arXiv:1905.06518v2 [cs.SY] 27 Nov 2019

Efficient hinging hyperplanes neural network and its application in nonlinear system identification

Jun Xu [a], Qinghua Tao [b,c], Zhen Li [a], Xiangming Xi [a] , Johan A. K. Suykens [c] , Shuning Wang [b]

[a] *School of Mechanical Engineering and Automation, Harbin Institute of Technology, Shenzhen, 518055, China*

[b] *BNRist, Department of Automation, Tsinghua University, Beijing, 100084, China*

[c] *STADIUS, ESAT, KU Leuven, Leuven, 3001, Belgium*

Abstract

In this paper, the efficient hinging hyperplanes (EHH) neural network is proposed, which is basically a single hidden layer neural network. Different from the dominant single hidden layer neural networks, the hidden layer in the EHH neural network can be seen as a directed acyclic graph (DAG) and all the nodes in the DAG contribute to the output. It is proved that for every EHH neural network, there is an equivalent adaptive hinging hyperplanes (AHH) model, which was proposed based on the model of hinging hyperplanes (HH) and finds good applications in system identification. Analog to the proof for the AHH model, the universal approximation ability of the EHH neural network is provided. Different from other neural networks, the EHH neural network has interpretability ability, which can be easily obtained through its ANOVA decomposition (or interaction matrix). The interpretability can then be used as an indication for the importance of the input variables. The construction of the EHH neural network includes initial network generalization and parameter optimization (including the structure and weights parameters). A descent algorithm for searching the locally optimal EHH neural network is proposed and the worst-case complexity of the algorithm is also provided. The EHH neural network is applied in nonlinear system identification, the simulation results show that satisfactory accuracy can be achieved with relatively low computational cost, and at the same time, some insights into the regressors and the interactions among the regressors can be revealed.

Key words: Artificial neural networks; Hinging hyperplanes; Interpretability; Identification methods.

1 Introduction

Nonlinear system identification is an important procedure to predict future behavior of a dynamic system, to apply model-based control or to understand the physical insights. Basically, the identification problem is to infer relationships between the system output and its past input-output data. Collect a finite number of past inputs $u(k)$ and outputs $y(k)$ into the regressor $\varphi(k) = [y(k-1), \ldots, y(k-n_b), u(k), u(k-1), \ldots, u(k-n_a)]^T$, then the problem is to understand the relationship between the next output $y(k)$ and $\varphi(k)$ [21]. Since most of the practical systems in the real world are so complicated that the analytical models can not be established at low cost, the nonlinear black-box models are often used to depict the nonlinear system, of which nonlinear neural networks find wide applications [18,24]. Ever since its emergence, the neural network has gained a lot of popularity, especially when it was proved that with enough hidden nodes, a neural network model can approach almost all continuous systems at arbitrary precision [10]. However, the neural network also receives criticism as it lacks interpretability, meaning that using it as a black-box model to depict a nonlinear system can not reveal the intrinsic mechanism.

⋆ This paper was not presented at any IFAC meeting. Corresponding author Xiangming Xi.

Email addresses: xujunqgy@hit.edu.cn (Jun Xu), taoqh14@mails.tsinghua.edu.cn (Qinghua Tao), 1654995440@qq.com (Zhen Li), xixiangming@gmail.com (Xiangming Xi), johan.suykens@esat.kuleuven.be (Johan A. K. Suykens), swang@tsinghua.edu.cn (Shuning Wang).

The piecewise linear (PWL) neural network is a special kind of neural networks, which admits a linear relationship in each subregion. In PWL neural networks, the nonlinearity only comes from the PWL activation functions. For example, the fully connected neural network which employs the popular rectified linear units (ReLU), $\max\{0, x\}$, is a kind of PWL neural networks, since ReLU is exactly the basis function of the hinging hyperplanes (HH) model [3]. A second example goes to the maxout neural networks [9], which lays its foundation on the generalized hinging hyperplanes (GHH) model proposed in [25]. Both the HH model and GHH model belong to the family of continuous PWL representations and they differ in the sense of representability, i.e., the HH model can not represent all the continuous PWL functions in higher than 2 dimensions while the GHH model can. Besides, another member in the family is the adaptive hinging hyperplanes (AHH) model, which was proposed to eliminate the inefficiency caused by the gradient-based method in the identification of the HH and GHH models [29]. The AHH model is derived using a recursive partitioning procedure with a greedy strategy, i.e., by traversing all existing basis functions, the basis function that yields the largest decrease in the optimized criterion is added to the model in the traversal manner. The AHH model has been successfully applied in regression, optimization and dynamic system identification [12,28,8]. However, the application of the AHH model to problems with a higher dimension is limited by two reasons. The first is the identification speed, which is exponential with respect to the number of basis function [29], and the increase is more abrupt when the dimension is higher. The second owes to the redundancy and duplication in the calculation of the same factors that appear more than once in the basis functions, thus the AHH model can not be realized as a distributed representation, which fails to be accelerated by clusters of computers.

From the structural point of view, the HH, GHH and AHH models can be regarded as single hidden layer PWL neural networks, and the weighted sum of the nodes that represent the basis functions constitute the output of the neural network. Different from the existing neural networks that based on the HH and GHH models, we investigate a variant of the AHH structure that is essentially more extendable in applications. We name the proposed structure as the efficient hinging hyperplanes (EHH) neural networks for high-dimensional problems, which only employ the "min" and "max" activation functions. It is basically an **interpretable** one hidden layer PWL neural network, in which the hidden layer is modelled as a directed acyclic graph (DAG). Each EHH neural network corresponds to an AHH model. Moreover, the EHH representation has a distributed essence which the AHH representation fails to have.

The remainder of this paper is organized as follows: Section 2 gives a review of PWL models based on the model of HH. Section 3 describes the structure, the approximation ability as well as the interpretability of the EHH neural network, besides, the relationship between the AHH model and the EHH neural network is provided. Then the training of the EHH neural network is performed in Section 4, in which both the structure and the weights of the neurons are optimized. Section 5 illustrates the application of the EHH neural network to nonlinear system identification. Finally the paper ends with conclusions and future work in Section 6.

Notations. A vector $\mathbf{x}$ is written in bold, while a matrix is denoted by the capital bold, like $\mathbf{Z}$. The notation x_i denotes the i-th component in $\mathbf{x}$, while the notation $\mathbf{Z}_{ij}$ denotes the element in the i-th row, j-th column of $\mathbf{Z}$, and $\mathbf{Z}(:, i)$ denotes the i-th column in the matrix $\mathbf{Z}$. The superscript "T" denotes the transpose. The real scalar set is denoted by $\mathbb{R}$ while the real vector set in n dimension is written as $\mathbb{R}^n$. The notations var and σ denote the variance and standard deviation for a data vector, respectively. The italic capital, A, represents the nodes in the hidden layer, and D and C stand for the source nodes and intermediate nodes, respectively. The calligraphic capitals stands for sets.

2 Review of the HH family

The model of HH was first proposed by Breiman [3], and can be cast as a linear combination of basis functions, i.e.,

$$f_{\mathrm{HH}} = \sum_{m=1}^{M} a_m B_m(\mathbf{x}), \tag{1}$$

in which $\mathbf{x} \in \mathbb{R}^n$ is the input and $f_{\mathrm{HH}} \in \mathbb{R}$ is the output. In the HH model, the basis function $B_m(x)$ takes the form of

$$\max\{0, \ell_m(\mathbf{x})\}, \tag{2}$$

in which $\ell_m(\mathbf{x})$ is a linear (affine) function [3] (Here we use "linear" to represent both linear and affine functions). The expression (2) is called a hinge. It is worthy to note that the commonly used ReLU activation function in deep

networks is a special kind of the hinge by restricting the linear function $\ell_m(\mathbf{x})$ to be univariate affine. The model of HH has been applied quite successfully in regression, classification [11], and dynamic system identification [4].

As the model of HH can not represent some continuous PWL functions in 2 and higher dimensions, the model of GHH was proposed to generalize the model of HH by replacing the hinge function (2) in (1) with the generalized hinge function [25],

$$\max\{0, \ell_1(\mathbf{x}), \ldots, \ell_{k_n}(\mathbf{x})\} \tag{3}$$

with $k_n \leq n$ and $\ell_k(\mathbf{x}), k = 1, \ldots, k_n$ are affine. It has been proved that the GHH model can represent any continuous PWL functions in any dimension. And it is noted that the activation function employed in the popular maxout network is the generalized hinge [9]. In the training of the HH and GHH model, the structural parameters, like the number of basis functions as well as the number of linear functions in each generalized hinge, can not be changed while optimizing the weights parameter $a_1, \ldots, a_M$.

The model of AHH is a PWL representation which can be also seen as a linear combination of basis functions, just like (1), and the basis function takes the form of

$$B_m(\mathbf{x}) = \min\{g_1(\mathbf{x}), \ldots, g_K(\mathbf{x})\} \tag{4}$$

with

$$g_k(\mathbf{x}) = \max\{0, s_k(x_{v_k} - \beta_k)\},$$

$s_k = \pm 1$, $x_{v_1}, \ldots, x_{v_k}$ are indices of the input variables, and $\beta_k \in [0, 1)$ are predefined distinguished scalars, $k = 1, \ldots, K$. It is proved in [29] that the equation (4) can be obtained by restricting $\ell_k(\mathbf{x})$ in (3) to be univariate linear. Different from the HH and GHH models, the AHH model is trained using a recursive partitioning procedure, which is similar to the training of multivariate adaptive regression splines (MARS). The training of the AHH model includes a forward stepwise and a backward stepwise procedures, in which basis functions are added or deleted according to the greedy strategy. It is noted that the structure of the AHH model is dynamically adjusting during the training process. It has been shown in [29] that the computational complexity increases exponentially with respect to the number of basis functions, and the increase becomes more abrupt when the dimension grows higher and the sample data size is larger.

3 Efficient hinging hyperplanes neural network

In this section, we focus on the construction of the EHH neural network, and its properties such as the approximation ability and interpretability. Besides, the relationship between the AHH model and the EHH neural network is shown.

