Towards Deep Adaptive Hinging Hyperplanes

Qinghua Tao, Jun Xu, Zhen Li, Na Xie, Shuning Wang, Xiaoli Li, Johan A.K. Suykens, *Fellow, IEEE*

***Abstract*—The adaptive hinging hyperplanes (AHH) model is a popular piecewise linear representation with a generalized tree structure, and has been successfully applied in dynamic system identification. In this paper, we aim to construct the deep AHH (DAHH) model to extend and generalize the networking of AHH model for high dimensional problems. The network structure of DAHH is determined through a forward growth, in which the activity ratio is introduced to select effective neurons and no connecting weights are involved between layers. Then all neurons in the DAHH network can be flexibly connected to the output in a skip-layer format, and only the corresponding weights are the parameters to optimize. With such network framework, the back-propagation algorithm can be implemented in DAHH to efficiently tackle large scale problems and the gradient vanishing problem is not encountered in the training of DAHH. In fact, the optimization problem of DAHH can maintain convexity with convex loss in the output layer, which brings natural advantages in optimization. Different from the existing neural networks, DAHH is easier to interpret, where neurons are connected sparsely and ANOVA decomposition can be applied, facilitating to revealing the interactions between variables. Theoretical analysis towards universal approximation ability and explicit domain partitions are also derived. Numerical experiments verify the effectiveness of the proposed DAHH.**

***Index Terms*—Adaptive hinging hyperplanes, piecewise linear, skip-layer connection, domain partition, ANOVA decomposition.**

I. Introduction

DEEP neural networks (NN) have been widely applied and have achieved significant breakthroughs in various applications [1]–[5]. In deep NNs, the flexibility of networks greatly relies on their activation functions, which introduce nonlinearity to flexibly approximate complicated systems. In recent years, the rectifier linear unit (ReLU) was proposed and became one of the most popular activation functions since it can relieve gradient difficulties in back-propagation [6]. In essence, ReLU is a piecewise linear (PWL) function, meaning that it appears different linear expressions in different subregions. In fact, the neuron output of ReLU is equivalent to the hinging hyperplanes (HH), which is a popular model in PWL representation theory [7]. Then, many variants of ReLU were proposed, such as leaky ReLU [8], parametric ReLU [9], adaptive ReLU [10], etc [11], [12]. These variants can also be equivalently formulated as HH models. In [13], Goodfellow et al. proposed the Maxout network and achieved state-of-the-art performance in some benchmarks. The activation of the Maxout network is closely related to the generalized hinging hyperplanes (GHH), another important PWL model [14].

Although the deep structures have shown to be powerful, the existing deep NNs are generally considered as complicated black-boxes, which are difficult to interpret and analyze. The common way to achieve high accuracy and flexibility in such NNs is by simply stacking more layers, bringing more obstacles in interpretation and optimization.

In PWL representations, the adaptive hinging hyperplanes (AHH) model was proposed [15]. Similar to HH and GHH, AHH model consists of a linear combination of PWL basis functions, but differently it is built based on a generic tree topology, which is more interpretable than that of HH and GHH. The existing AHH is trained in an incremental manner, where the model is constructed progressively. In each iteration, it requires exhaustive searches on domain partitions by solving least squares (LS) problems repeatedly to generate two new basis functions, which is very computationally expensive [15]. Afterwards, another round of exhaustive searches is performed to identify the redundant basis functions and delete them to prevent over-fitting. Fast AHH (FAHH) was proposed to reduce the exhaustive searches and the least absolute shrinkage and selection operator (LASSO) was introduced to replace the exhaustive pruning, but the improvement of FAHH is still limited [16], [17]. Then, the efficient hinging hyperplanes (EHH) model was constructed to further improve the efficiency of AHH to formulate a directed acyclic graph, and achieved some improvement in dynamic system identification [18]. EHH inherits the method in FAHH to optimize parameters and also conducts the structure optimization, thus it still needs exhaustive searches to progressively generate the model with the lowest objective function in each iteration, where LASSO problems need to solve repeatedly. Hence, EHH is still prohibitive in efficiency and lacks extendability to various tasks [19].

In this paper, we revisit both AHH and EHH, and aim to bring a generalized neural network based on AHH by constructing the deep AHH (DAHH) neural network. The proposed DAHH can be regarded as an attempt to extend the PWL representation model of AHH to apply for high dimensional regression and classification problems.

DAHH is established as a compact neural network without harming the interpretability of AHH and makes the training

This work is jointly supported by National Natural Science Foundation of China U181322, 71772195, and 61873006, National Key Research and Development Project (2018YFC1602704, 2018YFB1702704), ERC Advanced Grant E-DUALITY (787960), KU Leuven Grant CoE PFV/10/002, and Grant FWO G0A4917N. *(Corresponding author: Na Xie.)*

Qinghua Tao is with STADIUS, ESAT, KU Leuven, Belgium and the Department of Automation, Tsinghua University, Beijing 100084, China, email: qinghua.tao@esat.kuleuven.be.

Jun Xu and Zhen Li are from School of Mechanical Engineering and Automation, Harbin Institute of Technology, Shenzhen 518055 China, email: xujunqgy@hit.edu.cn, zueslee.hitsz@foxmail.com.

Na Xie is with School of Management Science and Engineering, Central University of Finance and Economics, Beijing 100081, China, email: xiena@cufe.edu.cn.

Shuning Wang is with the Department of Automation, Tsinghua University, Beijing 100084, China, email: swang@tsinghua.edu.cn.

Xiaoli Li is with Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China, email: lixiaolibjut@bjut.edu.cn.

Johan A.K. Suykens are with STADIUS, ESAT, KU Leuven, Belgium, email: johan.suykens@esat.kuleuven.be.

techniques in deep NNs well fitted. In the forward growth of network structure, DAHH makes a trade-off between exhaustive searches in AHH and EHH by incorporating the concept of activity ratio to obtain effective neurons, which resembles the generation of leaf nodes in decision trees and random forests. In the forward growth, neurons are connected in a skip-layer format with "$\min\{\cdot,\cdot\}$" activations and no connecting weight is involved. Then all neurons from each layer can also be flexibly connected to the output by skipping layers, completing the DAHH network.

Under the proposed structure diagram, back-propagation algorithm can be easily implemented in DAHH, which can tackle large scale problems and flexibly extend the current LS residuals to different loss functions, such as the cross entropy for classification. In training DAHH, we only need to optimize the weights connecting the output, which not only avoids the gradient difficulty in back-propagation, but also maintains convexity with convex losses in the output layer, bringing natural advantages in optimization. Meanwhile, the backward pruning is also employed to reduce redundancy resulted from forward growth, in which the neurons with zero connecting weights can be easily pruned to obtain more concise structures.

Compared to the typical fully connected NNs or multilayer percetrons (MLP), DAHH has a sparser structure, which facilitates detecting the interactions among variables in each neuron and extracting the explicit domain partitions and locally linear expressions. Theoretical analysis of the approximation ability is also given. Numerical experiments are then conducted to verify the effectiveness of the proposed DAHH.

The rest of the paper is organized as follows. Section II briefly introduces related PWL representations and activations in NNs. Section III introduces the forward growth of DAHH network structure and Section IV proposes the training method for DAHH. In Section V, detailed theoretical analysis of DAHH is provided. Section 6 reports the numerical experiments. Finally, Section VII ends the paper with brief conclusions.

II. Background

A. PWL Activations and Hinging Hyperplanes Models

Among the existing activations in NNs, ReLU is one of the most popular ones, and is written as $\max\{0, x\}$. In fact, the neuron output of ReLU activation is equivalent to the basis function in HH model, which is formulated as

$$f_{\text{HH}}(\boldsymbol{x}) = \sum_{m=1}^{M} w_m \max\{\boldsymbol{a}_m^T \boldsymbol{x} + b_m, 0\}, \tag{1}$$

where $w_m, b_m \in \mathbb{R}$, $\boldsymbol{a}_m, \boldsymbol{x} \in \mathbb{R}^d$ and M is the number of basis functions (hidden units) [7].

Analogously, the Maxout activation is then proposed, and it is expressed as $\max_{i\in I}\{z_i\}$, where I is an index set [13]. As ReLU to HH, Maxout closely resembles GHH, i.e.,

$$f_{\text{GHH}}(\boldsymbol{x}) = \sum_{m=1}^{M} w_m \max\{\boldsymbol{a}_{m,1}^T \boldsymbol{x} + b_{m,1}, \ldots, \boldsymbol{a}_{m,i_m}^T \boldsymbol{x} + b_{m,i_m}\} \tag{2}$$

where i_m is the number of linear functions contained in the mth basis function [14].

