven, ESAT-STADIUS, Kasteelpark Arenberg 10, Leuven (Heverlee) B-3001, Belgium

A R T I C L E I N F O

Article history:
Received 8 July 2017
Revised 2 November 2017
Accepted 27 December 2017
Available online 24 February 2018

Keywords:
Deep learning
Neural networks
Explicit feature mapping
Kernel methods
Hybrid models

A B S T R A C T

This paper introduces a novel hybrid deep neural kernel framework. The proposed deep learning model makes a combination of a neural networks based architecture and a kernel based model. In particular, here an explicit feature map, based on random Fourier features, is used to make the transition between the two architectures more straightforward as well as making the model scalable to large datasets by solving the optimization problem in the primal. Furthermore, the introduced framework is considered as the first building block for the development of even deeper models and more advanced architectures. Experimental results show an improvement over shallow models and the standard non-hybrid neural networks architecture on several medium to large scale real-life datasets.

1. Introduction

Conventional machine learning techniques were limited in processing natural data in their raw forms and a lot of domain experts were required in transforming raw data into meaningful features or representations. Recent years have witnessed considerable interests in models with deep architectures, inspired by the layered architecture of the human visual cortex, due to their successful impact in revolutionizing many application fields ranging from auditory to vision sensory signal processing such as computer vision, speech processing, natural language processing and game playing among others.

Deep Learning is a class of machine learning techniques that belongs to the family of representation learning models [1,2]. Deep learning models deal with complex tasks by learning from subtasks. In particular, several nonlinear modules are stacked in hierarchical architectures to learn multiple levels of representation (hierarchical features) from the raw input data. Each module transforms the representation at one level into a slightly more abstract representation at a higher level, i.e., the higher-level features are defined in terms of lower-level ones. Deep learning architectures have grown significantly, resulting in different models such as stacked denoising autoencoders [3,4], Restricted Boltzmann Machines [5–7], Convolutional Neural Networks [8,9], Long Short Term Memories [10] among others.

Recent works in machine learning have highlighted the superiority of deep architectures over shallow architectures in terms of accuracy in several application domains [1,11]. However, training deep neural networks involves costly nonlinear optimization problems and demands huge amount of labeled training data. The generalization performance of deep artificial neural networks largely depends on the parameters of the model of which they can be thousands to learn. Furthermore, finding the right architecture such as the number of layers and hidden units, the type of activation functions among others, as well as the networks associated hyper-parameters become a difficult task with increasing complexity of deep architectures. Most of the developed deep learning models are based on artificial neural networks (ANN) architecture, whereas deep kernel based models have not yet been explored in great detail. On the other hand support vector machines (SVM) and kernel based methods have also made a large impact in a wide range of application domains, with their strong foundations in optimization and learning theory [12–14] and are able to handle high-dimensional data directly.

Therefore, exploring the existing synergies or hybridization between ANN and Kernel based models can potentially lead to the development of models that have the best of two worlds. One has started already to explore such directions e.g., Kernel Methods for Deep Learning and a family of positive-definite kernel functions that mimic the computation in multilayer neural networks [15], Convolutional kernel networks [16], Deep Gaussian processes [17,18]. In particular, the authors in [19] introduced a convex deep learning model via normalized kernels. The authors in [20] investigated iterated compositions of Gaussian kernels with an interpretation that resembles a deep neural networks architecture. A kernel based Convolutional Neural Network is introduced in [16] where

* Corresponding author.
E-mail addresses: siamak.mehrkanoon@esat.kuleuven.be, mehrkanoon2011@gmail.com (S. Mehrkanoon), johan.suykens@esat.kuleuven.be (J.A.K. Suykens).

https://doi.org/10.1016/j.neucom.2017.12.065
0925-2312/© 2018 Elsevier B.V. All rights reserved.

new representations of the given image are obtained by stacking and composing kernels at different layers. A survey of recent attempts and motivations existing in the community for finding such a synergy between the two frameworks is also discussed in [21].

In this paper, we discuss possible strategies to bridge neural networks and kernel based models. The approach has been originally proposed in our previous work [22] where a two layer hybrid deep neural kernel network is introduced. Here, the model introduced in [22] is used as first building block for developing even deeper models. Aspects of model selection are discussed. Some new test problems are added and comparisons with other deep architectures are performed.

This paper is organized as follows. In Section 2, the existing connections between artificial neural networks and kernel based models are explored. Section 3, briefly reviews existing techniques in kernel based models with explicit feature mapping and introduces the proposed hybrid deep neural-kernel architecture that can take advantage of the best of two worlds by bridging between the ANN and kernel based models. Section 4 reports the experimental results. Conclusions and future works are drawn in Section 5.

2. ANNs vs. kernel architecture

In neural networks based approaches, the input is projected into a new space via multiplication with a weight matrix followed by applying a nonlinear activation function. If we consider the described operations in a module, then one can stack together several of these modules and form a deep architecture. In this way, different configurations of stacking as well as wiring used in the entire networks can cover different modeling strategies. On the other hand, in kernel based approaches one often works with the primal-dual setting. In the primal formulation, one projects the input data into a potentially infinite dimensional space using implicit feature mapping. The projected data points are then mapped to a target space by means of an inner product with a weight matrix (primal decision variables). Alternatively one could also work with an explicit feature map and project the data into a finite dimensional feature space where each of its elements can be approximated. In the case of an implicit feature mapping, the projection space corresponds to a hidden layer in a neural network with infinite number of neurons [23]. Whereas when using an explicit feature map the connection with neural network architectures becomes even closer. More precisely, the dimension of the explicit feature map corresponds to the number of hidden units in the hidden layer of a neural network architecture.

It should be noted that in kernel based approaches, when in the given dataset the number of instances is much larger than the feature dimensions, one may prefer to use an explicit feature map and solve the optimization problem in the primal. In the case that the number of variables is much larger than the number of instances, thanks to the implicit feature mapping and the kernel trick, the problem can be solved in the dual.

In particular, the Least Squares Support Vector Machines (LS-SVM) framework with implicit and explicit feature mapping is shown in Fig. 1. Here, given training data points $\mathcal{D} = \{x_1, \dots, x_n\}$, where $\{x_i\}_{i=1}^n \in \mathbb{R}^d$ and the targets $\{y_i\}_{i=1}^n$, one assumes that the underlying function describing the relation between input and output of the system has the following form:

$$y(x) = w^T \varphi(x) + b, \tag{1}$$

where $\varphi(\cdot) : \mathbb{R}^d \to \mathbb{R}^h$ is the feature map and h is the dimension of the feature space. Thanks to the nonlinear feature map, the data are embedded into a feature space and the optimal solution is sought in that space by minimizing the residual between the model outputs and the measurements. To this end, one formulates

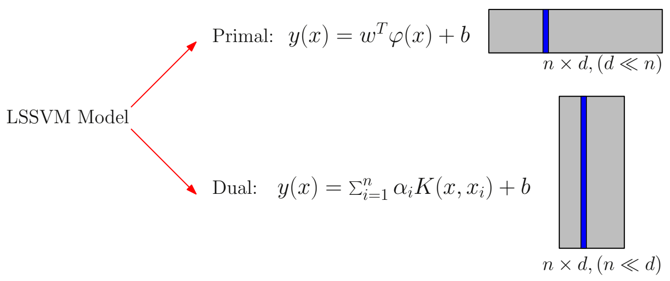
LSSVM Model
Primal: $y(x) = w^T\varphi(x) + b$
$n \times d, (d \ll n)$
Dual: $y(x) = \Sigma_{i=1}^n \alpha_i K(x, x_i) + b$
$n \times d, (n \ll d)$

Fig. 1. Primal and Dual Formulation of LS-SVM kernel models.

the following optimization problem known as primal LS-SVM formulation for regression problems [23]:

$$\begin{aligned} \underset{w,b,e}{\text{minimize}} \quad & \frac{1}{2} w^T w + \frac{\gamma}{2} e^T e \\ \text{subject to} \quad & y_i = w^T \varphi(x_i) + b + e_i, \quad i = 1, \dots, n, \end{aligned} \tag{2}$$

where $\gamma \in \mathbb{R}^+, b \in \mathbb{R}, \ w \in \mathbb{R}^h$. Depending whether an explicit or implicit feature map is used, one could solve (2) in the primal or dual.