3.1 Network Structure

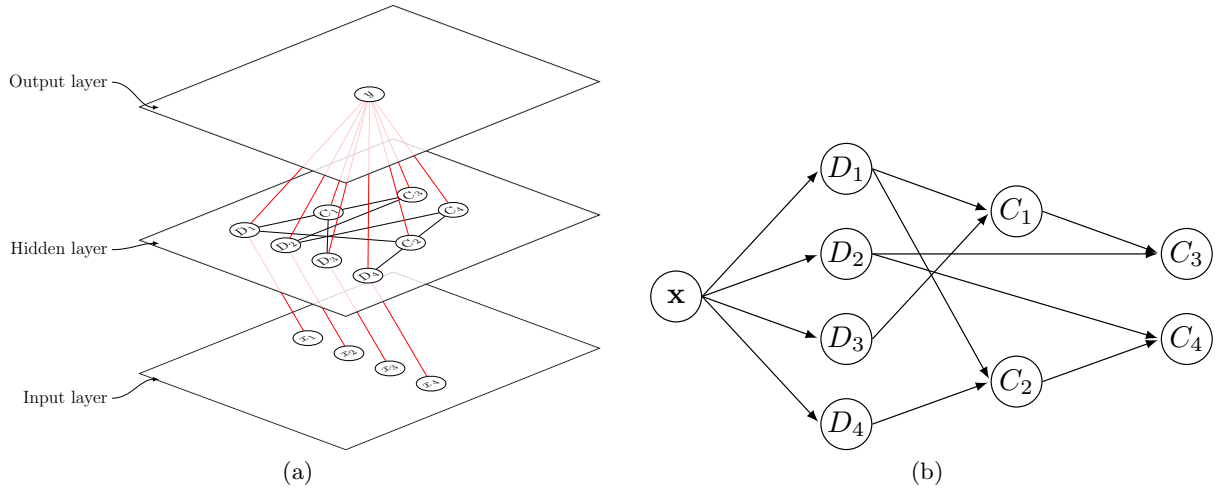
Fig. 1(a) shows a typical EHH neural network. It can be seen as a single hidden layer PWL neural network, which includes an input layer, a hidden layer and an output layer. In the following discussions, we will describe the connections of each 2 subsequent layers. Besides, the hidden layer is described using a directed acyclic graph (DAG), and the connections of nodes in the DAG is explained in detail.

3.1.1 Pre-processing

Generally speaking, the input variables for the neural networks may differ from each other in the ranges of candidate values. For example, the variables may involve the temperature and the humidity of the environment, which are not in the same units and range of possible values. Thus, in order to avoid computation deficiency and severe variations in the parameter space, we are supposed to pre-process the input data before sending to the hidden layer. Specifically, assume the sampled data is $(\tilde{\mathbf{x}}(k), y(k))_{k=1}^{N_s}$, this is done by normalizing each of the original input variables independently, i.e.,

$$x_i(k) = \frac{\tilde{x}_i(k) - \min(\tilde{\mathbf{x}}_i)}{\max(\tilde{\mathbf{x}}_i) - \min(\tilde{\mathbf{x}}_i)},$$

in which $\tilde{\mathbf{x}}_i = [\tilde{x}_i(1), \ldots, \tilde{x}_i(N_s)]^T$. After the standardization process, each $x_i(k) \in [0, 1], \forall i = 1, \ldots, n, k = 1, \ldots, N_s$. Hereafter in the paper, the input variables refer to be normalized.



(a) (b)

Fig. 1. Illustration of the EHH neural network. (a) The structure of a typical EHH network. (b) Local view of the hidden layer: DAG.

3.1.2 Input and hidden layer connection

Let $\mathbf{x} = [x_1, \ldots, x_n]^T$ be the pre-processed input variables. Since the hidden layer is essentially a DAG, we define two kinds of nodes (neurons) in the hidden layer, i.e., the source nodes and the intermediate nodes. The source nodes are the nodes that only accept inputs from the input layer, meaning that there are edges directed from the input nodes to the source nodes. Other than the source nodes, we define the intermediate nodes, which can accept inputs from the source nodes or other intermediate nodes in the hidden layer, i.e., there are edges directed from other nodes to the intermediate nodes.

In the EHH neural network, the source nodes in the hidden layer are designed to have the "max" activation function on a single variable in the inputs. Therefore, the output of a source node can be expressed as

$$\mathrm{nn}(\mathbf{x}) = \max\{0, x_v - \beta_v\},$$

where v is the index of the input variable, and β_v is the offset. As is shown in Fig. 1(b), the nodes $\mathrm{D}_1, \ldots, \mathrm{D}_{N_d}$ ($N_d = 4$) are the source nodes.

In general, since the EHH neural network employs the univariate activation function, we can express the output of the source nodes as follows,

$$\begin{aligned}
&\mathrm{nn}_{D_1}(\mathbf{x}) = \max\{0, x_1\}, \qquad \mathrm{nn}_{D_2}(\mathbf{x}) = \max\{0, x_1 - \beta_{11}\}, \qquad \ldots, \ \mathrm{nn}_{D_q}(\mathbf{x}) = \max\{0, x_1 - \beta_{1,q-1}\}, \\
&\vdots \\
&\mathrm{nn}_{D_{(n-1)q+1}}(\mathbf{x}) = \max\{0, x_n\}, \ \mathrm{nn}_{D_{(n-1)q+2}}(\mathbf{x}) = \max\{0, x_n - \beta_{n1}\}, \ \ldots, \ \mathrm{nn}_{D_{nq}}(\mathbf{x}) = \max\{0, x_n - \beta_{n,q-1}\}
\end{aligned} \tag{5}$$

in which $\beta_{i1}, \ldots, \beta_{i,q-1}$ are the predefined bias in each dimension. For example, we can take $\beta_{i1}, \ldots, \beta_{i,q-1}$ as the q-quantiles in the i-th dimension. Hence in total we have $N_d = nq$ source nodes.

3.1.3 Connection in the hidden layer

Besides the source nodes, nodes in the hidden layer that are not directly connected to the nodes in the input layer are called the intermediate nodes, such as the node C_j, $j = 1, \ldots, N_c$, in Fig. 1(b). In the hidden layer of the EHH neural network, the intermediate nodes are generated and ordered sequentially, and are designed to take inputs from exactly two other nodes in the hidden layer, and take the "min" activation function.

In order to construct the EHH neural network systematically, we establish the connection in the hidden layer involving intermediate nodes as follows.

Assumption 1 *Given three nodes $A_{j_1}, A_{j_2}, A_{j_3}$ in the hidden layer in an EHH neural network, if there are edges directed from A_{j_1} and A_{j_2} to A_{j_3}, respectively, then the following two rules should be followed,*

Rule 1 $A_{j_1}, A_{j_2} \in \mathcal{J} \triangleq \{D_p \mid p = 1, \ldots, N_d\} \cup \{C_t \mid t < j_3\}$;
Rule 2 $\mathcal{S}_{A_{j_1}} \cap \mathcal{S}_{A_{j_2}} = \emptyset$, *where* $\mathcal{S}_{A_{j_1}}$ *and* $\mathcal{S}_{A_{j_2}}$ *are the sets of indices of input variables that appeared in the nodes* A_{j_1} *and* A_{j_2}, *respectively.*

In this way, we can express the output of A_{j_3} as

$$\mathrm{nn}_{A_{j_3}}(\mathbf{x}) = \min\{\mathrm{nn}_{A_{j_1}}(\mathbf{x}), \mathrm{nn}_{A_{j_2}}(\mathbf{x})\}. \tag{6}$$

In order to depict the connections in the hidden layer of an EHH neural network, we introduce a new concept as follows.

Definition* 1** *Given an EHH neural network that has M neurons in the hidden layer, the matrix $\mathbf{A_d} = [\mathbf{A_d}]_{ij} \in \mathbb{R}^{M\times M}$ is called the* ***adjacency matrix *of the EHH neural network, if*

- $\mathbf{A_d}_{ij} = 1$ *indicates that there is an edge directed from the i-th node to the j-th node in the hidden layer, and* $\mathbf{A_d}_{ij} = 0$ *indicates otherwise.*

Specifically, we take the EHH neural network shown in Fig. 1(b) for example. Suppos that the source nodes $D_1, \ldots, D_4$ contain different input variables. Without loss of generality, we can denote the output of the four sources nodes as,

$$\mathrm{nn}_{D_i}(\mathbf{x}) = \max\{0, x_i - \beta_i\}, i = 1, 2, 3, 4. \tag{7}$$

According to the connections involving the intermediate nodes $C_1, \ldots, C_4$ in Fig.1(b), we have

$$\begin{aligned}
&\mathrm{nn}_{C_1}(\mathbf{x}) = \min\{\mathrm{nn}_{D_1}(\mathbf{x}), \mathrm{nn}_{D_3}(\mathbf{x})\}, \mathrm{nn}_{C_2}(\mathbf{x}) = \min\{\mathrm{nn}_{D_1}(\mathbf{x}), \mathrm{nn}_{D_4}(\mathbf{x})\} \\
&\mathrm{nn}_{C_3}(\mathbf{x}) = \min\{\mathrm{nn}_{D_2}(\mathbf{x}), \mathrm{nn}_{C_1}(\mathbf{x})\}, \mathrm{nn}_{C_4}(\mathbf{x}) = \min\{\mathrm{nn}_{D_2}(\mathbf{x}), \mathrm{nn}_{C_2}(\mathbf{x})\}.
\end{aligned}$$

The index set for each hidden nodes can be written as

$$\begin{aligned}
&\mathcal{S}_{D_1} = \{1\}, \quad \mathcal{S}_{D_2} = \{2\}, \qquad \mathcal{S}_{D_3} = \{3\}, \quad \mathcal{S}_{D_4} = \{4\}, \\
&\mathcal{S}_{C_1} = \mathcal{S}_{D_1} \cup \mathcal{S}_{D_3} = \{1, 3\}, \quad \mathcal{S}_{C_2} = \mathcal{S}_{D_1} \cup \mathcal{S}_{D_4} = \{1, 4\}, \\
&\mathcal{S}_{C_3} = \mathcal{S}_{D_2} \cup \mathcal{S}_{C_1} = \{1, 2, 3\}, \; \mathcal{S}_{C_4} = \mathcal{S}_{D_2} \cup \mathcal{S}_{C_2} = \{1, 2, 4\}.
\end{aligned} \tag{8}$$

Then the adjacency matrix $\mathbf{A_d}$ that describes the connections of nodes in the DAG for the EHH network shown in Fig.1(a) can be written as

$$\mathbf{A_d} = \left[\begin{array}{c|cccc:cc:cc}
 & D_1 & D_2 & D_3 & D_4 & C_1 & C_2 & C_3 & C_4 \\ \hline
D_1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
D_2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
D_3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
D_4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hdashline
C_1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
C_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hdashline
C_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right] \tag{9}$$

The following remarks give further insights into the construction of the connections among the neurons in the hidden layer of the EHH neural network.