B. AHH

In PWL representation theory, AHH is proposed , such that

$$\begin{aligned} f_{\text{AHH}}(\boldsymbol{x}) &= w_0 + \textstyle\sum_{m=1}^{M} w_m z_m(\boldsymbol{x}) \\ z_m(\boldsymbol{x}) &= \min_j\{\max\{0, \delta_{j,m}(x_{v_{j,m}} - \beta_{j,m})\}\}, \end{aligned} \tag{3}$$

where $v_{j,m} \in J_m$, $J_m \subseteq \{1, \ldots, d\}$, $\delta_{j,m} = \pm 1$, $x_{v_{j,m}}$ is the $v_{j,m}$th variable of $\boldsymbol{x} \in \mathbb{R}^d$, and $\beta_{j,m}$ is the splitting knot on variable (dimension) $x_{v_{j,m}}$ [15].

Different from HH and GHH, variables $x_1, \ldots, x_d$ are coupled by nested $\min\{\cdot\}$ and $\max\{\cdot\}$ operators in AHH, instead of a linear combination $\boldsymbol{a}^T\boldsymbol{x} + b$. Meanwhile, the variables in each basis function can also be different and need to be identified. The existing identification method of AHH is developed in an incremental manner to generate basis functions, which can be interpreted as the recursive domain partitions, where each basis function corresponds to a subregion. Specifically in each iteration, it sequentially chooses one of the existing basis functions (subregions) as the root, and then traverses all the candidate knots in each dimension to select the one with the sharpest error decrease to split, leading to the generation of two new basis functions. A simple example is presented in Fig. 1.

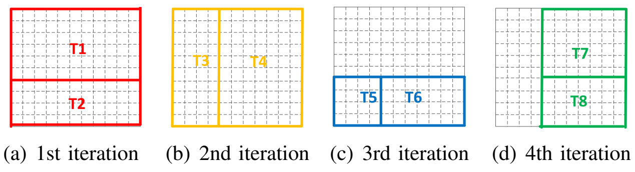
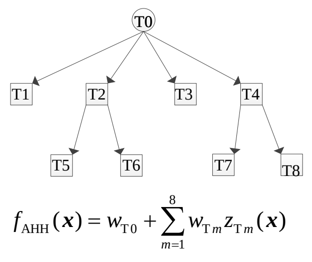


Fig. 1: An simple illustration on the geometrical description of $f_{\text{AHH}}(\boldsymbol{x}) = \sum_{m=1}^{8} w_m z_m(\boldsymbol{x})$, where $z_1(\boldsymbol{x}) = \max\{0, x_2 - 0.4\}, z_2(\boldsymbol{x}) = \max\{0, 0.4 - x_2\}, z_3(\boldsymbol{x}) = \max\{0, 0.4 - x_1\}, z_4(\boldsymbol{x}) = \max\{0, x_1 - 0.4\}, z_5(\boldsymbol{x}) = \min\{z_2, z_3\}, z_6(\boldsymbol{x}) = \min\{z_2, z_4\}, z_7(\boldsymbol{x}) = \min\{z_1, z_4\}, z_8(\boldsymbol{x}) = \min\{z_2, z_4\}$, where $z_m(\boldsymbol{x})$ can be regarded to corresponds to a subregion T_m.

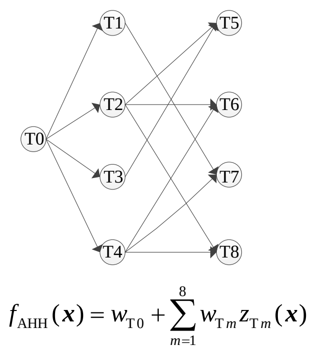
In Fig. 1, each iteration traverses all the existing basis functions in all candidate splitting knots along x_1 and x_2, and then select the best split to generate two new basis functions. For instance, the 4th iteration sequentially chooses the existing 6 basis functions $z_1, \ldots, z_6$ (concerning $\text{T}_1, \ldots, \text{T}_6$) as the root, and traverses all candidate splits $x_1 = 0, 0.1, \ldots, 0.9$ and $x_2 = 0, 0.1, \ldots, 0.9$, where their corresponding parameters all need to be computed for $6 \times (10 + 10) = 120$ times by LS and then the split with the lowest fitting error ($x_2 = 0.4$) is selected, generating z_7 and z_8.

The above recursive partition can be interpreted as a generic tree with basis functions $z_m(\boldsymbol{x})$ as nodes, whose linear combination is the output. In the tree topology, the relations among variables are not depicted explicitly and the information of previously generated basis functions is not easy to be reused. In fact, such tree topology can be equivalently performed as a special network [18]. Fig. 2 gives the topology of the AHH model described in Fig. 1.

Similar with the exhaustive searching in AHH, EHH also requires to traversing all candidate splits to obtain the model


T0
T1
T2
T3
T4
T5
T6
T7
T8
$f_{\mathrm{AHH}}(\boldsymbol{x}) = w_{\mathrm{T}0} + \sum_{m=1}^{8} w_{\mathrm{T}m} z_{\mathrm{T}m}(\boldsymbol{x})$

(a) Tree topology.


T0
T1
T2
T3
T4
T5
T6
T7
T8
$f_{\mathrm{AHH}}(\boldsymbol{x}) = w_{\mathrm{T}0} + \sum_{m=1}^{8} w_{\mathrm{T}m} z_{\mathrm{T}m}(\boldsymbol{x})$

(b) Network topology.

Fig. 2: Topology of AHH model depicted in Fig. 1.

by repeatedly solving multiple LASSO problems and checking their corresponding objective values, and then selects the best one. Thus it is still prohibitive in efficiency and extendability to large scale problems in various tasks.

III. Deep Adaptive Hinging Hyperplanes Neural Networks

In this section, inspired by the PWL representation of AHH, we aim at constructing the neural network based on AHH to tackle large scale problems in regression and classification with efficient network structures and flexible training methods.

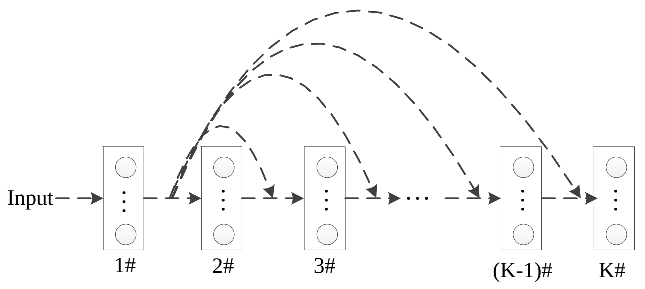
A. Sketch of DAHH Neural Network

In this paper, we aim at constructing and training a novel neural network based on AHH. The existing AHH and EHH are applicable to low dimensions with a few variables in basis functions, while the proposed network focuses on large scale problems with deeper structure. Thus, we name it as DAHH.

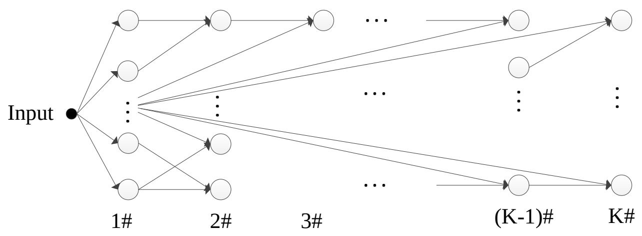
DAHH performs the forward growth to generate the network structure and the output. Then, backward training is employed to efficiently train DAHH. The diagram of the forward growth to obtain DAHH network structure is shown in Fig. 3.

In Fig. 3(a), the dashed lines indicate the data flow and the generation of network layers, where the data pass through the DAHH network via layers. Different from the standard MLP, there exist skip-layer connections in DAHH, where the initial layer ($1\#$) is connected with each subsequent layer. In the geometrical interpretation of DAHH, such skip-layer connections give further feature space partitions in generating more flexible PWL neurons. In the deep residual neural network, the concept of skip-layer connection is also adopted [5]. Differently, there is no connecting weight in the forward growth of DAHH, and such skip-layer connection is an intrinsic property from DAHH itself instead of extra modifications.