- **Implicit feature map:** If one uses an implicit feature map φ which in general can be infinite dimensional, then the optimization problem cannot be solved in the primal. Therefore the kernel trick is used and the problem is solved in the dual [23]. Obtaining the Lagrangian of the constrained optimization problem (2) and eliminating the primal variables e_i and w leads to the following linear system in the dual problem:

$$\left[\begin{array}{c|c} \Omega + I_n/\gamma & 1_n \\ 1_n^T & 0 \end{array}\right] \begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix} \tag{3}$$

where $\Omega_{ij} = K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$ is the ijth entry of the positive definite kernel matrix. $1_n = [1, \dots, 1]^T \in \mathbb{R}^n$, $\alpha = [\alpha_1, \dots, \alpha_n]^T$, $y = [y_1, \dots, y_n]^T$ and I_n is the identity matrix. The model in the dual form becomes:

$$y(x) = w^T \varphi(x) + b = \sum_{i=1}^n \alpha_i K(x, x_i) + b. \tag{4}$$

- **Explicit feature map:** In case that an explicit feature map is used, then one can rewrite the original constraint optimization problem (2) as the following unconstrained optimization problem and solve it in the primal:

$$\min_{\hat{w},b} J(\hat{w}, b) = \frac{1}{2} \sum_{\ell=1}^{Q} \hat{w}^T \hat{w} + \frac{\gamma}{2} \sum_{i=1}^{n} (y_i - \hat{w}^T \hat{\varphi}(x_i) - b)^2 \tag{5}$$

where $\hat{\varphi}(\cdot)$ is an explicit finite dimensional feature map. In the next section some of the existing techniques are mentioned.

Inspecting the mathematical expressions of ANNs and kernel based models with explicit feature mapping, reveals the differences and similarities between the two frameworks, see Fig. 2.

In classical artificial neural network architectures the nonlinear activation functions such as sigmoid, hyperbolic tangent or Rectified Linear Units (ReLU), are applied on the weighted sum of the given input instance. For instance, a single-layer ANN where the inputs are directly connected to the output can be formulated as follows:

$$y(x) = f(Wx + b),$$

where $x \in \mathbb{R}^d$, $W \in x \in \mathbb{R}^{d_h \times d}$ and b is the bias vector. Here f denotes the activation function. On the contrary, in the kernel based approaches, the nonlinear feature map is directly applied on the given input instance and then the target value is estimated by

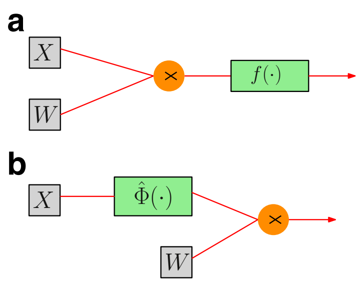

Fig. 2. Computational graph corresponding to (a) A single module of a neural network architecture. (b) A single module of kernel based models with explicit feature mapping.

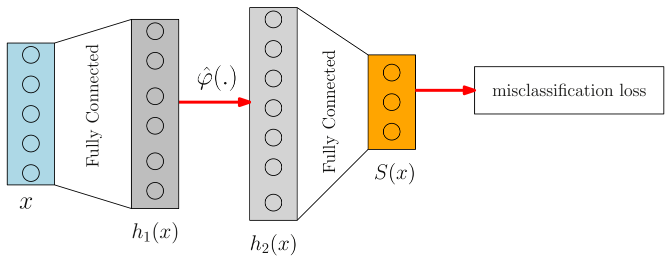

Fig. 3. Two-layer hybrid neural kernel network architecture.

means of a weighted sum of the projected sample. In the kernel based approach with an explicit feature map, the optimal values for the model parameters, i.e., weights and biases, can be learned in the primal. Furthermore, in this case, the dimension of $\hat{\varphi}(\cdot)$ can be larger or smaller than that of the input layer X. In contrast to the ANN module shown in Fig. 2(a), a single module of kernel based model is linear in the weight matrix W, therefore convex optimization techniques can be applied to obtain an optimal values of W. In addition, compared to a single ANN module, in practice, it has a better capability for learning nonlinear decision boundaries with a satisfactory generalization performance. It is worth noting that the matrix of the hidden layer shown in Fig. 2(a) can also be treated at the tuning parameter level (see [24]).

In today's applications, addressing large scale problems is inevitable. ANN based models can rely on the stochastic gradient descent algorithm for obtaining the optimized model parameters when the size of the data set is large. In kernel based approaches, on the other hand, one can for instance work with low rank approximation of the kernel matrix and avoid building and storing the entire kernel matrix which is not computationally efficient.

3. Proposed deep hybrid model

3.1. Two-layer model

Consider training data points $\mathcal{D}=\{x_1,\dots,x_n\}$, where $\{x_i\}_{i=1}^n \in \mathbb{R}^d$, the labels $\{y_i\}_{i=1}^n$ and the total number of classes is Q. This section introduces a new deep architecture configuration that bridges the ANN and kernel based models in the primal level. The proposed model is suitable for both regression and classification. For the kernel based model counterpart we employ an explicit feature mapping. In the literature, several methodologies are introduced to scale up the kernel based models for dealing with large scale problems. For instance Greedy basis selection techniques [25], incomplete Cholesky decomposition [26], and Nyström methods [27,28] aim at providing a low-rank approximation of the kernel matrix. In particular the Nyström approximation method as well as a reduced kernel technique have been previously successfully applied in the context of large scale semi-supervised learning for providing an approximation of the feature map and solving the optimization problem in the primal (see [29]).