***Remark* 1** *There should be only two "1"s in a column in* $\mathbf{A_d}$ *as each neuron has exactly two inputs as is designed, and there may be multiple "1"s in a row, as each neuron may be inputs to more than 1 neuron.*

***Remark* 2** *The input variables involved in each term of a single neuron are ensured to be distinct, which guarantees that none of the terms can be removed without changing the output of the neuron, i.e., the parameters in the neuron are irredundant. For example, without loss of generality, if we include in one neuron two identical input variables, such as,*

$$\mathrm{nn}_{C_1}(\mathbf{x}) = \min\{\max\{0, x_i - \beta_{k_i}\}, \max\{0, x_i - \beta_{k_j}\}\},$$

where $\beta_{k_i} < \beta_{k_j}$*, we can rewrite* $\mathrm{nn}_{C_1}(\mathbf{x})$ *as*

$$\mathrm{nn}_{C_1'}(\mathbf{x}) = \min\{\max\{0, x_i - \beta_{k_j}\}\},$$

which implies that the parameters in the neuron C_1 *are redundant and can be simplified to the neuron* C_1'*. Therefore, the number of input variables contained in each neuron will not exceed the input dimension,* n*.*

3.1.4 Hidden and output layer Connection

In the EHH neural network, both the source and intermediate nodes are designed to be connected to the nodes in the output layer, as Fig. 1(a) shows. Therefore, the output of the EHH neural network takes the form of the weighted sum of all neurons (with a bias) in the hidden layer, i.e.,

$$f_{\mathrm{EHH}}(\mathbf{x}) = \sum_{k=1}^{M} \alpha_k \mathrm{nn}_{A_k}(\mathbf{x}) + \alpha_0, \tag{10}$$

in which A_k is the neuron in the hidden layer, $\mathrm{nn}_{A_k}(\mathbf{x})$ and $\alpha_k \in \mathbb{R}$ denote the output and coefficient of the neuron A_k, $k \in \{1, \ldots, M\}$, and α_0 is the constant bias. The coefficients $\alpha_0, \alpha_1, \ldots, \alpha_M$ are called **weights of the EHH neural network**.

***Remark* 3** *When the EHH neural network is configured to have* $N_d = n$ *source nodes and* $N_c = 0$ *intemediate nodes, and the offsets* β_i *are set to be* 0*,* $\forall i$*, we have*

$$\mathrm{nn}_{D_i}(\mathbf{x}) = x_i, i = 1, \ldots, n,$$

since $\mathbf{x} \in [0, 1]^n$*, and the EHH neural network is degenerated to a linear approximator,*

$$f_{\mathrm{EHH}}(\mathbf{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i. \tag{11}$$

3.2 EHH neural network versus AHH model

The EHH neural networks are also within the HH family, in particular, we will explain that the EHH neural network can find the equivalent AHH model. It has been shown in [29] that the AHH model is a tree structure, which is obtained by recursively partitioning of the domain coordinatewisely. Each partition of the domain (say x_{v_i} at $\beta_{k_{v_i}}$) generates 2 new bases by taking the minimum of the current basis functions $B_1, \ldots, B_M$ and $\max\{0, x_{v_i} - \beta_{k_{v_i}}\}$ or $\max\{0, -(x_{v_i} - \beta_{k_{v_i}})\}$, i.e., $\min\{B_1, \ldots, B_M, \max\{0, x_{v_i} - \beta_{k_{v_i}}\}\}$ or $\min\{B_1, \ldots, B_M, \max\{0, -(x_{v_i} - \beta_{k_{v_i}})\}\}$. Then the backward stepwise procedure deletes redundant basis functions according to a greedy strategy.

We start from a simple example. Fig. 2 is a simple AHH tree structure, in which the circle denotes the root, and the squares denote the leaves. The first generation takes the form of $B_1 = \max\{0, x_1 - \beta_1\}, B_1' = \max\{0, -(x_1 - \beta_1)\}, B_2 = \max\{0, x_2 - \beta_2\}, B_2' = \max\{0, -(x_2 - \beta_2)\}, B_3 = \max\{0, x_3 - \beta_3\}, B_3' = \max\{0, -(x_3 - \beta_3)\}$ and $B_4 = \max\{0, x_4 - \beta_4\}, B_4' = \max\{0, -(x_4 - \beta_4)\}$.

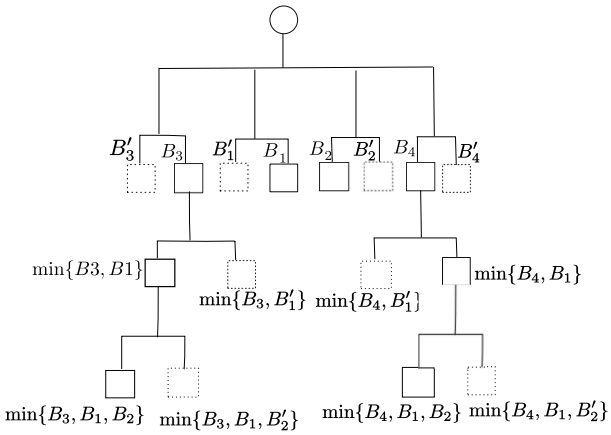


Fig. 2. A simple AHH.

It can be seen in Fig. 2 that the first generation of the AHH tree, i.e., $B_1, B_1', B_2, B_2', B_3, B_3', B_4, B_4'$ are derived by splitting the domain at $x_1 = \beta_1, x_2 = \beta_2$, $x_3 = \beta_3, x_4 = \beta_4$, respectively. The second generation of the AHH tree is obtained by partitioning the regions in which the basis function B_3 and B_4 are active ($x_3 \geq \beta_3$ and $x_4 \geq \beta_4$ respectively) at $x_1 = \beta_1$. Then for the third generation, the splitting occurs at $x_2 = \beta_2$. Suppose in this case the leaves containing terms $\max\{0, -(x_i - \beta_i)\}, i = 1, 2, 3, 4$ are all removed during the backward stepwise process.

In the AHH tree structure, the relationships between the fathers and daughters are clear. However, different daughters may contain the genes from the same father, say the daughters $\min\{B_3, B_1\}$ and $\min\{B_4, B_1\}$. And in the tree structure, no connections indicate this. Besides, this representation is not a distributed representation. For example, the calculation of $\min\{B_3, B_1\}$ does not use the information calculated for B_1, hence for this simple tree structure, the information of B_1 has to be calculated for 3 times.

It is not hard to verify the AHH tree shown in Fig. 2 and the EHH hidden layer shown in Fig. 1(b) are equivalent (Let nn_{D_i} correspond to $B_i, i = 1, 2, 3, 4$, respectively). From Fig. 2, we can see that the first generation of the AHH tree corresponds to the first column in the DAG. In the EHH network hidden layer, the interaction between B_1 and B_3 (nn_{D_1} and nn_{D_3}) is shown clearly, thus the information of B_1 and B_3 can be reused. This is especially useful when the network outputs need to be calculated many times, hence the EHH networks in Fig. 1(a) is a distributed representation.

It is noted that in the AHH tree, the term $\max\{0, -(x_v - \beta_{k_v})\}$ exists in the bases, while in the EHH neural network, the sign "-" does not appear. The following lemma explains the relationship of the 2 representations.

Lemma 2 *Every EHH neural network can find an equivalent AHH tree.*

PROOF. In the EHH networks, assume a node A_j receives inputs from the neurons A_{j_1}, A_{j_2}, i.e.,

$$\text{nn}_{A_j} = \min\{\text{nn}_{A_{j_1}}, \text{nn}_{A_{j_2}}\}.$$

The same rule applies to $\text{nn}_{A_{j_1}}$ and $\text{nn}_{A_{j_2}}$, thus finally we can rewrite nn_{A_j} as

$$\text{nn}_{A_j} = \min_{k \in \mathcal{K}_j} \{\text{nn}_{A_k}\} \tag{12}$$

where $\mathcal{K}_j$ are the index sets of the source nodes that have direct or indirect connections to nn_j. Compared with the AHH basis function in [29], which can be written as

$$B_m(\mathbf{x}) = \min_{k \in \mathcal{K}_m} \{\max\{0, s_{km}(x_{v_{km}} - \beta_{km})\}\}, \tag{13}$$

with $s_{km} = \pm 1$. By letting all the s_{km} to be 1, we know that the outputs of the EHH nodes can be written as equivalent AHH basis functions. As each node can find an equivalent AHH basis, every EHH neural network can find an equivalent AHH tree.

However, for an AHH tree, as the possible existence of $\max\{0, -(x_i - \beta_{k,i})\}$, there may be no equivalent EHH network. As the model of AHH can approximate any continuous functions in a compact domain, attention is paid to the approximation ability of the EHH neural networks.

3.3 Approximation ability of EHH neural networks

For the case of simplicity, we consider the approximation on the compact set $[0,1]^n$.

Theorem 3 *Let $f : [0,1]^n \to \mathbb{R}$ be a continuous function. Then for any $\varepsilon > 0$, there exists an EHH network f_{EHH}, such that for all $\mathbf{x} \in [0,1]^n$, we have*

$$\|f(\mathbf{x}) - f_{\text{EHH}}(\mathbf{x})\| < \varepsilon. \tag{14}$$

PROOF. Assume the source nodes of the EHH neural network are

$$\max\{0, x_1 - \beta_1\}, \ldots, \max\{0, x_1 - \beta_q\}, \ldots, \max\{x_n - \beta_1\}, \ldots, \max\{x_n - \beta_q\}$$

with $\beta_1 = 0, \beta_2 = 1/q, \ldots, \beta_q = (q-1)/q$, in which q is a positive integer and can be tuned.

For the intermediate nodes, the full connection strategy is employed, i.e., taking the minimum of any 2 existing neurons with different input variables to form a new neuron. Notice that this full connection strategy is different from "real" full connection, as some connections are duplicated or contradict Rule 2.

The resulting EHH neural network can be written as (arranged according to the number of input variables contained in a neuron)

$$\begin{aligned}
f_{\text{EHH}}(\mathbf{x}) = \alpha_0 &+ \sum_{k_1=1}^{n} \sum_{j_1=1}^{q} a_{j_1}^{(k_1)} \max\{0, x_{k_1} - \beta_{j_1}\} \\
&+ \sum_{k_1=1}^{n-1} \sum_{k_2=k_1+1}^{n} \sum_{j_1=1}^{q} \sum_{j_2=1}^{q} a_{j_1 j_2}^{(k_1 k_2)} \cdot \min\{\max\{0, x_{k_1} - \beta_{j_1}\}, \max\{0, x_{k_2} - \beta_{j_2}\}\} \\
&+ \cdots + \sum_{k_1=1}^{n-r+1} \cdots \sum_{k_r=k_1+r-1}^{n} \sum_{j_1=1}^{q} \cdots \sum_{j_r=1}^{q} a_{j_1 \ldots j_r}^{(k_1 \ldots k_r)} \cdot \min\{\max\{0, x_{k_1} - \beta_{j_1}\}, \cdots, \max\{0, x_{k_r} - \beta_{j_r}\}\} \\
&+ \cdots + \sum_{j_1=1}^{q} \cdots \sum_{j_n=1}^{q} a_{j_1 \ldots j_n} \cdot \min\{\max\{0, x_1 - \beta_{j_1}\}, \cdots, \max\{0, x_n - \beta_{j_n}\}\}.
\end{aligned} \tag{15}$$

The number of neurons (plus the constant neuron) is

$$1 + \binom{n}{1} \cdot q + \binom{n}{2} \cdot q^2 + \cdots + \binom{n}{r} \cdot q^r + \cdots + \binom{n}{n} q^n = (q+1)^n. \tag{16}$$

Fig. 3 illustrates the full connection strategy in the EHH neural network with $n = 3$ and $q = 2$, in which the neurons including the same number of input variables are arranged in the same column. The source nodes D_1, D_2 involve the input variable x_1, while D_3, D_4 contain x_2 and D_5, D_6 consist of x_3, respectively.