In Fig. 3(b), the solid lines are the connections among neurons. For representation simplicity, the output connection is omitted in Fig. 3(b) and will be introduced with details in Section III-A3. Fig. 3(b) illustrates that the neurons are connected sparsely, where the neurons in layer $K_i\#$ after the initial layer are only connected with one neuron from the previous layer $(K_i-1)\#$ and another one from the initial layer



(a) Forward growth of network structure.

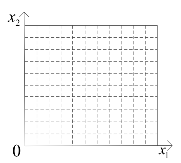


(b) Neuron connections in the forward growth.

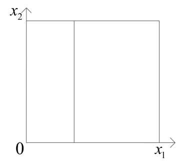
Fig. 3: Diagram of the forward procedure in generating DAHH network, where the output connections are omitted for expression simplicity.

$1\#$. The mathematical expressions concerning each component of DAHH, and the motivation and scientific basis of such structure are presented in Section III-A1 and Section III-A2.

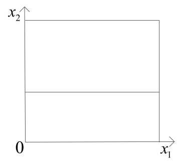
1) Motivation to the diagram of DAHH: In DAHH, the basic components constituting the neurons generate from the initial layer $1\#$ consisting of the units $z_{1,s}$, which initially partition the domain along one dimension, where $s \in \{1, 2, \ldots, n_1\}$ is the index of the neuron and n_1 is the number of neurons in the initial layer. Taking Fig 4 for illustration, the resulting domain partitions from the initial layer are the dashed lines over each dimension (x_1 and x_2) in Fig. 4(a), where the input domain is assumed to normalized into $[0, 1]^2$. Fig. 4(b) and Fig. 4(c) are two examples of the partition in the initial layer.



(a) Total partitions



(b) $x_1 = 0.4(z_1, z_2)$

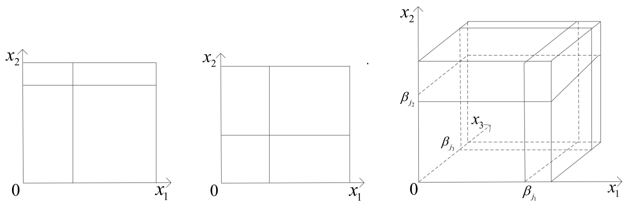


(c) $x_2 = 0.4(z_1, z_2)$

Fig. 4: An illustration on domain partitions of the initial layer in DAHH.

Fig 4 shows that the neurons in the initial layer initially partitions the domain along one dimension, i.e., $x_1, x_2 = 0, \ldots, 0.9$ with $n_1 = 20$. For instance, the partition $x_1 = 0.4$ in Fig. 4(b) can lead to neurons $z_1(\boldsymbol{x}) = \max\{0, x_2 - 0.4\}$ and $z_2(\boldsymbol{x}) = \max\{0, 0.4 - x_2\}$. Similarly, $x_2 = 0.4$ in Fig. 4(c) leads to neurons $z_3(\boldsymbol{x}) = \max\{0, 0.4 - x_1\}$ and $z_4(\boldsymbol{x}) = \max\{0, x_1 - 0.4\}$.

In PWL representations, model flexibility increases when the domain is partitioned into more subregions, each of which appears different linear response and then formulates the final PWL output. Therefore, analogous to the partitions of the initial layer, we can further partition the domain to obtain more flexible neurons, which motivates to generate the subsequent layers in DAHH. For instance, Fig 4(b) can be further partitioned on x_2, such as $x_2 = 0.8$ in Fig 5(a), and Fig 4(c) can be partitioned by adding $x_1 = 0.4$ in Fig 5(b). Without loss of generality, we can also give an illustration on the domain partitions in the 3-dimensional space over a cube $[0,1]^3$, which is presented in Fig 5(c), where the partitions are $x_1 = \beta_{j_1}$, $x_2 = \beta_{j_2}$ and $x_3 = \beta_{j_3}$.



(a) Fig 4(b) split by $x_2 = 0.8$ (b) Fig 4(c) split by$x_1 = 0.4$ (c) A cube split by $x_i = \beta_{j_i}$

Fig. 5: An illustration on domain partitions of the neurons in DAHH.

As demonstrated in Fig 4 and Fig 5, the mechanism of flexible domain partitions motivates us to attain the structure of the proposed DAHH in this paper, i.e., connecting the initial layer with the current layer to formulate the neurons with deeper partitions over the domain. According to the diagram in Fig. 3, the following neurons $z_{k,s}(k\#, k \geq 2)$ are constructed based on further partitioning the subregions represented by the $(k-1)$th layer along an extra dimension by a neuron from the initial layer, bringing more subregions and approaching a more flexible PWL neuron output. Therefore, the subsequent layers $(k\#, k \geq 2)$ are formulated as $z_{k,s}(\boldsymbol{x}) = \min\{z_{1,q_{s,1}}(\boldsymbol{x}), z_{k-1,q_{s,2}}(\boldsymbol{x})\}$, where neuron $z_{k,s}$ is connected (generated) with $z_{1,q_{s,1}}$ and $z_{k-1,q_{s,2}}$ located in the 1st layer and the $(k-1)$th layer, meaning that neuron $z_{k,s}$ gives deeper partitions to $z_{k-1,q_{s,2}}$ on $z_{1,q_{s,1}}$. $q_{s,1}$ and $q_{s,2}$ are the corresponding indices (neuron connections) with $q_{s,1} \in \{1,\ldots,n_1\}$, $q_{s,2} \in \{1,\ldots,n_{k-1}\}$, where n_1 and n_{k-1} denote the numbers of neurons in the 1st layer and the $(k-1)$th layer, respectively. Then, these generated neurons in all layers can be connected and contribute to the final flexible PWL network, i.e., DAHH.

2) Network Layers and their neurons in DAHH: In DAHH, we denote $z_{k,s}$ as the sth neuron in the kth layer, where n_k is the number of neurons in the kth layer with $s \in \{1,\ldots,n_k\}$. Correspondingly as n Section III-A1, neurons in the kth $(k \geq 2)$ layer are sparsely connected with the $(k-1)$th layer and the initial 1st layer. Thus, the neuron outputs can be cast as follows.

- The initial (1st) layer:

$$z_{1,s}(\boldsymbol{x}) = \max\{\delta_{1,s}(x_{v_{1,s}} - \beta_{1,s}), 0\}, \tag{4}$$

where $\delta_{1,s} = \pm 1$, $s \in \{1,2,\ldots,n_1\}$, $v_{1,s} \in \{1,\ldots,d\}$, $\beta_{1,s}$ is the splitting knot and $x_{v_{1,s}}$ is the splitting variable in neuron $z_{1,s}$. For instance, assuming the 5th neuron in the initial layer as $z_{1,5}(\boldsymbol{x}) = \max\{x_3 - 0.7, 0\}$, we have $\beta_{1,5} = 0.7$ as the splitting knot and $v_{1,5} = 3$ as the index of splitting variable.

- The kth layer $(k \geq 2)$:

$$\begin{aligned} z_{k,s}(\boldsymbol{x}) = & \min\{z_{1,q_{s,1}}(\boldsymbol{x}), z_{k-1,q_{s,2}}(\boldsymbol{x})\}, \\ = & \min_{v_{s_1},\ldots,v_{s_k} \in J_{k,s}}\{\max\{\delta_{s_1}(x_{v_{s_1}} - \beta_{s_1}), 0\}, \\ & \ldots, \max\{\delta_{s_k}(x_{v_{s_k}} - \beta_{s_k}), 0\}\}, \end{aligned} \tag{5}$$

where $\delta_{k,s} = \pm 1$ and the cardinality of $J_{k,s}$ is $|J_{k,s}| = k$. $J_{k,s} = \{v_{s_1},\ldots,v_{s_k}\}$ denots the set which contains the indices of all the splitting dimensions (variables) $x_{v_{s_1}},\ldots,x_{v_{s_k}}$ contained in neuron $z_{k,s}$, where $\{v_{s_1},\ldots,v_{s_k}\} \subseteq \{1,\ldots,d\}$ with $v_{s_i} \in \{1,\ldots,d\}, i = 1,\ldots,k$, meaning that k variables are interacted in the sth neuron of the kth layer, i.e., neuron $z_{k,s}$ gives multiple splits on k dimensions.