In the Nyström approximation method, an explicit expression for φ can be obtained by means of an eigenvalue decomposition of the kernel matrix Ω. More precisely, the ith component of the the n-dimensional feature map feature map $\hat{\varphi}:\mathbb{R}^d \to \mathbb{R}^n$, for any point $x\in\mathbb{R}^d$, can be obtained as follows:

$$\hat{\varphi}_i(x)=\frac{1}{\sqrt{\lambda_i}}\sum_{k=1}^{n} u_{ki}K(x_k,x), \quad (6)$$

where $K(\cdot,\cdot)$ is the kernel function, λ_i and u_i are eigenvalues and eigenvectors of the kernel matrix $\Omega_{n\times n}$ whose (i, j)th element is defined as $K(x_i, x_j)$. The kth element of the ith eigenvector is denoted by u_{ki}. In practice when n is large, one can work with a subsample (prototype vectors) of size $m \ll n$. In this case, the m-dimensional feature map $\hat{\varphi}:\mathbb{R}^d\to\mathbb{R}^m$ can be approximated as follows:

$$\hat{\varphi}(x)=[\hat{\varphi}_1(x),\dots,\hat{\varphi}_m(x)]^T \quad (7)$$

where

$$\hat{\varphi}_i(x)=\frac{1}{\sqrt{\lambda_i}}\sum_{k=1}^{m} u_{ki}K(x_k,x),\ i=1,\dots,m \quad (8)$$

where λ_i and u_i are now eigenvalues and eigenvectors of the constructed kernel matrix $\Omega_{m\times m}$ using the selected prototype vectors. Among existing approaches for selecting the prototype vectors are for instances random selection, incomplete Cholesky factorization [26], clustering and entropy based techniques. The authors in [30], compared the performance of the Nyström approximation and the reduced kernel techniques to a newly introduced scalable semi-supervised kernel spectral clustering model by means of random Fourier features on real large scale datasets. In particular, as the RFF-MSSKSC model does not involve eigen-decomposition step, therefore it requires less training computation times, while the test accuracy is comparable to that of other explicit feature mapping techniques.

Therefore, here we use random Fourier features to compute an explicit feature map and build a module that can be cast into a neural networks architecture. Random Fourier features have recently been introduced in the field of kernel methods by exploiting the classical Bochner's theorem in harmonic analysis [31]. The Bochner's theorem states that a continuous kernel $K(x,y)=K(x-y)$ on $\mathbb{R}^d$ is positive definite if and only if K is the Fourier transform of a non-negative measure. If a shift-invariant kernel k is properly scaled, its Fourier transform $p(\xi)$ is a proper probability distribution. This property is used to approximate kernel functions with linear projections on D random features as follows [31]:

$$K(x-y)=\int_{\mathbb{R}^d} p(\xi)e^{j\xi^T(x-y)}d\xi=\mathbb{E}_\xi[z_\xi(x)z_\xi(y)^*], \quad (9)$$

where $z_\xi(x)=e^{j\xi^T x}$. Here $z_\xi(x)z_\xi(y)^*$ is an unbiased estimate of $K(x,\ y)$ when ξ is drawn from $p(\xi)$ (see [31]). To obtain a real-valued random feature for K, one can replace the $z_\xi(x)$ by the mapping $z_\xi(x)=\cos(\xi^T x)$. The random Fourier features $\hat{\varphi}(x)$, for the sample x, are then defined as

$$\hat{\varphi}(x)=\frac{1}{\sqrt{D}}[z_{\xi_1}(x),\dots,z_{\xi_D}(x)]^T\in\mathbb{R}^D. \quad (10)$$

Here $\frac{1}{\sqrt{D}}$ is used as normalization factor to reduce the variance of the estimate and $\xi_1,\dots,\xi_D\in\mathbb{R}^d$ are sampled from $p(\xi)$. For a Gaussian kernel, they are drawn from a Normal distribution $\mathcal{N}(0,\sigma^2 I_d)$. Assuming that an explicit feature map is given, we formulate a two layer hybrid architecture as follows (see also Fig. 3):

$$\begin{aligned} h_1 &= W_1 x + b_1, \\ h_2 &= \hat{\varphi}(h_1) \\ s &= W_2 h_2 + b_2, \end{aligned} \tag{11}$$

where $W_1 \in \mathbb{R}^{d_1 \times d}$ and $W_2 \in \mathbb{R}^{Q \times d_2}$ are weight matrices and the bias vectors are denoted by b_1 and b_2.

Equivalently, one can re-write (11) as follows:

$$s = W_2 \hat{\varphi}(W_1 x + b_1) + b_2. \tag{12}$$

which makes the exiting connections to the standard neural networks architecture more clear. Here, the dimensions of the hidden variables h_1 and h_2 are user defined parameters. Here we assume that the input data are first projected to a d_1 dimensional space in the first layer followed by a d_2 dimensional space in the second layer. The formulation of the proposed method is as follows:

$$\min_{W_1, W_2, b_1, b_2} J(W_1, W_2, b_1, b_2) = \frac{\gamma}{2} \sum_{j=1}^{2} Tr(W_j W_j^T) + \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i). \tag{13}$$

The cost function is composed of regularization terms and the loss function $L(\cdot)$. Any misclassification loss function can be employed in the misclassification loss layer.

Here, in order to have probabilistic membership assignments to each instance, the cross-entropy loss function (also known as softmax function) is used which equips the results with a probabilistic interpretation by minimizing the negative log likelihood of the correct class. Let us denote the class scores for a single instance x_i as s_i^ℓ for $\ell = 1, \ldots, Q$. Then the cross-entropy loss for this instance can be calculated as:

$$L(x_i, y_i) = -\log\left(\frac{\exp(s_i^{y_i})}{\sum_{j=1}^{Q} \exp(s_i^j)}\right). \tag{14}$$

Here s_i^j denotes the score that is assigned to jth class for the instance x_i. Here y_i is the true class membership for the instance x_i. As can be seen from expression (14), the softmax classifier can be interpreted as the normalized probability assigned to the correct label y_i given the instance x_i. The first two terms in the objective function are regularization terms over the model parameters. The last term in (13) aims at minimizing the negative log likelihood of the correct class. The (stochastic) gradient descent algorithm can be employed to train the model in (13). Some theoretical analysis regarding the approximation quality of the Random Fourier features (RFF) for shift-invariant kernels and their derivatives can be found in [32]. The pseudocode of our approach is described in Algorithm 1 . The stopping criterion is when the residual loss function is less than a threshold $\epsilon = 10^{-3}$ or the maximum number of iterations is reached. After obtaining the model parameter W_1 and W_2, the score variable for the test point x_{test} can be computed as follows:

$$s_{\text{test}} = W_2 \, \tilde{\varphi}(W_1 x_{\text{test}} + b_1) + b_2. \tag{15}$$

The final class label for the test point x_{test} is computed as follows:

$$\hat{y}_{\text{test}} = \underset{\ell=1,\ldots,Q}{\text{argmax}}(s_{\text{test}}). \tag{16}$$

3.2. Deep hybrid model

As it has been shown in the literature, in many tasks, deep models with several staking nonlinear layers perform better than shallow models and are able to better learn the complex hierarchical representations of the given dataset. In particular, deep convolutional neural networks models are the state-of-the-art methods for learning the abstract representations of the labeled images [9,33]. It is possible to use the introduced hybrid model (11) as the first building block for exploring even deeper models. Here we develop a deeper model by staking the hybrid model (11) following the architecture shown in Fig. 4 where the input data passes through two linear and nonlinear layers in total before reaching the fully connected layer which is attached to the output.