It is noted that according to (16), there are supposed to be 23 intermediate nodes in the network. However, only 20 intermediate nodes are generated in Fig. 3. One reason is that the neurons containing the same variables, e.g. D_1 and D_2, are not allowed to form a new neuron according to Rule 2. Another reason is that some combinations may be duplicate. For example, $\min\{\text{nn}_{D_1}, \text{nn}_{C_9}\}$, $\min\{\text{nn}_{D_3}, \text{nn}_{C_3}\}$ are equivalent (all equal to $\min\{\text{nn}_{D_1}, \text{nn}_{D_3}, \text{nn}_{D_5}\}$), and we can only keep 1 combination to formulate a new neuron C_{13}.

It has been proved in [14] that the boundary configuration defined by (15) subdivides the hypercube $[0,1]^n$ into the simplices with a scaling factor $1/q$. Letting the equation (15) equal $f(x)$ at the vertex of the simplicies yields unique

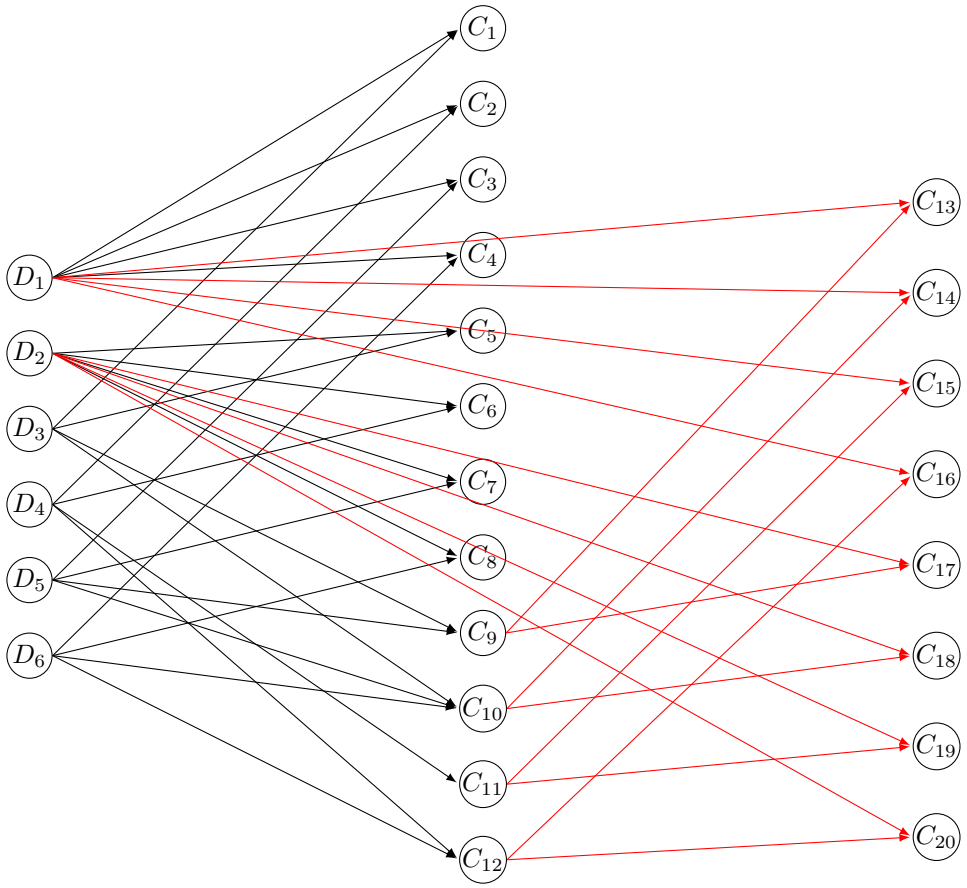


Fig. 3. Illustration of the full connection of neurons in the hidden layer of the EHH network with $n = 3$ and $q = 2$. The neurons with the same number of input variables are arranged in the same column, and D and C stand for the source and intermediate nodes, respectively.

coefficients $a_{j_1 \ldots j_r}^{(k_1 \ldots k_r)} (r = 1, \ldots, n)$. Moreover, the following holds,

$$\|f_{\text{EHH}}(\mathbf{x}) - f(\mathbf{x})\| \leq \epsilon, \forall \mathbf{x} \in [0,1]^n$$

where $\epsilon = \max\limits_{\Delta \in [0,1]^n} \max\limits_{\mathbf{x}_1, \mathbf{x}_2 \in \Delta} \|f(\mathbf{x}_1) - f(\mathbf{x}_2)\|$, and Δ denotes the simplices.

As for any given ε, due to the continuity of f on the compact set $[0,1]^n$, we can find δ such that

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| < \varepsilon, \quad \forall \|\mathbf{x}_1 - \mathbf{x}_2\| < \delta.$$

Choosing q to ensure

$$\|\mathbf{x}_1 - \mathbf{x}_2\| < \delta, \forall \mathbf{x}_1, \mathbf{x}_2 \in \Delta, \forall \Delta \in [0,1]^n,$$

then we have

$$\epsilon = \max_{\Delta \in [0,1]^n} \max_{\mathbf{x}_1, \mathbf{x}_2 \in \Delta} \|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| < \varepsilon.$$

Therefore the conclusion (14) follows.

As is shown in the proof of Theorem 3, the EHH neural network can approximate any continuous function in a compact domain with arbitrary precision, if sufficiently large number of neurons are provided. In other words, it requires that the simplices Δ are small enough. Actually, in the simulation study, we will show that the number of the neurons will not exceed a few hundreds in approximating a complex function in a few tens dimension, and the training of the EHH neural network will remove some of the preset neurons.

3.4 Interpretability of EHH

3.4.1 Interaction matrix

To describe the interpretability of the EHH network, we introduce the **interaction matrix Ir** to depict the interactions among different input variables.

Definition* 2** *Given an EHH neural network that has M neurons in the hidden layer, the matrix $\mathbf{Ir} = [\mathbf{Ir}]_{ij} \in \mathbb{R}^{M\times M}$ is called the* ***interaction matrix *of the EHH neural network, if it satisfies*

- $\mathbf{Ir}_{ij} = 1$ *indicates that there is a path (maybe not direct) from the node A_i to the node A_j, and $\mathbf{Ir} = 0$ indicates otherwise;*
- $\mathbf{Ir}_{ij} = 0$, $\forall j$, *if A_i is an intermediate node.*

As (12) indicates, for each neuron A_j, we can find the univariate neurons that constitute it, i.e., find the source nodes that have connections to the neuron A_j (may be not direct connection). Although the interaction matrix can not be obviously obtained from the structure of the EHH neural network, it can be obtained from the adjacency matrix, as Algorithm 1 shows.

Algorithm 1 *Obtaining the interaction matrix for an EHH neural network.*

Input: *The EHH neural network f_{EHH} with the adjacency matrix $\mathbf{A_d}$.*
Output: *The interaction matrix $\mathbf{Ir}$ for the EHH neural network.*
Initialize $\mathbf{Ir} = \mathbf{A_d}$;
for $j = 1 : M$ ***do***
  Let $\mathcal{I} = \{j\}$.
  for $i \in \mathcal{I}$ ***do***
    if $i > N_d$ ***then***
      Find i_1, i_2 *such that* $\mathbf{Ir}_{i_1,i} = 1, \mathbf{Ir}_{i_2,i} = 1$, *add* i_1, i_2 *to the set* $\mathcal{I}$.
      Let $\mathbf{Ir}_{i,j} = 0$.
    else
      Let $\mathbf{Ir}_{i,j} = 1$.
    end if
  end for
end for

In a word, the adjacency matrix $\mathbf{A_d}$ represents the connections of each neuron while the interaction matrix $\mathbf{Ir}$ describes the interactions among different input variables. For example, the interaction matrix for Fig. 1(b) are listed.

$$\mathbf{Ir} = \left[\begin{array}{c|cccc:cc:cc} & D_1 & D_2 & D_3 & D_4 & C_1 & C_2 & C_3 & C_4 \\ \hline D_1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ D_2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ D_3 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ D_4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hdashline C_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline C_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]. \tag{17}$$

***Remark* 4** *Both the adjacency matrix and interaction matrix are sparse, which is easy to be stored and applied to large-scale problems.*

3.4.2 ANOVA decomposition

Similar to the ANOVA decomposition used in [7], here we could easily get the ANOVA decomposition of the EHH neural network through the interaction matrix $\mathbf{Ir}$,

$$f_{\mathrm{EHH}}(\mathbf{x}) = \alpha_0 + \sum_i \sum_{|\mathcal{S}_{A_j}|=1} \alpha_j \mathrm{nn}_{A_j}(x_i) + \sum_{i,k} \sum_{|\mathcal{S}_{A_j}|=2} \alpha_j \mathrm{nn}_{A_j}(x_i, x_k) + \sum_{i,k,r} \sum_{|\mathcal{S}_{A_j}|=3} \alpha_j \mathrm{nn}_{A_j}(x_i, x_k, x_r), \tag{18}$$

in which $|\mathcal{S}_{A_j}|$ denotes the number of elements in the set $\mathcal{S}_{A_j}$, i.e., the number of nonzero elements in $\mathbf{Ir}(:, j)$. The first sum is over all source nodes, the second sum is over all neurons having exactly 2 input variables, representing (if present) two-variable interactions. Similarly, the third sum represents (if present) the contributions from three-variable interactions in corresponding neurons and so on. As (18) is similar to the decompositions provided by the analysis of variance for contingency tables, analog to the definition, we refer to (18) as the ANOVA decomposition of the EHH neural network.

It is easy to derive (18) for an EHH neural network. For example, each function in the first sum of (18) can be expressed as

$$f_i(x_i) = \sum_{\mathcal{S}_{A_j}=\{i\}} \alpha_j \mathrm{nn}_{A_j}(\mathbf{x}), \tag{19}$$

which is a sum over all source nodes involving the particular variable x_i.