Formula (4) and (5) show that the activations in DAHH network are only "max" and "min" operators. Different from MLP, the neurons in DAHH are connected sparsely with only two connections, in which one neuron can be connected in a skip-layer format from the initial layer. Besides, no connecting weights and hyperparameters are involved in the forward growth, which greatly facilitates the optimization of DAHH, shown in Section IV with details.

3) Output Connections: After the forward growth of generating DAHH network structure, the output can also be formulated with skip-layer connections, and thereby all the neurons can be connected to the output, such that $f_{\text{DAHH}}(\boldsymbol{x}) = \sum_k \sum_s w_{k,s} z_{k,s}(\boldsymbol{x})$, where the combination of these neurons lead to a flexible PWL output, i.e., the DAHH network.

In practice, for some high dimensional problems in which the interactions among variables are complicated, the neurons in shallow layers can have trivial influence to the output directly, while only deep layers in DAHH play critical roles. Hence, DAHH generalizes the output connection of networking AHH, namely the full skip-layer and partial skip-layer output, i.e.,

$$\begin{aligned} &\textbf{Full skip-layer:} && f_{\text{DAHH}}(\boldsymbol{x}) = \textstyle\sum_{k=1}^{K} \sum_s w_{k,s} z_{k,s}(\boldsymbol{x}), \\ &\textbf{Partial skip-layer:} && f_{\text{DAHH}}(\boldsymbol{x}) = \textstyle\sum_{k=K_{\text{out}}}^{K} \sum_s w_{k,s} z_{k,s}(\boldsymbol{x}). \end{aligned} \tag{6}$$

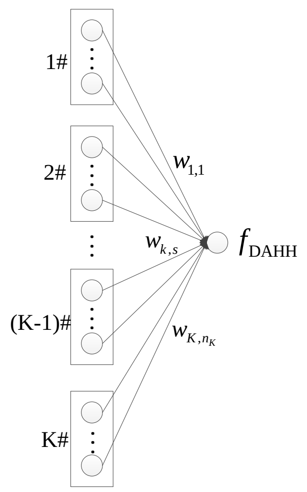
The topology is shown in Fig. 6. In fact, the output of an MLP is a special case of partial skip-layer connection with $K_{\text{out}} = K$.

B. *Forward Growth of DAHH Network Structure*

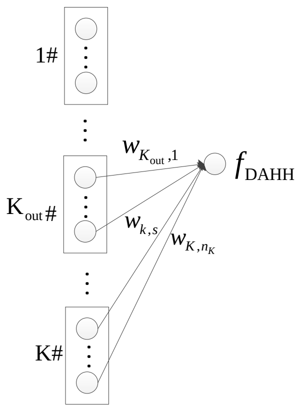
In the forward growth of DAHH, neurons are generated in each layer. Instead of exhaustive searches to find the optimal split $x_{v_{k,s}}$ and $\beta_{k,s}$ in AHH and EHH, DAHH revisits the neuron property in Section III-A, and makes a trade-off between optimal exhaustive searches and computational complexity to select effective neurons.

In decision trees, each leaf node is generated by checking through the splitting variables and knots to find the optimal one, similar to the existing exhaustive searches in AHH. Since the neuron output in DAHH is formulated as formula (5), the neuron $z_{k,s}$ can only be activated with nonzero outputs on the condition that at least one component $\max\{\delta_{s_i}(x_{v_{s_i}} - \beta_{s_i}), 0)\}$ satisfies $\delta_{s_i}(x_{v_{s_i}} - \beta_{s_i}) > 0$. Otherwise, it always outputs $z_{k,s}(\boldsymbol{x}) = 0$ and makes no contribution to network output.

In decision trees, the data contained in (activated by) the leaf nodes can usually be imposed with a confinement to



(a) Full skip-layer output.



(b) Partial skip-layer output.

Fig. 6: Topology of output connections in DAHH network.

enhance the performance. Inspired by that, we introduce the activity ratio r_{act} to select the neurons in the forward growth, meaning that a certain number of training data are required to be activated by the neurons. Thus, DAHH proposes to select more effective neurons $z_{k,s}$ which suffices

$$\frac{C\{z_{k,s}(\boldsymbol{x}^i)\}_{i=1}^N}{N} \geq r_{\text{act}}, \tag{7}$$

where $C\{\cdot\}$ denotes the number of training data $\boldsymbol{x}^i$ activated by the neuron $z_{k,s}(\boldsymbol{x}^i) > 0$. Smaller value of r_{act} brings sparser neuron outputs with the given training data.

With the activity ratio r_{act} in formula (7), DAHH abandons the neurons which are activated only by a trivial proportion of data and are prone to bring less influence on the network output. In fact, the proposed activity ratio r_{act} in DAHH also resembles the mechanism behind random forest. To generate the decisions trees in the forest, random forest method replaces the original exhaustive traversal by only selecting the most potential ones among the subset which is randomly chosen from the complete splits of the trees.

A simple example is tested on MNIST image classification dataset [20] to evaluate the effectiveness of the proposed activity ratio. Two DAHH networks are constructed both of depth $K = 6$. DAHH1 and DAHH2 have simple structures, where the number of weights is similar with that of a single-layer MLP consisting of 10 to 20 hidden units. For DAHH1 (AR) and DAHH2 (AR), "AR" means that the activity ratio $r_{\text{act}} = 0.1$ is used. Tables I gives the average of 20 runs, in which the proposed activity ratio strategy is shown to bring more effective neurons for better performance. To simplify the tuning, we fix $r_{\text{act}} = 0.1$ for all numerical experiments, showing as a satisfactory choice for various cases. In this paper, $r_{\text{act}} = 0.1$ is suggested as the defaulting setting.

In the forward growth of DAHH, no parameter needs to be estimated, since the splitting knots $\beta_{1,s}$ of the univariate units $z_{1,s}$ in the initial layer are pre-allocated and there are no connecting weights between neurons, such that

TABLE I: Example for verifying activity ratio strategy.

Dataset	DAHH1	DAHH1 (AR)	DAHH2	DAHH2 (AR)
MNIST	95.57%	**97.48%**	95.11%	**96.87%**

$z_{k,s}(\boldsymbol{x}) = \min\{z_{1,q_{s,1}}(\boldsymbol{x}), z_{k-1,q_{s,2}}(\boldsymbol{x})\}$. Thus, we only need to determine the network structure of DAHH in the forward growth, whose details are summarized in Algorithm 1.

Algorithm 1 Forward Growth of DAHH Network Structure

```
Input: Network structure budgets $K$ and $n_k$, pre-allocated splitting knots $\beta = \{\beta_1, \ldots, \beta_{n_1}\}$ and $r_{\text{act}}$.
Output: DAHH network with neurons $z_{k,s}$.
• Generate neurons $z_{1,s}, s = 1, \ldots, n_1$ by formula (4) such that $z_{1,s}(\boldsymbol{x}) = \max\{\delta_{1,s}(x_{v_{1,s}} - \beta_{1,s}), 0\}, \beta_{1,s} \in \beta$.
for $k = 2$ to $K$ do
    for $s = 1$ to $n_k$ do
        • Generate the neuron $z_{k,s}$ by formula (5) $z_{k,s}(\boldsymbol{x}) = \min\{z_{1,q_{s,1}}(\boldsymbol{x}), z_{k-1,q_{s,2}}(\boldsymbol{x})\}$ which satisfies formula (7) with activity ratio $r_{\text{act}}$, i.e., $\frac{C\{z_{k,s}(\boldsymbol{x}^i)\}_{i=1}^N}{N} \geq r_{\text{act}}$.
    end for
end for
• Obtain the DAHH network structure with neurons $z_{k,s}$ in each layer $k = 1, \ldots, K$.
```

In Algorithm 1, similar with MLP, the tuning parameters are the determination of network structure, i.e., the depth K of the network, the number of neurons $n_k, k = 1, \ldots, K$ in each layer and the number of splitting knots β. In DAHH, the splitting knots are pre-allocated on. More splits bring more subregions in the initial layer with increasing parameters. Since the following layer can further partition the feature space into smaller subregions, the number of splitting knots in the initial layer are not necessarily required to set very large. In our previous works [15], [16], the number of splitting knots is suggested to set within $[5, 10]$, which can generally bring satisfactory results. Thus, in this manuscript, the defaulted number of splitting knots in each dimension is set as 7. Inheriting from the AHH tree, the following Section IV introduces the backward pruning in DAHH to trim the network with proper size. Thus, the structure setting in the forward growth can be regarded as the budget of DAHH network rather than the precise final structure, where only the effective neurons can be kept after the backward pruning. For simplicity, the number of neurons can be set as the same, such that $n_1 = \ldots = n_K$. For the depth K, we can set from a very small number, since it has been noticed that the interaction level K among variables are usually distinctively less than the dimensionality d of the data. The numerical details are presented in Section VI.