Algorithm 1: Deep Hybrid Neural-Kernel Networks.

```
Input: Training data set $\mathcal{D}$, regularization constants $\gamma_1, \in \mathbb{R}^+$,
       Kernel parameter, learning rate $\eta$ and test set
       $\mathcal{D}^{\text{test}} = \{x_i\}_{i=1}^{n_{\text{test}}}$.
Output: Class label for the test instances $\mathcal{D}^{\text{test}}$.

1 Initialize the model parameters $\theta = [W_1, W_2, b_1, b_2]$.
2 while the stopping criterion is not satisfied do
3     Evaluate the gradient of the cost function w.r.t model
      parameters.
4     Make a gradient step and update the model
      parameters:$\theta^{\text{new}} = \theta - \eta \nabla_\theta J(\theta)$.
5     $\theta \leftarrow \theta_{\text{new}}$.
6 return $W$
7 Compute the score variable $s$, for the test instances in $\mathcal{D}^{\text{test}}$
  using (15).
8 Predict the test labels using (16).
```

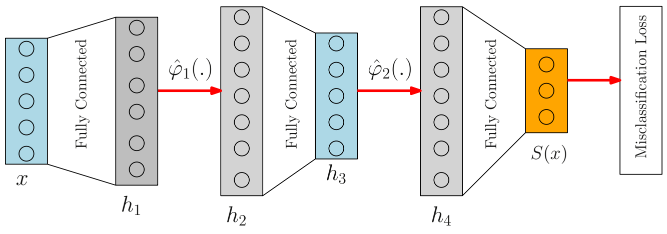

Fig. 4. Deep hybrid neural kernel network architecture.

One can formulate the stacked hybrid deep neural kernel model as follows:

$$\begin{aligned} h_1 &= W_1 x + b_1, \\ h_2 &= \hat{\varphi}_1(h_1) \\ h_3 &= W_2 h_2 + b_2, \\ h_4 &= \hat{\varphi}_2(h_3) \\ s &= W_3 h_4 + b_3, \end{aligned} \tag{17}$$

where $x \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{d_1 \times d}$, $W_2 \in \mathbb{R}^{d_3 \times d_2}$ and $W_3 \in \mathbb{R}^{Q \times d_4}$ are weight matrices and the bias vectors are denoted by b_1, b_2 and b_3. The dimension of the hidden layers h_1, h_2, h_3 and h_4 are denoted by d_1, d_2, d_3 and d_4 respectively. One can equivalently re-write (17) as follows:

$$s = W_3 \hat{\varphi}_2(s_1) + b_3, \tag{18}$$

where s_1 denotes the output of the two-layer hybrid model introduced in (12). The optimization problem corresponding to the proposed stacked hybrid model can be formulated as follows:

$$\min_{\{W_i\}_{i=1}^3, \{b_i\}_{i=1}^3} J(\{W_i\}_{i=1}^3, \{b_i\}_{i=1}^3) = \frac{\gamma}{2} \sum_{j=1}^{3} Tr(W_j W_j^T) + \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i). \tag{19}$$

The role of the regularization terms in (19) is to avoid overfitting by keeping the weights of the model small. The regularization parameter γ controls the relative importance given to the regularization terms. Here $L(\cdot, \cdot)$ is the cross-entropy loss defined as previously in (14).

Table 1
Dataset statistics.

Dataset	# instances	# attributes	# classes
Australian	690	14	2
Spambase	4597	57	2
Sonar	208	60	2
Titanic	2201	3	2
Monk2	432	6	2
Balance	625	4	3
Madelon	2600	500	2
USPS	9298	256	10
Digit-MultiF1	2000	240	10
Digit-MultiF2	2000	216	10
CNAE-9	1080	856	9
Large scale data			
Magic	19,020	10	2
Covertype	581,012	54	3
SUSY	5,000,000	18	2

Remark 1. Note that choosing γ properly is important as too low values result in the effect of the regularizer term to be neglected. On the other hand, for too high values the optimal model will set all the weights to zero.

It should be noted that stacking different hybrid layers can potentially bring more flexibility to the model as the new representation of the data can be learned in multiple levels corresponding to different scales in terms of feature mapping. However, one also requires more training times and possibly more data to get the most out of these types of models.

In training the stacked model, we will take advantage of the previously learned weights in model (13). More precisely, we transfer the learned weights of model (13) to the new model (17) and will keep it unchanged. In this way, not only the training computation times gets reduced as there will be less number of parameters to be learned but also the stacked model benefits from the previously learned weights to better optimize and learn the nonlinear decision boundaries. Alternatively, one could also use the learned weights of model (13) for the initialization of the first two layers of the stacked model and fine tune them while learning the weights of the remaining layers in (17). Here, we start by first obtaining the optimal model parameters of the two-layer hybrid neural kernel networks by solving (13). Then we cut-off the very last fully connected layer as well the Softmax layer from model (11) and build the new stacked model, see Fig. 4. When learning the parameters of the new model, the first wights of the first two layers are not trained. This process can in particular be of more interest when analyzing large scale datasets with the complex underlying non-linearity.

After obtaining the model parameter $\{W_i\}_{i=1}^3$ and $\{b_i\}_{i=1}^3$ the score variable for the test point x_{test} can be computed as follows:

$$\begin{aligned} h_3 &= W_2\,\tilde{\varphi}_1(W_1 x_{\text{test}} + b_1) + b_2, \\ s_{\text{test}} &= W_3\,\tilde{\varphi}_2(h_3) + b_3. \end{aligned} \tag{20}$$

The final class label for the test point x_{test} is computed as follows:

$$\hat{y}_{\text{test}} = \underset{\ell=1,\ldots,Q}{\text{argmax}}(s_{\text{test}}). \tag{21}$$

4. Experimental results and discussion

In this section experimental results on several real-life datasets taken from UCI machine learning repository [34] and KEEL-datasets [35] are reported. All the experiments are performed on a laptop computer with Intel Core i7 CPU and 16 GB RAM under Matlab 2014a.

The descriptions of the used datasets can be found in Table 1. Here we provide the information concerning some of these datasets and for the remaining ones, one may refer to the following links: http://archive.ics.uci.edu/ml/index.php and http://sci2s.ugr.es/keel/datasets.php. The Multiple Features (i.e., Digit-MultiF1 and Digit-MultiF2 in Table 1) datasets contains ten classes, (0–9), of handwritten digits with 200 images per class, thus for a total of 2000 images. Originally these digits are represented in terms of six different types of features. The features that are considered here are profile correlations (216 dimensional) and pixel average (240 dimensional). The USPS dataset is a handwritten digit dataset that contains 9298, 16×16 handwritten digit images in total. The CNAE-9 is a highly sparse dataset containing 1080 documents of free text business descriptions of Brazilian companies categorized into a subset of 9 categories. The supersymmetric particles (SUSY) dataset is a benchmark classification where the task is to distinguish between a process where new supersymmetric particles (SUSY) are produced, leading to a final state, in which some particles are detectable and others are invisible to the experimental apparatus, and a background process with the same detectable particles but fewer invisible particles and distinct kinematic features [36]. This benchmark problem is currently of great interest to the field of high-energy physics, and there is a strong effort in the literature to build high-level features which can aid in the classification task.