Each bivariate function in the second sum of (18) can be expressed as

$$f_{i,k}(x_i, x_k) = \sum_{\mathcal{S}_{A_j}=\{i,k\}} \alpha_j \mathrm{nn}_{A_j}(\mathbf{x}), \tag{20}$$

which is a sum over all neurons involving the particular pair of the variables x_i and x_k. Adding this to the corresponding univariate contributions (19) (if present) indicates the joint bivariate contribution of x_i and x_k to the model. Similarly, each trivariate function in the third sum can be obtained by collecting together all neurons involving the particular variable triples, i.e.,

$$f_{i,k,r}(x_i, x_k, x_r) = \sum_{\mathcal{S}_{A_j}=\{i,k,r\}} \alpha_j \mathrm{nn}_{A_j}(\mathbf{x}). \tag{21}$$

Adding this to the corresponding univariate and bivariate functions (19), (20) involving x_i, x_k and x_r, provides the joint contribution of these three variables to the model. Terms involving more variables (if present) can be collected together and represented similarly.

The summation (19)-(21) can be easily determined through the interaction matrix. For example, in order to determine the index j such that $\mathcal{S}_{A_j} = \{i, k, r\}$, assume the variables x_i, x_k and x_r are contained in the source nodes A_{γ_i}, A_{γ_k} and A_{γ_r}, respectively. That means $\mathcal{S}_{A_{\gamma_i}} = \{i\}$, $\mathcal{S}_{A_{\gamma_k}} = \{k\}$, and $\mathcal{S}_{A_{\gamma_r}} = \{r\}$. It is noted that such γ_i, γ_k and γ_r may not be unique. Then if for any $\gamma_i, \gamma_k, \gamma_r$, the following is true,

$$\mathbf{Ir}_{\gamma_i,j} = 1, \mathbf{Ir}_{\gamma_k,j} = 1, \mathbf{Ir}_{\gamma_r,j} = 1,$$

we have $\mathcal{S}_{A_j} = \{i, k, r\}$.

Interpretation of the EHH neural network is greatly facilitated through its ANOVA decomposition (18). From (18), we can know the importance of the input variables as well as the interactions among different input variables appeared in the EHH neural network .

4 Training of the EHH neural network

For neural networks, suitable initial parameters of the model, like the number of layers and the number of neurons, are essential. To search for a good neural network structure, conventional approaches apply evolution algorithm or reinforcement learning over a discrete and non-differentiable search space, which is computationally demanding [30]. One kind of speeding up approaches includes imposing a particular structure of the search space [16].

Now that the structure of the EHH neural network can be described through the adjacency matrix like (9), the determination of the EHH neural network can be formulated as determining the binary parameters in the adjacency matrix as well as continuous variables $\alpha_j, j \in \{0, 1, \ldots, M\}$ in (10).

As is well known, to simultaneously obtain the optimal binary and continuous variables, we have to solve a mixed-integer nonlinear programming problem, which is NP-hard. For the EHH neural networks with large number of neurons, this is prohibitive. Hence in our work, we propose a descent algorithm, which search the solution in a subspace, but yet result in a local optimal solution. Besides, the worst-case complexity analysis as well as a speeding up strategy of the algorithm is given.

4.1 Initial EHH neural network generation

Given the pre-defined number of neurons M, the initial EHH network with M neurons is generated randomly according to Rule 1 and 2. Apart from the source nodes, as each neuron (node) in the hidden layer is generated by taking the minimum of 2 existing neurons, we have

$$\mathbf{A_d}(i, j) = 0, \ \forall i \geq j, j \in \{1, \ldots, M\}. \tag{22}$$

In order to determine the weights $\alpha_j, j = 0, 1, \ldots, M$ in (10), an ℓ-1 regularization problem, namely Lasso optimization problem is solved. The reason of imposing an ℓ-1 penalty is to remove neurons that are either dependent on each other or contribute less to the output. Specifically, Lasso pruning is done through the optimization problem described below:

$$\min_{\boldsymbol{\alpha}} \ J(\alpha) = (y - \mathbf{Z}\alpha)^T (y - \mathbf{Z}\alpha) + \lambda \|\boldsymbol{\alpha}\|_1, \tag{23}$$

where $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \ldots, \alpha_M]^T$, $\mathbf{Z}$ is called the data matrix

$$\mathbf{Z} = \begin{bmatrix} 1 & \mathrm{nn}_1(\mathbf{x}(1)) & \cdots & \mathrm{nn}_M(\mathbf{x}(1)) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \mathrm{nn}_1(\mathbf{x}(N_s)) & \cdots & \mathrm{nn}_M(\mathbf{x}(N_s)) \end{bmatrix}. \tag{24}$$

The first all one column in $\mathbf{Z}$ represents the constant bias, $\{\mathbf{x}(k), y(k)\}_{k=1}^{N_s}$ are the training samples, and $\mathrm{nn}_j(\mathbf{x}(k))$ is the output of the neuron A_j for the k-th sample.

***Remark* 5** *The shrinkage parameter λ controls the amount of sparsity, which should be selected according to specific problems. A common choice of λ is done through cross validation. Here we assume*

$$\lambda = \zeta \sqrt{2 \log(l_\alpha)}, \tag{25}$$

where l_α denotes the length of $\boldsymbol{\alpha}$.

The parameter ζ is chosen from a set of values (different for specific problems) as the one yielding the least generalized cross-validation criterion GCV [5],

$$\mathrm{GCV}(M) = \frac{\sum_{i=1}^{N_s} [y(i) - f_{\mathrm{EHH}}(\mathbf{x}(i))]^2}{N_s \left[1 - \frac{C(M)}{N_s}\right]^2}, \tag{26}$$

in which the complexity function takes the form of

$$C(M) = \mathrm{trace}(\mathbf{Z}(\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T) + d \cdot M \tag{27}$$

according to [7]. The quantity d represents a cost for each neuron optimization and larger value for d will lead to fewer neurons in the network. The principle of the selection of d can be found in [7], and accordingly, here we choose $d = 3$. The term $\mathrm{trace}(\mathbf{Z}(\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T)$ is actually the column rank of the matrix $\mathbf{Z}$. When the columns of $\mathbf{Z}$ are independent, i.e., the outputs of all neurons are independent, we have $C(M) = M + 1$.

By solving the problem (23), we have introduced sparsity in the entries of $\boldsymbol{\alpha}$, i.e., the connections of some neurons to the output become 0, meaning that such neurons contribute much less significantly to the output. Introducing an augmented adjacency matrix $\bar{\mathbf{A}}_\mathbf{d}$,

$$\bar{\mathbf{A}}_\mathbf{d} = \begin{bmatrix} & \alpha_1 \\ \mathbf{A_d} & \vdots \\ & \alpha_M \end{bmatrix}, \tag{28}$$

then there may be rows with all zero elements in $\bar{\mathbf{A}}_\mathbf{d}$, i.e., there may be neurons that are neither connected to other neurons nor to the output. We can delete these neurons in the network without affecting the performance of the EHH neural network. This can be fulfilled through deleting corresponding rows and columns in the augmented adjacency matrix $\bar{\mathbf{A}}_\mathbf{d}$, and such deletion can be repeated until there are no all zero rows. Take Fig. 1(b) for example, the augmented adjacency matrix is,

$$\bar{\mathbf{A}}_\mathbf{d} = \left[\begin{array}{c|cccccccc|c} & D_1 & D_2 & D_3 & D_4 & C_1 & C_2 & C_3 & C_4 & y \\ \hline D_1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \alpha_1 \\ D_2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \alpha_2 \\ D_3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \alpha_3 \\ D_4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \alpha_4 \\ C_1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \alpha_5 \\ C_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \alpha_6 \\ C_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_7 \\ \hline C_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_8 \end{array}\right]. \tag{29}$$

If α_8 becomes 0 after solving the optimization problem (23), which means C_4 is not connected to the output, then both the 8-th row and 8-th column can be deleted in $\bar{\mathbf{A}}_\mathbf{d}$. Then if further $\alpha_6 = 0$, as C_4 has been removed, the neuron C_2 is not connected to any other neurons nor the output, both the 6-th row and 6-th column can be also deleted. Notice that if $\alpha_6 = 0, \alpha_8 \neq 0$, no neurons can be deleted, as there are no all zero rows in this case, which means that although C_2 is not connected to the output, it has connection to C_4. After deleting all redundant neurons, the neural network is much more condensed and the new adjacency matrix $\mathbf{A_d}$ can be obtained by just removing the last column in the resulting $\bar{\mathbf{A}}_\mathbf{d}$.

4.2 Optimization of the EHH neural network

The optimization of the EHH neural network refers to the simultaneously determining the continuous and binary variables, and can be cast as the problem below:

$$\min_{\mathbf{A_d}, \boldsymbol{\alpha}} \; J(\mathbf{A_d}, \boldsymbol{\alpha}) = (y - \mathbf{Z}\boldsymbol{\alpha})^T (y - \mathbf{Z}\boldsymbol{\alpha}) + \lambda \|\boldsymbol{\alpha}\|_1, \tag{30}$$

in which the data matrix $\mathbf{Z}$ is uniquely defined by the adjacency matrix $\mathbf{A_d}$ and the sampled data $(\mathbf{x}(k), y(k))_{k=1}^{N_s}$.

As it's extremely time-consuming to solve a mixed-integer nonlinear programming problem, here in this paper, the variables $\mathbf{A_d}$ and $\boldsymbol{\alpha}$ are optimized by 2 subproblems in turn, i.e., the structure optimization concerning $\mathbf{A_d}$ with $\boldsymbol{\alpha}$ fixed, and the weight optimization problem concerning $\boldsymbol{\alpha}$ with $\mathbf{A_d}$ fixed.

4.2.1 Structure optimization

For the structure optimization, as the source nodes are provided according to (5), hence the first N_d columns of $\mathbf{A_d}$ are fixed to be zero.

For the next $(M - N_d)$ columns, considering the column $j(N_d < j \leq M)$, according to (22), only $j-1$ elements can be nonzero. Hence the total number of binary variables to be optimized is

$$N_d + (N_d + 1) + \ldots + M - 1 = \frac{(N_d + M - 1)(M - N_d)}{2},$$

which is quadratic in M. When M is large, it is prohibitive to obtain an optimal structure through the binary programming problem. Hence, a natural thought is to optimize one part of the binary variables at a time and fix the other parts. Then different parts of the binary variables are optimized in turn.

Specifically, for the column j, randomly letting 2 of the $j-1$ elements to be 1, then there will be $j(j-1)/2$ combinations. Traversing all the combinations and choose the one that yields the least cost in (30). It is noted that when optimizing the column j, the other columns in $\mathbf{A_d}$ are fixed.