IV. Training Method of DAHH Neural Network

The optimization problem of DAHH training is cast as

$$\min_{w_{k,s}} \mathcal{L}(w_{k,s}; \boldsymbol{x}^i, \boldsymbol{y}^i, \beta) + \lambda \mathcal{P}(w_{k,s}), \tag{8}$$

where $\boldsymbol{x}^i \in \mathbb{R}^d, \boldsymbol{y}^i \in \mathbb{R}^l, i = 1, \ldots, N$ are the training data and labels, β is the set containing the splitting knots, the $w_{k,s}$ is the weight of neuron $z_{k,s}$ connecting the output layer, i.e., formula (6). $\mathcal{L}(\cdot)$ is the loss function and $\mathcal{P}(\cdot)$ is the regularization with coefficient λ.

A. Back-Propagation Framework

Though the proposed DAHH network has special structures with sparse neuron connections, it is still a compact neural network, in which the back-propagation framework can be well fitted. Hence, the stochastic gradient descent algorithm and batch-wise optimization can be applied for tackling large scale problems. The back-propagation algorithm in DAHH is performed by following a gradient descent direction

$$\Delta w_{k,s} = -\eta\left(\frac{\partial\mathcal{L}(w_{k,s};\boldsymbol{x}^i,\boldsymbol{y}^i,\beta)+\partial\mathcal{P}(w_{k,s})}{\partial w_{k,s}}\right), \tag{9}$$

in which η is the learning rate.

Obviously, the derivative of regularizer $\mathcal{P}(\cdot)$ is easy to compute, thus we pay more emphasis on the part of loss function $\mathcal{L}(\cdot)$. Assuming $\sigma(\cdot)$ as the activation function of the output layer, we have the derivative of the loss $\mathcal{L}(\cdot)$, i.e.,

$$\begin{aligned}\frac{\partial\mathcal{L}(w_{k,s};\boldsymbol{x},\boldsymbol{y},\beta)}{\partial w_{k,s}} &= \frac{\partial\mathcal{L}(w_{k,s};\boldsymbol{x},\boldsymbol{y},\beta)}{\partial\sigma(\sum_{k,s}w_{k,s}z_{k,s}(\boldsymbol{x}))}\frac{\partial\sigma(\sum_{k,s}w_{k,s}z_{k,s}(\boldsymbol{x}))}{\partial w_{k,s}}\\ &= G\frac{\partial(\sum_{k,s}w_{k,s}z_{k,s}(\boldsymbol{x}))}{\partial w_{k,s}}\\ &= Gz_{k,s}(\boldsymbol{x}),\end{aligned} \tag{10}$$

where $G = \frac{\partial\mathcal{L}(w_{k,s};\boldsymbol{x},\boldsymbol{y},\beta)}{\partial\sigma(\sum_{k,s}w_{k,s}z_{k,s}(\boldsymbol{x}))}\frac{\partial\sigma(\sum_{k,s}w_{k,s}z_{k,s}(\boldsymbol{x}))}{\partial(\sum_{k,s}w_{k,s}z_{k,s}(\boldsymbol{x}))}$.

Equation (10) illustrates that only the derivative of the activation function in the output layer $\frac{\partial\sigma(x)}{\partial x}$ needs to compute and the chain rule is not necessary. Thus, the optimization process is quite simple and does not suffer from the gradient difficulties in back-propagation. Especially, when $\sigma(\cdot)$ and $\mathcal{P}(\cdot)$ are chosen as convex, the optimization problem (8) remains convex, since the convex function can maintain convexity with affine inputs $\sum_{k,s}w_{k,s}z_{k,s}(\boldsymbol{x})$.

Under such training framework, it is capable of handling large scale problems, and different loss functions $\mathcal{L}(\cdot)$ can also be flexibly extended in DAHH together with different regularization items $\mathcal{P}(\cdot)$ for various tasks, such as cross entropy for classification, l_1 norm and l_2 norm regularizers.

B. Selection of Splitting Knots

Besides the connecting weights $w_{k,s}$, the splitting knots β_{ij} within the neurons also determine the final DAHH. Without loss of generality, we assume that the domain is normalized as $[0,1]^d$. The original AHH determines the splitting knots β_{ij} by uniformly sampling each dimension, i.e., $\beta_{k1}=0,\ \beta_{k2}=1/e,\ \beta_{k3}=2/e,\ldots,\ \beta_{ke}=1-1/e$, where $k=1,\ldots,d$ and $1/e$ is the sampling interval [15].

In our previous works [16], an alternative is proposed to determine the splitting knots by utilizing the distribution of the training data, i.e., the quantiles in each dimension, which is more adaptive to the data distributions. Then, the splitting knots are generated as

$$\beta_{k1}=\hat{x}_k^{n_k,1},\ \beta_{k2}=\hat{x}_k^{n_k,2},\ \ldots,\ \beta_{ke}=\hat{x}_k^{n_k,e}. \tag{11}$$

In fact, under the back-propagation framework in Section IV-A, the splitting knots $\beta_{ij}\in\beta$ can be optimized together with the weights $w_{k,s}$, such that

$$\min_{w_{k,s},\beta_{ij}} \mathcal{L}(w_{k,s},\beta_{ij};\boldsymbol{x}^i,\boldsymbol{y}^i)+\lambda\mathcal{P}(w_{k,s}). \tag{12}$$

Analogously, the gradient of splitting knots β_{ij} can be computed as

$$\frac{\partial\mathcal{L}(\beta_{ij};\boldsymbol{x},\boldsymbol{y},w_{k,s})}{\partial\beta_{ij}} = G_1G_2\sum_{\hat{k},\hat{s}}w_{\hat{k},\hat{s}}\frac{\partial z_{\hat{k},\hat{s}}(\beta_{ij};\boldsymbol{x})}{\partial\beta_{ij}}, \tag{13}$$

where $\{\hat{k},\hat{s}\}$ denotes the neurons $z_{\hat{k},\hat{s}}$ containing β_{ij}, $G_1=\frac{\partial\mathcal{L}(\beta_{ij};\boldsymbol{x},\boldsymbol{y},w_{k,s})}{\partial\sigma(\sum_{k,s}w_{k,s}z_{k,s}(\beta_{ij};\boldsymbol{x}))}$ and $G_2=\frac{\partial\sigma(\sum_{k,s}w_{k,s}z_{k,s}(\beta_{ij};\boldsymbol{x}))}{\partial(\sum_{k,s}w_{k,s}z_{k,s}(\beta_{ij};\boldsymbol{x}))}$.

Though the splitting knots β_{ij} can be optimized by following the gradient in (13), the induced optimization problem (12) becomes highly non-convex and more complex than (8), thus the performance of the stochastic gradient descent algorithm can be affected. In Section VI-A1, numerical experiments are conducted to evaluate the optimization methods of the splitting knots β_{ij} and verifies the effectiveness of the method in (11). Therefore, in this paper, we suggest to use quantiles to determine the splitting knots in (11), and then therein tackle the relatively simpler optimization problem as (8).

C. Neuron Pruning

The forward growth of DAHH resembles the identification mechanism in AHH, which deliberately over-fits the data with an excessively large model for gaining greater flexibility. Afterwards, a backward pruning is incorporated to trim the model to a proper size.

Instead of exhaustive searchies in pruning AHH, we employ sparsity penalty to prune the model, where l_1 norm is chosen as the regularization item $\mathcal{P}(\cdot)$, i.e., $P(w_{k,s})=\sum_{k,s}|w_{k,s}|$ [16]. In DAHH network structure, the neurons with $w_{k,s}=0$ weights connecting the output layer are redundant and can be easily pruned in the output connection, which brings a more concise DAHH network.

Besides, with the neuron pruning process, the determination of network structures can also be benefited, since we only need to set the budgets of network depth K and the number of neurons $n_k, k=1,\ldots,K$ in each layer. The final network structure of DAHH is then obtained by pruning the redundant neurons. Therefore, the backward training process of DAHH network can be summarized as Algorithm 2.