Almost in all the experiments, the given dataset is randomly partitioned to 80% training and 20% test sets respectively. The performance of the proposed deep hybrid model is compared with that of the shallow LS-SVM with implicit and explicit feature mapping, as well as multilayer perceptron with a comparable number of layers. In the LS-SVM framework, in the case of implicit feature mapping the problem is solved in the dual whereas when an explicit feature mapping is used one solves the problem in the primal (suitable for large scale data, as the complexity of the optimization problem grows linearly with the number of data points). In our experiments, in order to have a fair comparison, the dimension of the explicit random Fourier feature in both deep and shallow models are kept the same. The proposed two-layer and stacked layers hybrid model resemble the neural networks architecture with one and two hidden layers respectively. Therefore, we also compare our model with the standard feedforward artificial neural networks architectures defined as follows:

- One hidden layer (One-layer): Input $\rightarrow$ Fully Connected $\rightarrow$ ReLU activation $\rightarrow$ Fully Connected $\rightarrow$ Softmax.
- Two hidden layers (Two-layer): Input $\rightarrow$ Fully Connected $\rightarrow$ ReLU activation $\rightarrow$ Fully Connected $\rightarrow$ ReLU activation $\rightarrow$ Fully Connected $\rightarrow$ Softmax.

For the sake of a fair comparison, the dimension of the hidden layers in the above-mentioned network structures is kept the same as the one used in the proposed deep hybrid model. Comparing Fig. 4 with the classical neural networks architecture, implies that the explicit feature mapping in the hybrid model is acting as a nonlinear activation function as in the neural networks. Therefore it is interesting to explore its impact on the accuracy and the training computation times of the hybrid model compared to those of the classic non-hybrid neural networks architecture.

The parameters of the proposed deep hybrid model are the dimension of the fully connected layers and the explicit feature maps. In addition, in the case of the RBF kernel, the variance of the normal distribution from which one constructs the random Fourier features. Some of the existing methods for hyper-parameter tuning are for instance standard grid-search, random search [37] and Bayesian Optimization [38]. Following the lines of [37], we adopted the random search strategy for tuning the hyper-parameters of the proposed deep hybrid model as well as the feedforward neural networks. As compared to grid search, random search finds better models by effectively searching a larger and more promising con-

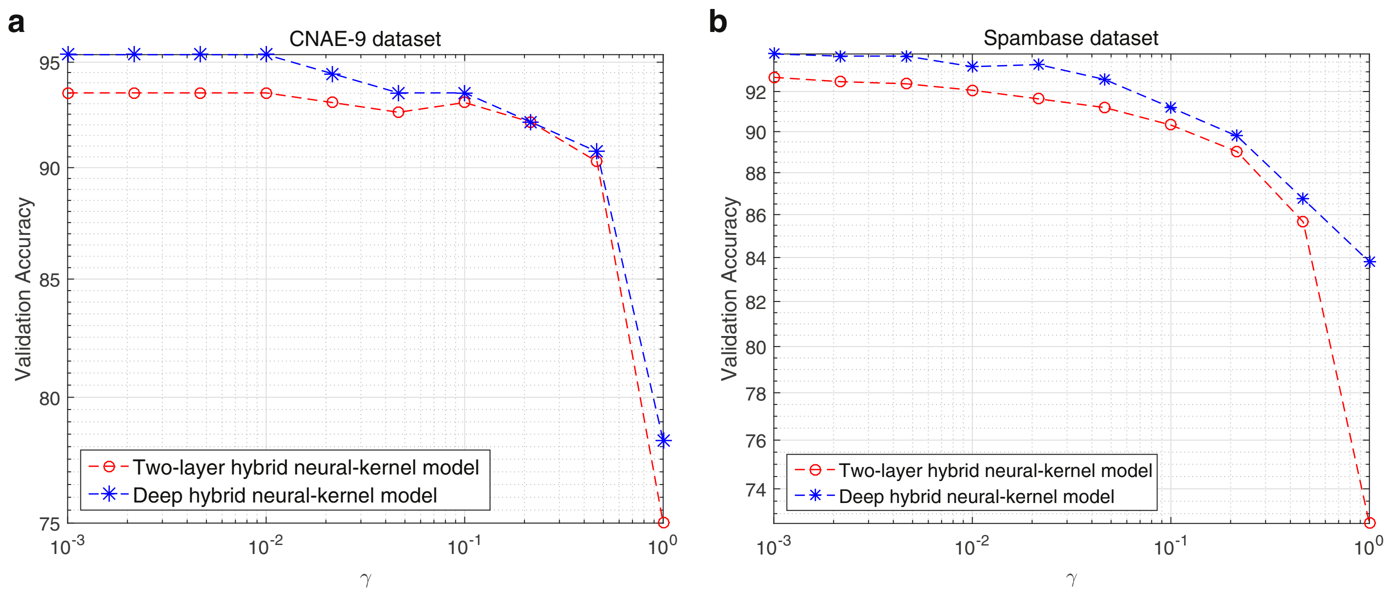

Fig. 5. (a) The validation accuracy of the two-layer hybrid neural-kernel model as well as deep hybrid neural-kernel model corresponding to different regularization parameters for CNAE9 dataset. (b) The validation accuracy of the two-layer hybrid neural-kernel model as well as deep hybrid neural-kernel model corresponding to different regularization parameters for Spambase dataset.

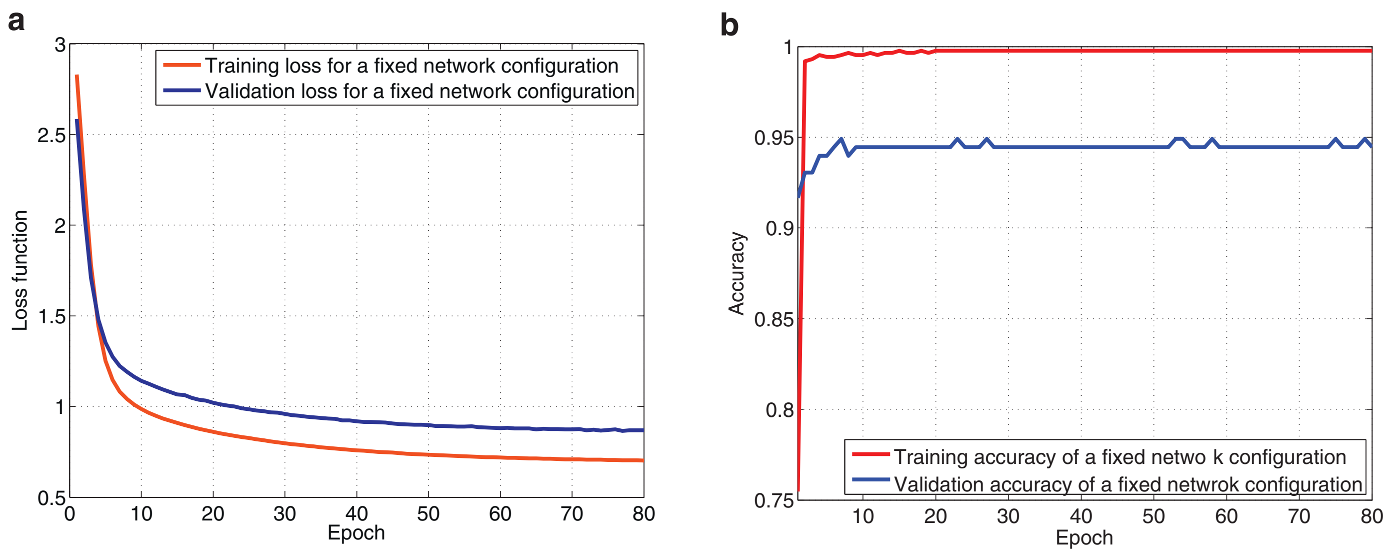

Fig. 6. CNAE9 dataset. (a) The training and validation loss for corresponding to the stacked deep hybrid model with $[h_1, h_2, h_3, h_4] = [300, 400, 200, 100]$ and $[\sigma_1, \sigma_2 = [0.7, 0.8]]$ configuration settings. (b) The training and validation accuracy with the previously mentioned hybrid model configuration.

figuration space [37]. In our experiments, the ranges from which the dimensions of the middle layers are sought are $h_1 \in [100, 500]$, $h_2 \in [30, 400]$, $h_3 \in [100, 200]$ and $h_4 \in [100, 300]$.