4.2.2 Weights optimization

For the optimization of the weights $\boldsymbol{\alpha}$, the Lasso optimization problem (23) is solved with respect to the newly obtained adjacency matrix $\mathbf{A_d}$ in the structure optimization. Notice that after weights optimization, $\mathbf{A_d}$ may have changed due to the sparsity of $\boldsymbol{\alpha}$.

Iteratively, the optimization of the EHH neural network starts again to optimize the columns $N_d + 1, \ldots, M$ in $\mathbf{A_d}$ and then finding the optimal weights $\boldsymbol{\alpha}$ given a fixed $\mathbf{A_d}$. The optimization cycles can be repeated until the cost function in (30) stops deceasing. Algorithm 2 lists the overall optimization procedure. In this paper, we refer to **1-cycle optimization** of the EHH neural network as a sequent structure optimization and weights optimization, corresponding to line 4 to 11.

Algorithm 2 *Training the EHH neural network.*

Input: *Training data* $(\mathbf{x}(k), y(k))_{k=1}^{N_s}$*, the preset number of neurons.*
Output: *The EHH neural network.*
1: *Get the q-quantiles in each dimension, say* $\beta_{11}, \ldots, \beta_{1,q-1}, \ldots, \beta_{n1}, \ldots, \beta_{n,q-1}$*;*
2: *The neurons in the first hidden layer are listed as (5).*
3: ***repeat***
4: ***for*** $j = nq + 1 : M$ ***do***
5: ***repeat***
6: *Choose 2 elements of* $\mathbf{A_d}(i, j)$ *with* $i < j$ *to be 1.*
7: *Fixing* $\boldsymbol{\alpha}$ *and calculate the cost in (30).*
8: ***until*** *All the* $\frac{j(j-1)}{2}$ *combinations are traversed.*
9: *Choose the combination that yields the least cost in (30).*
10: ***end for***
11: *Solve the lasso optimization problem (23) to get the optimal weights* $\boldsymbol{\alpha}$*.*
12: ***until*** *The cost in (30) stops decreasing.*

4.3 Complexity analysis

Before the complexity analysis, we show that Algorithm 2 is a non-increasing algorithm and can reach a local optimum in neighbours. The definition of neighbours for this particular problem is defined as follows.

Definition 4 *For the optimization problem (30), given the adjacency matrix* $\mathbf{A_{d0}}$ *and weights* $\boldsymbol{\alpha}_0$*, the neighbour is defined as the set* $\mathcal{N}(\mathbf{A_{d0}}, \boldsymbol{\alpha}_0)$ *such that* $\forall(\mathbf{A_d}, \boldsymbol{\alpha}) \in \mathcal{N}(\mathbf{A_{d0}}, \boldsymbol{\alpha}_0)$*, the following holds*

1) $\mathbf{A_d}$ *and* $\mathbf{A_{d0}}$ *only differ in 1 column.*
2) $\|\boldsymbol{\alpha} - \boldsymbol{\alpha}_0\| \leq \delta$*, in which* δ *is a scalar small enough.*

Lemma 5 *Algorithm 2 is a non-increasing algorithm, i.e.,*

$$J(\mathbf{A_{d1}}, \boldsymbol{\alpha}_1) \leq J(\mathbf{A_{d0}}, \boldsymbol{\alpha}_0),$$

where $\mathbf{A_{d}}_0, \boldsymbol{\alpha}_0$ *and* $\mathbf{A_{d}}_1, \boldsymbol{\alpha}_1$ *are the adjacency matrices as well as the weights before and after 1-cycle optimization.*

PROOF.

For the optimization of the structure, when optimizing the column $\mathbf{A_d}(:, j)(j \in \{N_d+1, \ldots, M\})$, the other columns $\mathbf{A_d}(:, k), k \in \{1, \ldots, M\}, k \neq j$ remain unchanged. Considering the cost

$$J(\mathbf{A_d}) = (y - \mathbf{Z}(\mathbf{A_d}))^T (y - \mathbf{Z}(\mathbf{A_d})) + \lambda \|\boldsymbol{\alpha}_0\|_1, \tag{31}$$

in which $\boldsymbol{\alpha}_0$ is fixed. The column $\mathbf{A_{d}}_1(:, j)$ is chosen as the one yielding the least cost (31). During this process, only 1 column is optimized at a time. Therefore, after optimizations for the column $\mathbf{A_d}(:, N_d+1), \ldots, \mathbf{A_d}(:, M)$, the cost (31) keeps non-increasing and

$$J(\mathbf{A_{d}}_1, \boldsymbol{\alpha}_0) \leq J(\mathbf{A_d}, \boldsymbol{\alpha}_0), \forall (\mathbf{A_d}, \boldsymbol{\alpha}_0) \in \mathcal{N}(\mathbf{A_{d}}_0, \boldsymbol{\alpha}_0).$$

As $\boldsymbol{\alpha}_0$ stays constant in the optimization of the structure, it may not be optimal for the optimization problem (23). Hence, after the structure has been obtained, solving the problem (23) would further reduce the cost $J(\boldsymbol{\alpha})$ in (23), i.e.,

$$J(\mathbf{A_{d}}_1, \boldsymbol{\alpha}_1) \leq J(\mathbf{A_{d}}_1, \boldsymbol{\alpha}_0)$$

Therefore, repeated optimization of the adjacency matrix $\mathbf{A_d}$ and the weights $\boldsymbol{\alpha}$ will result in a non-increasing cost $J(\mathbf{A_d}, \boldsymbol{\alpha})$ and the algorithm stops at a point with minimum cost in neighbours.

The worst-case complexity of 1-cycle optimization, i.e., line 4-11 in Algorithm 2 is given in the following lemma.

Lemma 6 *The worst-case complexity for 1-cycle optimization in Algorithm 2 is* $O(M^4 N_s)$.

PROOF. In the structure optimization, for the optimization of the column j with $j \in \{N_d + 1, \ldots, M\}$, we have to traverse

$$\frac{j(j-1)}{2}$$

combinations in order to find the optimal value of the column during this cycle, corresponding to line 6-7.

For each combination, in order to obtain the data matrix $\mathbf{Z}$, we have to conduct $2 \cdot N_s$ comparisons for the neuron j, then for neurons such that $\mathbf{A_d}_{jk} = 1$ with $j < k \leq M$, the corresponding values in the data matrix $\mathbf{Z}$ also have to be changed. Thus we have at most $(M - j) \cdot N_s$ comparisons. After obtaining the data matrix $\mathbf{Z}$, $N_s \cdot (M + 1)$ multiplications and $N_s \cdot (M + 1)$ additions have to be done to get $\mathbf{Z} \cdot \boldsymbol{\alpha}$. Then N_s multiplications and N_s additions are required to get the quantity $(\mathbf{y} - \mathbf{Z}\boldsymbol{\alpha})^T(\mathbf{y} - \mathbf{Z}\boldsymbol{\alpha})$. Hence in total, for the column j, at most

$$\frac{j(j-1)}{2} \cdot (3M + 6) \cdot N_s$$

mathematical operations are required. Hence the worst-case complexity for the optimization of each column is $O(M^3 N_s)$. Considering the optimization for all columns, the worst-case complexity for line 4-11 is

$$O(M^4 N_s).$$

After the structure optimization, the Lasso optimization problem is solved using the ADMM algorithm. The worst-case complexity of ADMM algorithm is $O(1/\epsilon^2)$, where ϵ is the primal and dual residuals in the ADMM algorithm. Usually a few tens of iterations is enough to obtain a moderate accuracy, which is sufficient for many large-scale applications [2]. Hence in 1-cycle optimization of Algorithm 2, the complexity of solving a Lasso optimization problem is negligible compared with $O(M^4 N_s)$.

Therefore the overall worst-case complexity for each cycle of optimization is $O(M^4 N_s)$.

It is clear that the worst-case complexity is high for large M. To reduce complexity, we can redefine a smaller neighbour and design corresponding algorithms. For example, instead of optimizing a column at a time, we can only optimize an element in the column at a time, i.e., pick up $\mathbf{A_d}_{k_1,j}$ and $\mathbf{A_d}_{k_2,j}$ such that

$$\mathbf{A_d}_{k_1,j} = 1, \mathbf{A_d}_{k_2,j} = 1.$$

And we optimize k_1 while fixing k_2 and the other columns in $\mathbf{A_d}$ as well as the weights $\boldsymbol{\alpha}$. After k_1 has been optimized, we optimize k_2 while fixing the newly generated k_1 and the other columns in $\mathbf{A_d}$ as well as $\boldsymbol{\alpha}$. In this case, the mathematical operations for each column j will be reduced to

$$2(j-1) \cdot (3M+6) \cdot N_s.$$

Hence the worst-case complexity for each cycle of optimization is $O(M^3 N_s)$.

In this situation, the neighbour is smaller by adding the following condition to the conditions 1) and 2),

3) $\mathbf{A_d}$ and $\mathbf{A_d}_0$ only differs in 1 element.

The number of cycles of optimizations is problem-dependent, hence the worst-case complexity for Algorithm 2 can not be estimated. In the simulations, generally the number of cycles is around 10, and we will report this in detail in the simulations. For problems with hundreds of dimensions, usually there should be thousands of neurons in order to get a high accuracy, and in this case even the reduced complexity $O(M^3 N_s)$ is also high. Currently we are mainly dealing with problems that only need hundreds of neurons, and extremely large-scale problems are left for our future work.

5 Nonlinear dynamic system identification using the EHH neural network

Given a nonlinear dynamic system

$$y(k) = f(\varphi(k), \theta) + \varepsilon(k), \tag{32}$$

where $y(k)$ denotes the output at time k, f is an unknown nonlinear relationship, $\varphi(k)$ is the regression vector and $\varepsilon(k)$ is the error term. The identification goal is to find a nonlinear mapping such that the additive error term $\varepsilon(k)$ is small.

The nonlinear dynamic system identification problem can be described as,

$$\hat{y}(k) = \hat{f}(\varphi(k)), \tag{33}$$

where $\hat{y}(k)$ denotes the predicted output at time k, $\hat{f}$ is the approximated nonlinear relationship to be determined. Here we consider the black-box NARX model [22,24], i.e., the regression vector $\varphi(k)$ consists of the past outputs $y(k-1), \ldots, y(k-n_b)$ and the current as well as past inputs $u(k), u(k-1), \ldots, u(k-n_a)$,

$$\varphi(k) = [y(k-1), \ldots, y(k-n_b), u(k), u(k-1), \ldots, u(k-n_a)]^T. \tag{34}$$

Given the input $u(k)$ and the observed output $y(k)$ for $t = 1, 2, \ldots, N$, our objective here is to find an optimal EHH neural network as the approximated nonlinear relationship $\hat{f}$. To test the model on the test set, the simulation error is considered [1], in which only the input is used to generate the simulated output, that is

$$y_s(k) = \hat{f}(\varphi_s(k)), \tag{35}$$

in which

$$\varphi_s(k) = [y_s(k-1), \ldots, y_s(k-n_b), u(k), u(k-1), \ldots, u(k-n_a)]^T.$$

The subsequent benchmark examples will show the effectiveness of the EHH neural networks for the modeling of nonlinear systems, besides, the importance of the regressors as well as the interactions among the regressors will be revealed.