Algorithm 2 Backward Training of DAHH Network

```
Input: DAHH network with neurons $z_{k,s}$ obtained from Algorithm 1, $\lambda$, $K_{\text{out}}$ and $F_{\text{full}}\in\{0,1\}$.
Output: Parameters $w_{k,s}$ of DAHH network.
if $F_{\text{full}}=1$ then
    • $K_{\text{out}}=1$.
end if
• Obtain the output $f_{\text{DAHH}}(\boldsymbol{x})=\sum_{k=K_{\text{out}}}^{K}\sum_s w_{k,s}z_{k,s}(\boldsymbol{x})$.
• Apply back-propagation algorithm with gradient in formula (9) to optimize the weights $w_{k,s}, k=K_{\text{out}},\ldots,K$.
• Prune the redundant output neurons $z_{k,s}$ with $w_{k,s}=0$ and obtain the final DAHH network.
```

V. Properties of DAHH Neural Network

A. Universal Approximator

A standard MLP with one hidden layer and enough hidden units is a universal approximator. The approximation ability of DAHH neural network is also proved by the following lemma.

Lemma 1. *Let C be a compact domain $\mathcal{C} \subset \mathbb{R}^n$, $f : \mathcal{C} \to \mathbb{R}$ be a continuous function. Given arbitrary positive real number $\epsilon > 0$, there exists a DAHH neural network $f_{\text{DAHH}}(\boldsymbol{x})$, such that for all $\boldsymbol{x} \in \mathcal{C}$, we have*

$$|f(\boldsymbol{x}) - f_{\text{DAHH}}(\boldsymbol{x})| < \epsilon. \tag{14}$$

Proof. For each neuron $z_{k,s}$ in DAHH network, the neuron output can be equivalently expressed as

$$z_{k,s}(\boldsymbol{x}) = \min_{j \in \mathcal{K}_{1,s}} \{z_{1,j}(\boldsymbol{x})\}, \tag{15}$$

where $\mathcal{K}_{1,s}$ is the index set of neurons $z_{1,j}(\boldsymbol{x})$ in the initial layer with $|\mathcal{K}_{1,s}| = k$. The data pass through the network via $z_{1,j}(\boldsymbol{x})$ and constitute the neuron $z_{k,s}$, $k \geq 2$. Considering the basis function of AHH in equation (3), it is obvious that each DAHH neural network can be transformed into an equivalent AHH tree and vice versa. As the AHH model has been proven to approximate any continuous function with arbitrary accuracy in a compact set [15], [21], the DAHH network is also then a universal approximator. $\square$

B. Explicit Domain Partitions and Locally Linear Expressions

DAHH is a flexible PWL neural network consisting of various skip-layer neurons, where the output of each neuron is also PWL over the input domain. In PWL models, the domain is partitioned into different subregions, appearing different linear functions as local responses. Thus, the flexibility of PWL models can generally be measured by their domain partitions, i.e., more partitions are the keys to bringing higher flexibility.

Also pointed out in [22], the number of linear regions of the functions that can be computed by a given model is a measure of the model flexibility. In [22], the complexity of functions computable by deep feedforward NNs is discussed by estimating the number of linear regions with activations of ReLU and Maxout, in which the bounds of such numbers are estimated by layer-wise composition and re-usage of previous layer computations. The existing works are performed for NNs in the fully connected format, where the explicit partitioning hyperplanes are difficult to extract. Thanks to the special network structure of DAHH, the resulting domain partitions regarding each individual neuron can be explored explicitly, which provides a view to reflect network flexibility and better understand the structured-PWL NN of DAHH.

For instance, Figure 7 shows a 2-dimensional example on a neuron output in DAHH, given the normalized domain $[0, 1]^2$.

Figure 7 demonstrates that the given neuron contains the partitions of $x_1 = 0.3$ and $x_2 = 0.5$. Therefore, the corresponding boundaries brought by such domain partitions can be described by the hyperplanes $x_1 = 0.3$, $x_2 = 0.5$, and $x_1 - 0.3 = x_2 - 0.5$, the third of which introduces 2 linear functions separated by $x_1 - 0.3 = x_2 - 0.5$ in the active subregion $\{0.3 \leq x_1 \leq 1, 0.5 \leq x_2 \leq 1\}$, where the neuron is activated with nonzero outputs. We can see that each neuron in DAHH can lead to different domain partitions towards PWL outputs and then gets combined to formulate the more flexible final DAHH, in which the domain partitions regarding each individual neuron can be detected explicitly, i.e., the partitioning hyperplanes can be listed. On such basis, we can obtain the following Theorem 1.

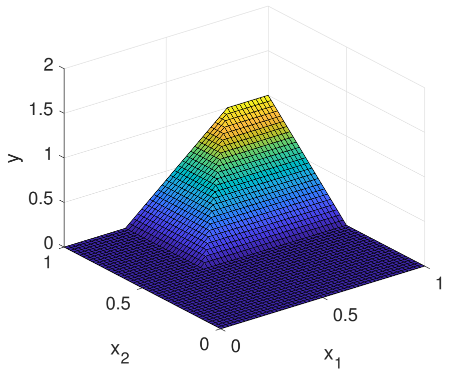


Fig. 7: A simple example of neuron output in DAHH, such that $z(\boldsymbol{x}) = \min\{\max\{x_1 - 0.3, 0\}, \max\{x_2 - 0.5, 0\}\}$.

Theorem 1. *Given a neuron $z_{k,s}$ in DAHH with domain $\Omega \subseteq \mathbb{R}^d$, its output $z_{k,s}(\boldsymbol{x})$ is continuous PWL, where Ω is partitioned into different subregions, each of which appears as a linear function. Then, the boundaries $\mathcal{B}_{k,s}$ between such subregions can be described by the following hyperplanes, i.e.,*

$$\mathcal{B}_{k,s} = \begin{cases} \{x_{v_{1,s}} = \beta_{1,s} | s \in J_{1,s}\} & k = 1 \\ \mathcal{B}^1_{k,s} \cup \mathcal{B}^2_{k,s} & k \geq 2, \end{cases} \tag{16}$$

where $\mathcal{B}^1_{k,s} = \{x_{v_{k,s}} = \beta_{k,s} | s \in J_{k,s}\}$, $\mathcal{B}^2_{k,s} = \{\delta_{k,s_i}(x_{v_{k,s_i}} - \beta_{k,s_i}) = \delta_{k,s_j}(x_{k,s_j} - \beta_{k,s_j}), s_i, s_j \in J_{k,s}, s_i \neq s_j\}$. Analogously, the number of such partitioning hyperplanes in $\mathcal{B}_{k,s}$ can be computed as

$$\mathcal{N}_{k,s} = \begin{cases} 1 & k = 1 \\ k(k+1)/2 & k \geq 2. \end{cases} \tag{17}$$

Proof. For a given neuron $z_{k,s}(\boldsymbol{x})$ in DAHH, from the neuron outputs in (4) and (5), we can obtain these hyperplanes partitioning the domain Ω, and then the boundaries of the active region ($z_{k,s}(\boldsymbol{x}) \geq 0$) can be easily obtained as

$$\delta_{k,s}(x_{v_{k,s}} - \beta_{k,s}) = 0, \forall v_{k,s} \in J_{k,s}, \tag{18}$$

which is equivalent to $x_{v_{k,s}} = \beta_{k,s}$, i.e., $\mathcal{B}^1_{k,s}$. Meanwhile, the active region can be further partitioned by additional hyperplanes to formulate the local PWL output, i.e., if $k \geq 2, \forall s_i, s_j \in J_{k,s}$, the additional partitioning hyperplanes are

$$\delta_{k,s_i}(x_{v_{k,s_i}} - \beta_{k,s_i}) = \delta_{k,s_j}(x_{k,s_j} - \beta_{k,s_j}), s_i \neq s_j. \tag{19}$$

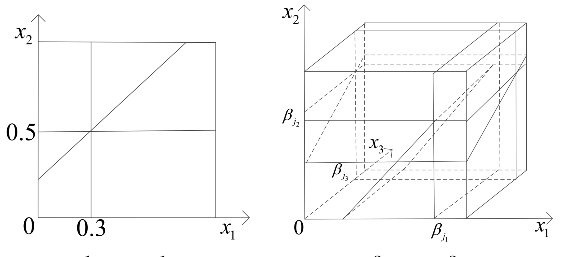
The number of partitioning hyperplanes brought by (18) is computed as $|J_{k,s}| = k$, and such number brought by (19) is $\binom{|J_{k,s}|}{2} = \binom{k}{2} = k(k-1)/2$. In the neuron output $z_{1,s}(\boldsymbol{x})$ of the initial layer, the hyperplane $x_{v_{1,s}} = \beta_{1,s}$ partitions Ω.