The influence of the regularization parameter γ in the formulations (13) and (19) for the CNAE9 and Spambase datasets is displayed in Fig. 5. Based on these observations, we set $\gamma = 0.0001$ for all the experiments. In addition, the training and validation loss as well as the accuracy of the proposed deep hybrid neural-kernel network model with $[h_1, h_2, h_3, h_4] = [300, 400, 200, 100]$ and $[\sigma_1, \sigma_2 = [0.7, 0.8]]$ configuration settings is illustrated in Fig. 6 for the CNAE9 dataset.

The parameters of the LS-SVM model are the kernel bandwidth σ and regularization parameter γ which are tuned using a two step procedure which consists of Coupled Simulated Annealing (CSA) [39] initialized with 5 random sets of parameters for the first step and the simplex method [40] for the second step. The coupled simulated annealing belongs to a class of optimization methods based on Simulated Annealing (SA) algorithm, a global optimization approach, that can be used to solve unconstrained non-convex problems in continuous variables. The CSA class is characterized by a set of parallel SA processes coupled by their acceptance probabilities. The coupling is performed by a term in the acceptance probability function that is a function of the energies of the current states of all SA processes [39]. After CSA converges to some local minima we select the parameters that attains the lowest error and start the simplex procedure to refine our selection.

The obtained empirical results of the proposed model, shallow LS-SVM model and standard feedforward neural networks are tabulated in Table 2. In most of the cases studied here the proposed deep hybrid model improves the accuracy over the shallow LS-SVM structure model. In addition, in some of the cases, the deep hybrid model shows an improvement over the two-layer hybrid model

Table 2
The average accuracy of the proposed deep models, the shallow LS-SVM with implicit and explicit feature maps and non-hybrid classic feedforward neural networks on several real-life datasets.

Dataset	$D_{\text{train}}/D_{\text{test}}$	Deep hybrid model		Shallow LS-SVM		Neural networks	
		Two-layer	Stacked layers	Primal	Dual	One-layer	Two-layer
Australian	552/138	0.85 ± 0.01	0.87 ± 0.01	0.81 ± 0.01	0.83 ± 0.01	0.83 ± 0.02	0.85 ± 0.01
Sonar	167/41	0.75 ± 0.04	0.77 ± 0.04	0.69 ± 0.07	0.72 ± 0.03	0.71 ± 0.02	0.73 ± 0.01
Titanic	1761/440	0.78 ± 0.01	0.78 ± 0.02	0.77 ± 0.02	0.78 ± 0.01	0.76 ± 0.01	0.78 ± 0.02
Spambase	3678/919	0.91 ± 0.03	0.93 ± 0.01	0.84 ± 0.05	0.85 ± 0.03	0.90 ± 0.02	0.92 ± 0.02
Monk2	346/86	1.00 ± 0.00	1.00 ± 0.00	0.93 ± 0.05	0.95 ± 0.02	0.94 ± 0.02	0.96 ± 0.01
Balance	500/125	0.96 ± 0.01	0.97 ± 0.02	0.93 ± 0.02	0.94 ± 0.01	0.95 ± 0.01	0.97 ± 0.01
CNAE-9	864/216	0.93 ± 0.01	0.94 ± 0.02	0.90 ± 0.04	0.91 ± 0.03	0.91 ± 0.02	0.92 ± 0.01
Digit-MultiF1	1600/400	0.97 ± 0.01	0.98 ± 0.01	0.95 ± 0.03	0.96 ± 0.02	0.95 ± 0.02	0.96 ± 0.02
Digit-MultiF2	1600/400	0.96 ± 0.02	0.97 ± 0.02	0.95 ± 0.02	0.96 ± 0.01	0.96 ± 0.01	0.97 ± 0.02
Madelon	2080/520	0.59 ± 0.06	0.63 ± 0.05	0.55 ± 0.08	0.57 ± 0.03	0.57 ± 0.01	0.62 ± 0.01
USPS	2789/6509	0.96 ± 0.01	0.97 ± 0.01	0.95 ± 0.02	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01
Magic	15,216/3804	0.86 ± 0.02	0.86 ± 0.02	0.84 ± 0.01	0.84 ± 0.01	0.83 ± 0.01	0.85 ± 0.02
Covertype	464,810/116,202	0.85 ± 0.02	0.86 ± 0.01	0.78 ± 0.01	N.A	0.83 ± 0.02	0.85 ± 0.01
SUSY	4,000,000/1,000,000	0.80 ± 0.02	0.81 ± 0.03	0.78 ± 0.01	N.A	0.79 ± 0.02	0.80 ± 0.01

Note: "N.A" stands for Not Applicable due the large size of the dataset. In the last two columns, One-layer and Two-layer refer to a neural networks with one hidden layer and two hidden layers respectively.