5.1 Narendra-Li benchmark

This benchmark was firstly introduced by [17], and has become a benchmark for nonlinear system identification, e.g., [26,29]. The benchmark is defined by the state-space form

$$\begin{aligned} x_1(k+1) &= \left(\frac{x_1(k)}{1+x_1^2(k)}+1\right)\sin x_2(k), \\ x_2(k+1) &= x_2(k)\cos x_2(k) + x_1(k)e^{-\frac{x_1^2(k)+x_2^2(k)}{8}} + \frac{u^3(k)}{1+u^2(k)+0.5\cos(x_1(k)+x_2(k))}, \\ y(k) &= \frac{x_1(k)}{1+0.5\sin x_2(k)} + \frac{x_2(k)}{1+0.5\sin x_1(k)} + e(k). \end{aligned} \tag{36}$$

According to [17], this system does not correspond to any real system. It is chosen to be sufficiently complex and nonlinear so that the conventional linear method will not give satisfactory performance. The Gaussian white noise $e(k)$ is generated with a variance of 0.1. The states are assumed not to be measurable. The training signal is obtained with an input sequence sampled uniformly from $[-2, 2]$, with a data length 2000. The test data is generated with the input

$$u(k) = \sin(2\pi k/10) + \sin(2\pi k/25), k = 1, \ldots, 200.$$

5.1.1 EHH approximation for the Narendra-Li benchmark

Following the discussion in [26,29], the regression vector is chosen to be

$$\varphi(k) = [y(k-1), y(k-2), y(k-3), u(k-1), u(k-2), u(k-3)]^T.$$

The same as [26], the performance of the approximation is assessed by the test VAF (variance- accounted-for), i.e.,

$$\mathrm{VAF}(\hat{f}) = \max\left\{0, 1 - \frac{\mathrm{var}(\hat{f}-y)}{\mathrm{var}(y)}\right\},$$

in which $\hat{f}$ denotes the approximations of the true system using different kinds of methods and var represents the variance.

To determine the number of source nodes N_d and the number of total neurons M, we test 9 groups of values, i.e.,

$$N_d = 6\times 5, 6\times 10, 6\times 20,$$

and

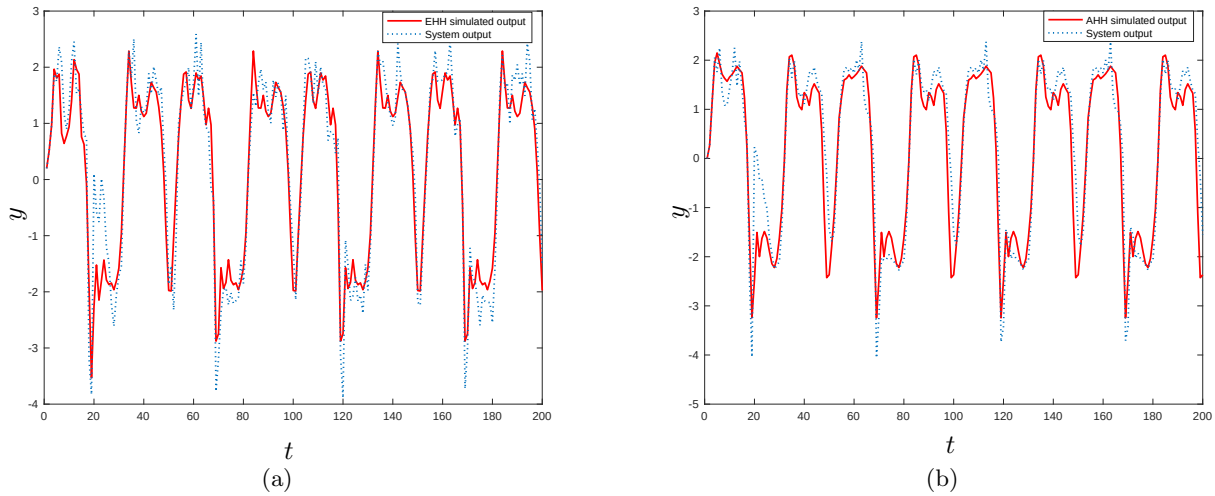
$$M = N_d + 30, N_d + 40, N_d + 50.$$

The number of parameters is not large as the dimension is not high. After training 9 different EHH neural networks, we select the one with the least GCV criterion. The corresponding structure is set to be the network structure in subsequent experiments. For this problem, the number of source nodes and intermediate nodes are preset to be 6×5 and 40, respectively.

10 experiments are then performed and the one resulting the best test RSSE is selected to compare with the other methods. The estimation procedure includes 5 cycles of evaluating line 4 to line 11, and the elapsed time is 2.83s. The number of neurons employed finally is 59, which is smaller compared with the preset 70. All the computations in this paper are implemented through MATLAB 2016b on a 2.7 GHz Intel Core i7 computer.

The simulated output of the EHH network is shown in red solid in Fig. 4(a), in which the system output is also shown (in blue dotted). The simulated output of the AHH model is taken from [29] and shown in red solid in Fig. 4(b) for comparison, in which again the system output is shown in blue dotted.

The test VAF (simulated VAF), the calculation complexity and the number of parameters (denoted as # Parameters) of the EHH neural network and the corresponding initial neural network are shown in Table 1. The results for the NARX model with AHH model [29], basis piecewise affine (BPWA) [26] are also listed for comparison. The



(a) (b)

Fig. 4. Test performance of the EHH neural network and the AHH tree for the Narendra-Li benchmark. (a) The EHH simulated output and the system output. (b) The AHH simulated output and the system output.

Table 1
Comparison of test performance of AHH model, the BPWA strategy and EHH neural network for the Narendra-Li system.

	AHH [29]	BPWA [26]	EHH_{ini}	EHH
RSSE	87.4%	89.1%	0	90.5%
Elapsed Time	49.8	-	0.64s	2.83s
# Parameters	132	847	187	178

computational complexity for the AHH model is re-evaluated through the current computer while that for the BPWA strategy is not available, and shown as "-".

It can be seen from Table 1 that the EHH neural network gives a good approximation of the original dynamic system (36) and outperforms the strategy of the AHH model and BPWA strategy. Compared with the AHH model, the elapsed time for the EHH neural network is much shorter while the accuracy is better. It is noted that the number of parameters employed for the EHH neural network is a little larger than that for the AHH tree, indicating the effectiveness of the AHH training procedure, although the training lasts for a longer time.

It is also noted that compared with the initial EHH neural network, the final EHH neural network gives a much better accuracy and at the same time, the number of parameters is reduced, verifying the effectiveness of the structure and weights optimization.

5.1.2 ANOVA decomposition of the EHH approximation

The ANOVA functions with the top 5 σ-values of the resulting EHH neural network are listed in Table 2. Larger σ-value indicates that the ANOVA function is more important. The term \GCV for an ANOVA function represents the GCV score for a model with all of the nodes corresponding to that particular ANOVA function removed. Hence it can provide another indication of the importance of the corresponding ANOVA function, i.e., larger \GCV indicates the more importance of the ANOVA function.

From the table we can know that the trend of the σ-value and \GCV are approximately the same. The most important contribution comes from the interaction among $y(k-1), y(k-2), u(k-2)$, and the additive terms $y(k-1), y(k-2), u(k-1)$ also contribute largely.

It is noted that for the NARX based system identification, the ANOVA decomposition only gives a rough indication of the importance of the regressors as well as the regressor interactions, as different regressors are actually dependent on each other.

Table 2
The ANOVA functions with the top 5 σ-values of the EHH neural network for the Narendra-Li system.

ANOVA Fun.	Regressor	σ	\GCV
1	$y(k-1), y(k-2), u(k-2)$	1.7080	3.8234
2	$u(k-1)$	1.2001	3.6748
3	$y(k-1), y(k-2)$	0.8815	1.0973
4	$y(k-2)$	0.7668	2.9347
5	$y(k-1)$	0.5451	0.6632

5.2 Bouc-Wen system

The Bouc-Wen model has been intensively used to model hysteretic effects in mechanical engineering, especially in the case of random vibrations. An extensive literature review about the Bouc-Wen modeling can be found in [13]. The Bouc-Wen benchmark system has been described in detail in [19,27]. The training signal consists of 5 periods of a multisine and has a total length of 40960 samples. The first 2 periods of this signal were removed for the transient [6]. There are 2 sets of test data, one is with multisine input and the other is with swept-sine input, with lengths 8192 and 153000, respectively.

5.2.1 EHH approximation of Bouc-Wen system

The same as [27], the regressor is chosen to be

$$\varphi(k) = [y(k-1), \ldots, y(k-15), u(k), \ldots, u(k-14)]^T.$$

And according to [19], the performance of the approximation is judged by the test RMSE (root mean square error), i.e.,

$$\mathrm{RMSE}(\hat{f}) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(\hat{f}(k) - y(k))^2},$$

in which $\hat{f}$ denotes the approximations of the true system using different kinds of methods.

To determine the number of source nodes N_d and the number of total neurons M, as the dimension for this problem is much higher than the Narendra-Li benchmark, we test the following 9 groups of values, i.e.,

$$N_d = 30 \times 5, 30 \times 10, 30 \times 20,$$

and

$$M = N_d + 100, N_d + 200, N_d + 250.$$

After training 9 different EHH neural networks, we select the one with the least GCV criterion. The corresponding structure is set to be the network structure in subsequent experiments. For this problem, the number of the source nodes and intermediate nodes are preset to be 30×5 and 200, respectively.

Then 10 experiments are done, and the one with the best test RMSE is shown in Table 3, in which the result of the initial EHH neural network (without the structure and weights optimization) is also listed. The time complexity for getting an initial EHH neural network is approximately 8.6s, then 305s is used to get the final EHH neural network. The whole procedure includes 5 cycles of structure and weights optimization, and the final EHH neural network includes 145 neurons. The error is listed in dB form, i.e., $20\log 10(\mathrm{RMSE})$. The results for the NARX model with tree-based local model networks [1], Volterra Feedback Model [20], and decoupled polynomial NARX model [27] are also listed for comparison. It is noted that the application of the AHH model in the Bouc-Wen benchmark problem is prohibitive due to the high dimension (30 dimensions).