In the subsequent layers, the hyperplanes in (19) result in more flexible partitions. Therefore, there are in total $k + \binom{k}{2} = k(k+1)/2$ such hyperplanes, and therein formula (17) holds true in DAHH. $\square$

Consequently, based on the well-known theorem about partitions of d-dimensional space by hyperplanes [23]. The following Corollary 1 is straightforward and capable of depicting the capacity of the given PWL neuron [24].

Corollary 1. *Given a neuron $z_{k,s}$ in DAHH, its PWL output $z_{k,s}(\boldsymbol{x})$ partitions the domain by the hyperplanes in $\mathcal{B}_{k,s}$, and then the maximal number of the resulting linear subregions is bounded by $\mathcal{C}_{k,s} = \sum_{i=0}^{d} \binom{\mathcal{N}_{k,s}}{i}$, where $\mathcal{N}_{k,s}$ is given in (17).*

Fig. 8 gives a visualized example for better demonstration, where Fig. 8(a) illustrates the explanation of Theorem 1 relating to the example in Fig. 7 and Fig. 8(b) is another simple example of a neuron in 3-dimensional space.



(a) $z^1(\boldsymbol{x}), \mathcal{N}^1 = 3$. (b) $z^2(\boldsymbol{x}), \mathcal{N}^2 = 6$.

Fig. 8: Boundaries of the subregions relating to the given neurons, where $z^1(\boldsymbol{x}) = \min\{\max\{x_1 - 0.3, 0\}, \max\{x_2 - 0.5, 0\}\}$ and $z^2(\boldsymbol{x}) = \min_{i=1,2,3}\{\max\{\beta_{j_i} - x_i, 0\}\}$.

Based on Theorem 1, when given an input $\hat{\boldsymbol{x}}$ feeding into a DAHH network $f_{\text{DAHH}}(\boldsymbol{x})$, we can easily compute the active neurons, i.e., the ones with nonzero outputs, such that $z_{\hat{k},\hat{s}}$, $(\hat{k},\hat{s}) \in \hat{T}$ with $\hat{T} = \{(k,s) | z_{k,s}(\hat{\boldsymbol{x}}) > 0\}$. Then, the corresponding linear subregion $\hat{\Omega}_{\hat{\boldsymbol{x}}}$ that $\hat{\boldsymbol{x}}$ locates can be obtained, i.e., Corollary 2.

Corollary 2. *Given an input $\hat{\boldsymbol{x}}$ into a DAHH network $f_{\text{DAHH}}(\boldsymbol{x})$, the subregion $\hat{\Omega}$ sharing the same local linear function with $\hat{\boldsymbol{x}}$ is $\hat{\Omega}_{\hat{\boldsymbol{x}}} = \{x \in \Omega | \delta_t(x_{v_t} - \beta_t) > 0, t \in J_{\hat{k},\hat{s}}\}$.*

Based on Corollary 2, given an arbitrary input $\hat{\boldsymbol{x}}$, the linear function of the PWL network $f_{\text{DAHH}}(\boldsymbol{x})$ in the corresponding local subregion $\hat{\Omega}_{\hat{\boldsymbol{x}}}$ can also be computed explicitly, i.e., Corollary 3.

Corollary 3. *Given an input $\hat{\boldsymbol{x}}$, the network DAHH $f_{\text{DAHH}}(\boldsymbol{x})$ locally appears as a linear function $l_{\hat{\boldsymbol{x}}}(\boldsymbol{x})$ in the subregion $\hat{\Omega}_{\hat{\boldsymbol{x}}}$, such that $\forall \boldsymbol{x} \in \hat{\Omega}_{\hat{\boldsymbol{x}}}$, $f_{\text{DAHH}}(\boldsymbol{x}) = l_{\hat{\boldsymbol{x}}}(\boldsymbol{x}) = \sum_{\hat{k}} \sum_{\hat{s}} w_{\hat{k},\hat{s}} \delta_{t^*_{\hat{k},\hat{s}}} (x_{v_{t^*_{\hat{k},\hat{s}}}} - \beta_{t^*_{\hat{k},\hat{s}}})$, with $t^*_{\hat{k},\hat{s}} = \arg\min_{t_{\hat{k},\hat{s}} \in J_{\hat{k},\hat{s}}} \{\delta_{t_{\hat{k},\hat{s}}} (\hat{x}_{v_{t_{\hat{k},\hat{s}}}} - \beta_{t_{\hat{k},\hat{s}}})\}$.*

Different from other PWL NNs, the proposed DAHH not only can estimate the upper bound concerning the number of linear subregions in Corollary 1, but also can list the hyperplanes which partition the domain and configure the boundaries among these subregions in Theorem 1. By definition, PWL systems own different linear expressions in different local subregions. The existing PWL NNs get benefited from the flexibility of piecewise linearity, but when given an arbitrary input $\hat{\boldsymbol{x}} \in \Omega$, it is generally difficult to give informative analysis towards the subregion in which $\hat{\boldsymbol{x}}$ locates, and the corresponding local linear expression. Owing the special network configuration, Corollary 2 and Corollary 3 present the informative results concerning the local subregion and its linear expression, providing varied theoretical perspectives to better understand the PWL NN of DAHH.

C. ANOVA Decomposition

In MLP, all variables are fully connected and stacked layerwisely. Hence, the relations among variables are difficult to explore and it is prone to bring model redundancy. In DAHH, each neuron can be more interpretable concerning the relation among variables. We can sort the neurons $z_{k,s}$ by its layer index k in DAHH, and then the explicit expression of each layer can be easily obtained, so that the interaction among variables in each neuron can be easily revealed, i.e.,

$$\begin{aligned}
z_{1,s}(\boldsymbol{x}) &= \min_{v_i \in J_{1,s}} \{\max\{\delta_i(x_{v_i} - \beta_i), 0\}\} \\
z_{2,s}(\boldsymbol{x}) &= \min_{v_i, v_j \in J_{2,s}} \{\max\{\delta_i(x_{v_i} - \beta_i), 0\}, \\
&\quad \delta_j(\max\{x_{v_j} - \beta_j), 0\}\}, \\
&\vdots \\
z_{K,s}(\boldsymbol{x}) &= \min_{v_{i_1}, \ldots, v_{i_K} \in J_{K,s}} \{\max\{\delta_{i_1}(x_{v_{i_1}} - \beta_{i_1}), 0\}, \\
&\quad \ldots, \max\{\delta_K(x_{v_{i_K}} - \beta_{i_{n_K}}), 0\}\}
\end{aligned} \tag{20}$$

where K is the depth of the DAHH network, i, $i_1, \ldots, i_K$ are the indices of neurons in its layer, and $J_{k,s} \subseteq \{1, \ldots, d\}$ with $|J_{k,s}| = k$ and $k \in \{1, \ldots, K\}$. Based on (20), similar with the analysis of variance (ANOVA) decomposition in [25], [26], we can also easily get the corresponding ANOVA decomposition of DAHH in (21), while the structure of MLP hinders such analysis.

$$\begin{aligned}
f_{\text{DAHH}}(\boldsymbol{x}) = \;& w_0 + \sum f^1_{r_1}(x_{r_1}) + \sum f^2_{r_1 r_2}(x_{r_1}, x_{r_2}) \\
& + \sum f^3_{r_1 r_2 r_3}(x_{r_1}, x_{r_2}, x_{r_3}) + \ldots,
\end{aligned} \tag{21}$$

where f^l denotes the neurons with $l(l \leq d)$ variables interacting, $r_k \in \{1, \ldots, d\}$ and f^l_J represents the neuron with interacting variable(s) $x_v, v \in J$.