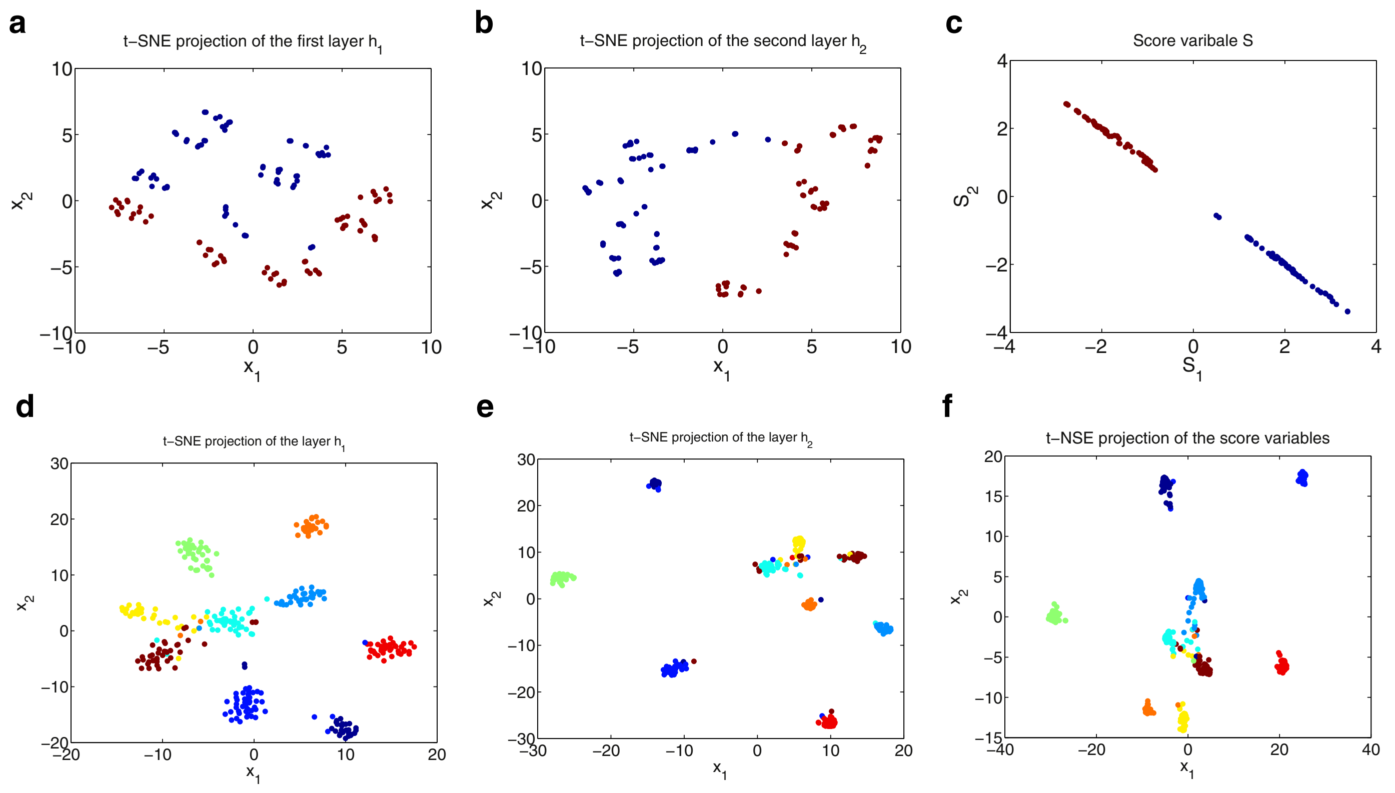

Fig. 7. Monk2 dataset. (a) and (b) t-SNE projections of the test data in the hidden layers h_1 and h_2. (c) The obtained score variables s, for the test data. CNAE9 dataset. (d), (e) and (f) t-SNE projections of the test data in the hidden layers h_1 and h_2 as well the score variables s. The different colors relate to the various classes.

as well as the feedforwad neural networks architecture. In all the datasets studied here, the two-layer neural networks outperforms the one-layer neural networks. As can be seen in Table 2, the two-layer hybrid model introduced in (11), i.e., a linear transformation of the input x followed by an explicit nonlinear embedding, can already achieve quite satisfactory results compared to its neural networks counterpart with one hidden layer (One-layer neural networks). In fact, although from architectural point of view, the two-layer hybrid model resembles one-layer neural networks, but in terms of accuracy its performance is more closer to two-layer neural networks model. In general, one can expect that when the underlying non-linearity of the data is complex, the deep hybrid model can potentially obtain a better decision boundary in the expense of more training computation times.

In Table (3), the training computation times of the proposed deep hybrid model and non-hybrid feedforward neural networks are given. From Table (3), one can observe that in general the hybrid model requires slightly more training times compared to standard neural networks, but on the other hand the accuracy is improved. It should also be noted that, in Table (3), the training times of the deep hybrid model is also slightly less than that of two-layer hybrid model. This is due the fact that in the deep hybrid model we utilize the learned weights of the two-layer hybrid model and also the dimension of the $h3$ and $h4$ hidden layers are chosen to be

Table 3
The training computation times (seconds) of the proposed deep hybrid model and non-hybrid classic feedforward neural networks.

Dataset	Deep hybrid model		neural networks	
	Two-layer	Stacked layers	One-layer	Two-layer
Australian	3.82	2.50	2.45	2.57
Sonar	1.17	1.15	1.11	1.03
Titanic	8.23	6.54	6.72	7.21
Spambase	16.02	14.04	15.03	15.47
Monk2	1.98	1.84	1.83	1.94
Balance	2.44	2.38	2.24	2.33
CNAE-9	4.60	4.34	3.79	3.42
Digit-MultiF1	7.63	6.02	6.09	6.26
Digit-MultiF2	6.75	6.37	6.19	7.04
Madelon	8.76	8.39	7.74	8.98
USPS	10.35	8.22	7.66	8.45
Magic	32.94	31.86	28.13	30.33
Covertype	483.10	464.05	439.70	445.32
SUSY	4100.06	4000.23	3767.08	3950.12

Note: In the last two columns, One-layer and Two-layer refer to a neural networks with one hidden layer and two hidden layers respectively.

less than that of $h1$ and $h2$ (refer to Eq. (17). Therefore the model has less parameters to learn in total.

The t-SNE visualization [41] of the obtained projections in the first and second layers as well as the score variables for the Monk2 and CNAE9 datasets are also depicted in Fig. 7 which shows the evolution of the features in the hybrid network. From Fig. 7, one can observe that the representation of the data is changing as it flows through the network. In particular, thanks to the deep structure of the network, there is an indication that the distribution of the classes has been better separated in the deeper representation layer, i.e., the learned representation right before the last layer has the best separability power. Ideally, one would expect that the new representation of the data can form clear clusters of classes. But due to the complex non-linear data manifold, one may encounter overlapping regions between new representation of each class.

5. Conclusions and future works

In this paper a new hybrid deep neural kernel network model is introduced. The similarities and differences between single artificial neural networks module and its kernel counterpart are discussed in detail. We showed how hybridization of kernel based models with explicit feature mapping and neural networks can lead to a new deep architecture, taking advantage of the two worlds. The proposed model is also considered as the first building block for deeper models using a stacking strategy. Our future work is devoted to studying several choices of misclassification loss functions as well as the extension of the proposed framework to semi-supervised learning with deep architecture.

Acknowledgments

This work was supported by • The Postdoctoral Fellowship of the Research Foundation-Flanders (FWO: 12Z1318N), • Reseach Council KUL: CoE PFV/10/002 (OPTEC), PhD grants Flemish Government; FWO: projects: G0A4917N, G.088114N. Siamak Mehrkanoon is a postdoctoral fellow of the Research Foundation-Flanders (FWO) working at the KU Leuven in Belgium. Johan Suykens is a full professor at the KU Leuven, Belgium.