It can be seen from Table 3 that the EHH neural network performs well, and outperforms the procedure in [1] and [20]. We can also see that the performance of the optimized EHH neural network is better than the initial EHH neural network while at the same time, the number of parameters is reduced, verifying the effectiveness of the structure and

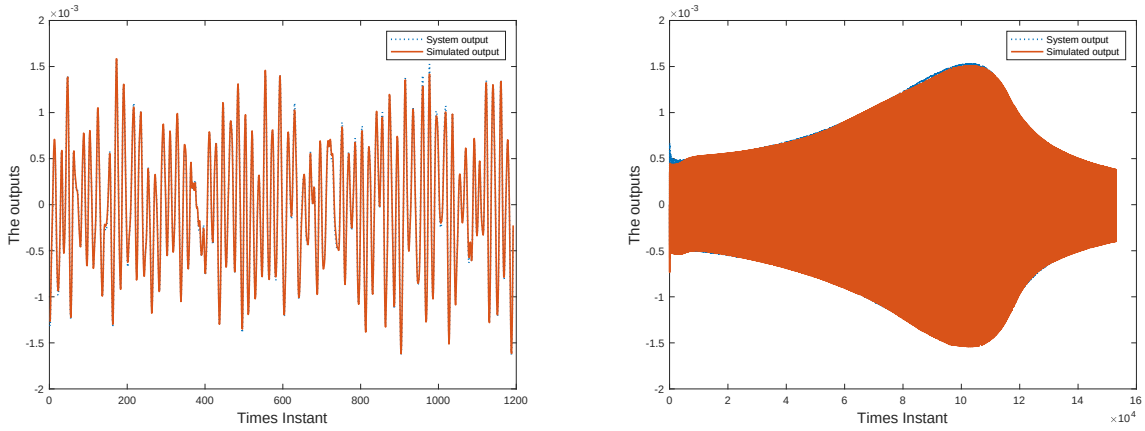
Table 3
Comparison of test performance of several approaches on Bouc-Wen system.

	NARX [1]	Volterra Feedback [20]	Decoupled NARX [27]	EHH_{ini}	EHH
RMSE (multisine)	-75.73	-81.51	-85.42	-84.56	**-86.11**
RMSE (swept-sine)	-77.20	-85.03	**-95.55**	-90.09	-92.39
# Parameters	-	-	226	610	436

weights optimization. Compared with the decoupled NARX [27], the RMSE for the multisine test data is smaller while that for the sinesweep data is a little larger.

Besides, the number of parameters employed in the EHH neural network is 436, while in [27], the number of parameters is 226. To further improve the accuracy and reduce the number of parameters of the EHH neural network, we can also employ a linear transformation before the input layer like that in [27], which somewhat reflects the physical insights of the dynamic system. And this part of work has been finished as our another work [15]. Or we can build a state-space version for the EHH neural network, like [23], which we will consider in the future.

Fig. 5 shows the simulated output (shown in red solid) as well as the system output (shown in blue dotted), in which Fig. 5(a) only selects the last 1200 points to provide a clear illustration. It can be seen from Fig. 5 that the simulated output approaches the system output quite well.



(a) (b)

Fig. 5. The simulated performance of the EHH neural network for the Bouc-Wen benchmark. (a) The simulated outputs of the EHH neural network and the system output with multisine input. (b) The simulated outputs of the EHH neural network and the system output with swept-sine input.

5.2.2 ANOVA decomposition of the EHH approximation

From the EHH NARX model, the contributions of regressors as well as the interactions among different regressors are clear. Here as the input dimension is high, the ANOVA functions with the top 10 σ-values of the resulting EHH neural network is listed in Table 4.

Table 4
The ANOVA functions with the top 10 σ-values of the EHH neural network for the Bouc-Wen example.

ANOVA Fun.	Regressor	σ	\GCV	ANOVA Fun.	Regressor	σ	\GCV
1	$y(k-1)$	8.13×10^{-4}	15.16	6	$u(k-7)$	2.69×10^{-5}	2.07×10^{-2}
2	$y(k-4)$	1.38×10^{-4}	0.45	7	$y(k-11)$	2.40×10^{-5}	1.35×10^{-2}
3	$y(k-5)$	6.06×10^{-5}	8.64×10^{-2}	8	$y(k-13)$	2.10×10^{-5}	1.19×10^{-2}
4	$u(k-2)$	5.68×10^{-5}	9.19×10^{-2}	9	$y(k-10)$	1.74×10^{-5}	7.88×10^{-3}
5	$u(k-1)$	2.86×10^{-5}	2.23×10^{-2}	10	$y(k-1), y(k-9)$	1.31×10^{-5}	1.30×10^{-3}

From Table 4, we can know that the most recent output $y(k-1)$ contributes to the current output most, besides, there is input delay in this system, as the most important input is $u(k-2)$ and $u(k-1)$. From the regressors contained in the ANOVA functions, we can know that the regressors influence the output mostly additively.

6 Conclusions

6.1 Conclusions

In this work, the efficient hinging hyperplanes (EHH) neural network is proposed, which can be used for function approximation and dynamic system identification. The EHH neural network belongs to the HH family, and for every EHH neural network, an equivalent AHH tree exists. Besides, the EHH neural network is a distributed representation, hence can be used for large-scale problems. After an initial EHH neural network has been obtained, the EHH neural network can be trained through the structure and weights optimization, in which convex optimization problems are solved sequentially. The interpretability of the EHH neural network can be clearly obtained through the interaction matrix and the ANOVA decomposition. The EHH neural network is applied in dynamic system identification, which can be used as a tool for system approximation and an indication of regressor importance. Simulation results shows the effectiveness of the proposed procedure.

6.2 Future work

Although in theory, the EHH neural network is a distributed representation and can be applied to large-scale problem, the application to problems with hundreds of dimensions is limited due to the efficiency of the present algorithm. Therefore, we are dedicated to improving the current EHH neural network on solving extremely large-scale problems and at the same time, reduce the number of parameters employed.

Acknowledgements

This work is jointly supported by the National Natural Science Foundation of China (U1813224), Chinese Scholarship Council (CSC) and Science and Technology Innovation Committee of Shenzhen Municipality (JCYJ2017-0811-155131785). Johan Suykens acknowledges support of ERC Advanced Grant E-DUALITY (787960), KU Leuven C1, FWO G0A4917N.

References

[1] J. Belz, T. Münker, T. O. Heinz, G. Kampmann, and O. Nelles. Automatic modeling with local model networks for benchmark processes. In *20th IFAC World Congress*, pages 460–465, 2017.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[3] L. Breiman. Hinging hyperplanes for regression, classification and function approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, 1993.

[4] Y. Chikkula and J. H. Lee. Dynamically scheduled model predictive control using hinging hyperplane models. *AIChE Journal*, 44(12):2658–2674, 1998.

[5] P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerische Mathematik*, 31:377–403, 1979.

[6] A. F. Esfahani, P. Dreesen, K. Tiels, J. P. Noël, and J. Schoukens. Parameter reduction in nonlinear state-space identification of hysteresis. *Mechanical Systems and Signal Processing*, 104:884–895, 2018.

[7] J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991.

[8] X. Gao, Y. Jiang, T. Chen, and D. Huang. Optimizing scheduling of refinery operations based on piecewise linear models. *Computers & Chemical Engineering*, 75:105–119, 2015.

[9] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, volume 28, pages III–1319–III–1327, 2013.

[10] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[11] X. Huang, S. Mehrkanoon, and J. A.K. Suykens. Support vector machines with piecewise linear feature mapping. *Neurocomputing*, 10(117):118–127, 2013.

[12] X. Huang, J. Xu, and S. Wang. Operation optimization for centrifugal chiller plants using continuous piecewise linear programming. In *2010 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1121–1126, 2010.

[13] M. Ismail, F. Ikhouane, and J. Rodellar. The hysteresis Bouc-Wen model, a survey. *Archives of Computational Methods in Engineering*, 16(2):161–188, 2009.

[14] P. Julián, A. Desages, and O. Agamennoni. High-level canonical piecewise linear representation using a simplicial partition. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, 46(4):463–480, 1999.

[15] X. Liang and J. Xu. Decoupled ReLU neural network and its application in nonlinear system identification. Submitted to IFAC 2020.

[16] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

[17] K. S. Narendra. Neural networks in control systems. In P. Smolensky, M. C. Mozer, and D. E. Rumelhart, editors, *Mathematical perspectives on neural networks*, pages 347–394. London: Lawrence Erlbaum Associates, 1996.

[18] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Automatic Control*, 1(1):4–27, 1990.

[19] J. P. Noël and M. Schoukens. Hysteretic benchmark with a dynamic nonlinearity. In *Workshop on Nonlinear System Identification Benchmarks*, pages 7–14, Brussels, Belgium, 2016.

[20] M. Schoukens and F. G. Scheiwe. Modeling nonlinear systems using a Volterra feedback model. In *Workshop on Nonlinear System Identification Benchmarks*, 2016.

[21] J. Sjöberg, H. Hjalmarsson, and L. Ljung. Neural networks in system identification. In *Proceedings of the 10th IFAC Symposium on System Identification*, pages 49–72, 1994.

[22] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, et al. Nonlinear black-box modeling in system identification: A unified overview. *Automatica*, 31(12):1671–1724, 1995.

[23] J. A. K. Suykens, B. L. De Moor, and J. Vandewalle. Nonlinear system identification using neural state space models, applicable to robust control design. *International Journal of Control*, 62(1):129–152, 1995.

[24] J. A. K. Suykens, J. P. L. Vandewalle, and B. L. de Moor. *Artificial neural networks for modelling and control of non-linear systems*. Springer Science & Business Media, 2012.

[25] S. Wang and X. Sun. Generalization of hinging hyperplanes. *IEEE Transactions on Information Theory*, 12(51):4425–4431, 2005.

[26] C. Wen, S. Wang, X. Jin, and X. Ma. Identification of dynamic systems using piecewise-affine basis function models. *Automatica*, 43(10):1824–1831, 2007.

[27] D. T. Westwick, G. Hollander, K. Karami, and J. Schoukens. Using decoupling methods to reduce polynomial NARX models. *IFAC-PapersOnLine 18th IFAC Symposium on System Identification, SYSID*, 51(15):796–801, 2018.

[28] J. Xu, X. Huang, X. Mu, and S. Wang. Model predictive control based on adaptive hinging hyperplanes model. *Journal of Process Control*, 22(10):1821–1831, 2012.

[29] J. Xu, X. Huang, and S. Wang. Adaptive hinging hyperplanes and its applications in dynamic system identification. *Automatica*, 45(10):2325–2332, 2009.

[30] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.