References

[1] Y. Bengio, A. Courville, P. Vincent, Representation learning: review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828.
[2] Y. Bengio, Learning deep architectures for AI, Found. Trends® Mach. Learn. 2 (1) (2009) 1–127.
[3] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (Dec) (2010) 3371–3408.
[4] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[5] R. Salakhutdinov, G.E. Hinton, Deep Boltzmann Machines., in: Proceedings of the AISTATS, vol. 1, 2009, p. 3.
[6] G. Hinton, A practical guide to training restricted Boltzmann machines, Momentum 9 (1) (2010) 926.
[7] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML-10), 2010, pp. 807–814.
[8] S. Lawrence, C.L. Giles, A.C. Tsoi, A.D. Back, Face recognition: a convolutional neural-network approach, IEEE Trans. Neural Netw. 8 (1) (1997) 98–113.
[9] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[10] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[11] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[12] V. Vapnik, Statistical Learning Theory, Wiley, 1998.
[13] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.
[14] C.M. Bishop, N.M. Nasrabadi, Pattern Recognition and Machine Learning, vol. 1, Springer New York, 2006.
[15] Y. Cho, L.K. Saul, Kernel methods for deep learning, in: Proceedings of the Advances in Neural Information Processing Systems, 2009, pp. 342–350.
[16] J. Mairal, P. Koniusz, Z. Harchaoui, C. Schmid, Convolutional kernel networks, in: Proceedings of the Advances in Neural Information Processing Systems, 2014, pp. 2627–2635.
[17] A. Damianou, N. Lawrence, Deep gaussian processes, in: Proceedings of the Artificial Intelligence and Statistics, 2013, pp. 207–215.
[18] K. Cutajar, E.V. Bonilla, P. Michiardi, M. Filippone, Random feature expansions for deep gaussian processes, in: Proceedings of the International Conference on Machine Learning, 2017, pp. 884–893.
[19] Ö. Aslan, X. Zhang, D. Schuurmans, Convex deep learning via normalized kernels, in: Proceedings of the Advances in Neural Information Processing Systems, 2014, pp. 3275–3283.
[20] I. Steinwart, P. Thomann, N. Schmid, Learning with hierarchical gaussian kernels, arXiv:1612.00824 (2016), 1–16.
[21] L. Belanche, M. Costa-jussa, Bridging deep and kernel methods, in: Proceedings of the Twenty-Fifth European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), 2017, pp. 1–10., 2017, pp. 1–10.
[22] S. Mehrkanoon, A. Zell, J.A.K. Suykens, Scalable hybrid deep neural kernel networks, in: Proceedings of the Twenty-Fifth European Symposium on Artificial Neural Networks, Computational Intelligence and machine Learning (ESANN), 2017, pp. 17–22., 2017, pp. 17–22.
[23] J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, J. Vandewalle, Least Squares Support Vector Machines, World Scientific Pub. Co., Singapore, 2002.
[24] J.A.K. Suykens, J. Vandewalle, Training multilayer perceptron classifiers based on a modified support vector method, IEEE Trans. Neural Netw. 10 (4) (1999) 907–911.
[25] A.J. Smola, B. Schölkopf, Sparse greedy matrix approximation for machine learning, in: Proceedings of the Seventeenth ICML, Stanford, 2000, pp. 911–918.
[26] F.R. Bach, M.I. Jordan, Predictive low-rank decomposition for kernel methods, in: Proceedings of the Twenty-Second ICML, ACM, 2005, pp. 33–40.
[27] S. Kumar, M. Mohri, A. Talwalkar, Sampling methods for the Nyström method, J. Mach. Learn. Res. 13 (1) (2012) 981–1006.
[28] C. Williams, M. Seeger, Using the Nyström method to speed up kernel machines, in: Proceedings of the Fourteenth Annual Conference on Neural Information Processing Systems, 2001, pp. 682–688. EPFL-CONF-161322
[29] S. Mehrkanoon, J.A.K. Suykens, Large scale semi-supervised learning using KSC based model, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2014, pp. 4152–4159.
[30] S. Mehrkanoon, J.A.K. Suykens, Scalable semi-supervised kernel spectral learning using random fourier features, in: Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2016, pp. 1–8.
[31] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: Proceedings of the Advances in neural information processing systems, 2007, pp. 1177–1184.
[32] B. Sriperumbudur, Z. Szabó, Optimal rates for random fourier features, in: Proceedings of the Advances in Neural Information Processing Systems, 2015, pp. 1144–1152.
[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
[34] A. Asuncion, D.J. Newman, UCI machine learning repository, 2007, http://www.ics.uci.edu/~mlearn/MLRepository.html.
[35] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework., J. Mult. Valued Logic Soft Comput. 17 (2011).
[36] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, Nat. Commun. 5 (2014).

[37] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (Feb) (2012) 281–305.
[38] J. Snoek, H. Larochelle, R.P. Adams, Practical Bayesian optimization of machine learning algorithms, in: Proceedings of the Advances in Neural Information Processing Systems, 2012, pp. 2951–2959.
[39] S. Xavier-De-Souza, J.A.K. Suykens, J. Vandewalle, D. Bollé, Coupled simulated annealing, IEEE Trans. Sys. Man Cyber. Part B 40 (2) (2010) 320–335.
[40] J.A. Nelder, R. Mead, A simplex method for function minimization, Comput. J. 7 (4) (1965) 308–313.
[41] L. Van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (Nov) (2008) 2579–2605.

Siamak Mehrkanoon received the B.Sc. degree in pure mathematics and the M.Sc. degree in applied mathematics from the Iran University of Science and Technology, Tehran, Iran, in 2005 and 2007, respectively, and the Ph.D. degrees in numerical analysis and machine learning from Universiti Putra Malaysia, Seri Kembangan, Malaysia, and Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2011 and 2015, respectively. He was a Visiting Researcher with the Department of Automation, Tsinghua University, Beijing, China, in 2014, a Post-Doctoral Research Fellow with the University of Waterloo, Waterloo, ON, Canada, from 2015 to 2016, and a Visiting Post-Doctoral Researcher with the Cognitive Systems Laboratory, University of Tübingen, Tübingen, Germany, in 2016. He is currently an FWO Post-Doctoral Research Fellow with the Stadius Center for Dynamical Systems, Signal Processing and Data Analytics, KU Leuven. His current research interests include deep learning, neural networks, kernel-based models, unsupervised and semisupervised learning, pattern recognition, numerical algorithms, and optimization. Dr. Mehrkanoon received several fellowships for supporting his scientific studies, including post-Doctoral Mandate Fellowship from KU Leuven and Fund for Scientific Research FWO Flanders.

Johan A. K. Suykens (SM'05) was born in Willebroek Belgium, on May 18 1966. He received the M.S. degree in Electro-Mechanical Engineering and the Ph.D. Degree in Applied Sciences from the Katholieke Universiteit Leuven, in 1989 and 1995, respectively. In 1996 he has been a Visiting Postdoctoral Researcher at the University of California, Berkeley. He has been a Postdoctoral Researcher with the Fund for Scientific Research FWO Flanders and is currently a Professor (Hoogleraar) with KU Leuven. He is author of the books Artificial Neural Networks for Modelling and Control of Non-linear Systems (Kluwer Academic Publishers) and Least Squares Support Vector Machines (World Scientific), co-author of the book Cellular Neural Networks, Multi-Scroll Chaos and Synchronization (World Scientific) and editor of the books Nonlinear Modeling: Advanced Black-Box Techniques (Kluwer Academic Publishers) and Advances in Learning Theory: Methods, Models and Applications (IOS Press). In 1998 he organized an International Workshop on Nonlinear Modeling with Time-series Prediction Competition. He is a Senior IEEE member and has served as associate editor for the IEEE Transactions on Circuits and Systems (1997–1999 and 2004–2007) and for the IEEE Transactions on Neural Networks (1998–2009). He received an IEEE Signal Processing Society 1999 Best Paper (Senior) Award and several Best Paper Awards at International Conferences. He is a recipient of the International Neural Networks Society INNS 2000 Young Investigator Award for significant contributions in the field of neural networks. He has served as a Director and Organizer of the NATO Advanced Study Institute on Learning Theory and Practice (Leuven 2002), as a program co-chair for the International Joint Conference on Neural Networks 2004 and the International Symposium on Nonlinear Theory and its Applications 2005, as an organizer of the International Symposium on Synchronization in Complex Networks 2007 and a co-organizer of the NIPS 2010 workshop on Tensors, Kernels and Machine Learning. He has been awarded an ERC Advanced Grant 2011 and has been elevated IEEE Fellow 2015 for developing least squares support vector machines.