With ANOVA decomposition, the contribution of different variables x_{r_k} can be detected so as the interactions among variables $x_{r_1}, \ldots, x_{r_k}$ with $k \leq d$, such that

$$\begin{aligned}
f^1_{r_1}(x_{r_1}) &= \sum_{J_{k,s} = \{r_1\}} w_{k,s} z_{k,s}(\boldsymbol{x}), \\
f^2_{r_1 r_2}(x_{r_1}, x_{r_2}) &= \sum_{J_{k,s} = \{r_1, r_2\}} w_{k,s} z_{k,s}(\boldsymbol{x}), \\
&\vdots \\
f^k_{r_1 \ldots r_k}(x_{r_1}, \ldots, x_{r_k}) &= \sum_{J_{k,s} = \{r_1, \ldots, r_k\}} w_{k,s} z_{k,s}(\boldsymbol{x}).
\end{aligned} \tag{22}$$

Based on ANOVA decomposition, the importance of different neurons in DAHH can be evaluated by the relevance to the network output [25]. Specifically, with training data, the standard variance of each ANOVA function in (22) can be calculated quantitatively by the ξ value, such that

$$\begin{aligned} \xi^k_{r_1\dots r_k} &= \sqrt{\mathrm{VAR}(f^k_{r_1\dots r_k}(\tilde{\boldsymbol{x}}))}, \\ f^k_{r_1\dots r_k}(\tilde{\boldsymbol{x}}) &= \sum_{\{\tilde{k},\tilde{s}\}\in\tilde{T}} w_{\tilde{k},\tilde{s}} z_{\tilde{k},\tilde{s}}(\tilde{\boldsymbol{x}}), \end{aligned} \tag{23}$$

where the variables interacted in the ANOVA function $f^k_{r_1\dots r_k}$ are denoted as $\tilde{\boldsymbol{x}} = [x_{r_1}, \dots, x_{r_k}]^T, r_j \in \{1, \dots, d\}, j = 1, \dots, k$. $\tilde{T} = \{(k,s)|z_{k,s}(\boldsymbol{x}) = z_{k,s}(\tilde{\boldsymbol{x}})\}$ contains the indices of the correlated neurons, and VAR$(\cdot)$ represents the variance of the corresponding model outputs.

In (23), the larger $\xi_{J^{k,i}}$ is, the more influential $\tilde{\boldsymbol{x}}$ is on the network output. Since formula (23) indicates the contribution of different neurons to the network output, i.e., the direct contribution of different variables or interactions between variables to the network output, the ξ value can be referred as the influence to the corresponding model output, meaning that it reflects the perturbation of corresponding ANOVA function in DAHH network output.

When generating the additive DAHH network ($K = 1$), the ANOVA decomposition can be cast as suggestions for variable selections based on the relative importance revealed by its ξ value. In fact, many datasets and systems are low dimensional in essence or even additive models, even though they come from a higher dimensionality feature space. Taking the 6 dimensional Monk 1/2/3 datasets for instance [27], we use a simple additive DAHH network, and a brief result is shown in Table II, where M is parameter number.

TABLE II: Accuracy on the Monk datasets with single-layer DAHH networks.

	Monk 1	Monk 2	Monk 3
Accuracy	75%	67.4%	97.2%
M	6	2	4
$\tilde{\boldsymbol{x}}$	x_5	x_5	(x_2, x_5)

Table II shows that only variables x_2 and x_5 are left in DAHH, indicating the importance of x_2 and x_5. Monk 3 dataset achieves a very good result with an additive DAHH of only 4 parameters. According to the description of Monk datasets in [27], it tells that $x_1 = x_2$ and $x_5 = 1$ are the critical conditions, which coincides with the property obtained by the ANOVA decomposition analysis from DAHH. In DAHH with more layers ($K \geq 2$), ANOVA decomposition can reveal variable interactions in each neuron and the coupling level K can reflect the complexity among variables in the given dataset, to some extent. Table III gives the results.

TABLE III: Accuracy on the Monk datasets with two-layered DAHH networks.

	Monk 1	Monk 2	Monk 3
Accuracy	100%	100%	97.2%
M	28	262	6
$\tilde{\boldsymbol{x}}$	(x_1, x_2, x_5)	$(x_1 \sim x_6)$	(x_2, x_5)

The accuracy of Monk 1/2 improves significantly with adding another layer, but Monk 3 dataset has a similar result with Table II, revealing the simple interaction among variables in Monk 3. As illustrated in Table III, the coupling level of variables lies in lower dimensionality, meaning that the 6 variables are not necessarily all interacted with each other. It can be noticed that DAHH can achieve state-of-the-art performance in Monk datasets with very simple structures of a few parameters.

Although the validity of ANOVA decomposition analysis cannot always be guaranteed with ground truth and deeper insights are worth further investigations, it can be regarded as an attempt to "look inside" DAHH neural networks.

D. Some Remarks

1) Piecewise Linearity: The activations in DAHH are simply the "max" and "min" operators, which maintain the piecewise linearity of the network.

2) Convexity: The splitting knots β are suggested to be pre-allocated, and then the weights $w_{k,s}$ connecting the output layer are the only parameters to be optimized in DAHH. Thus, the optimization problem maintains convexity when the loss function and the regularization are chosen as convex.

3) Simple optimization: Due to the special network structure, the optimization problem in training DAHH is quite simple, since it does not suffer from gradient vanishing and exploding difficulties in back-propagation .

4) Skip-layer connection: DAHH enables us to flexibly connect the neurons by skipping layers, which does not complicate the optimization but makes DAHH more flexible.

5) Explicit expression and domain partition: The explicit expressions of DAHH can be easily obtained in each layer, where the domain partition and their partitioning hyperplanes can be detected. Besides, the relative importance of variables can be explored through its ANOVA decomposition, which greatly facilitates the interpretation and can be regarded as a simple attempt to explore the "box" of neural networks.

VI. Numerical Experiments

In this section, we evaluate the performance of DAHH with comparisons to related state-of-the-art models. We first evaluate the effectiveness and properties of DAHH. Then, multiple benchmarks are tested for further discussions.

A. Evaluations on Model Effectiveness and Properties

Table IV introduces the selected datasets, where N_{train} and N_{test} are denoted as the numbers of instances in the training and the test datasets. "C" indicates the task for classification and "R" for regression. The dimensionality of input variables and output are expressed as d and l. For the datasets which do not separate the training and testing data, we randomly partition the data for training and testing with ratio 2. Considering the randomness, we report the average of 20 runs. For classification, the criterion is taken as the classifying accuracy, while in regression the criterion is chosen as normalized mean error (NME) in dB, defined as NME $= 10\log_{10}\frac{||\boldsymbol{y}-\hat{\boldsymbol{y}}||_2^2}{||\boldsymbol{y}||_2^2}$, where $\hat{y}$ represents the predicted output, y is the observed output, and $||\cdot||_2$ denotes the l_2 norm.

TABLE IV: Descriptions of the selected datasets.

Dataset	Task	N_{train}	N_{test}	d	l
Strike	R	416	209	6	1
Bodyfat	R	168	84	14	1
Spacega	R	2071	1036	6	1
Abalone	R	2784	1393	8	1
California Housing	R	14448	6192	8	1
Boston Housing	R	337	169	13	1
Satimage	C	4435	2000	36	6
Letter	C	13333	6667	16	26
Shuttle	C	43500	14500	9	7

In DAHH, the number of neurons in each layer is set default as $n_k = 1000$. The knots β of initial layer are uniformly pre-allocated as quantiles based on the distribution of training data with 7 splitting knots in each dimension and can be tuned with an increment of 2^{d-1} in this paper. For the l_1 norm regularization, we set the coefficient $\lambda = 10^{-10}$ with a change of timing 10, and it is decided by 5-fold cross validation. Then, only the number of layers K needs to be determined. Normally, when the number of training data is significantly larger than data dimension, we need a more complicated model to describe the system. Thus, we provide a heuristic suggestion for determining the depth of DAHH network. For $N_{\text{train}}/d < 100$, we set the number of layers as $K = 2$, and $K \in \{6, 11, d\}$ for $N_{\text{train}}/d > 100$, which can also be adaptively adjusted. The DAHH model is implemented in the platform of Tensorflow 1.12.0 with a 3.20GHz Intel(R) Core(TM) i7-8700 CPU of 16.0 GB, where the AdamOptizer is selected with default batch size of 500 and 50 for $N_{\text{train}} > 2000$ and $N_{\text{train}} \leq 2000$ respectively.

1) Selection of splitting knots β_{ij}: In Section IV-B, we point out that the splitting knots β_{ij} can be optimized together with the weights $w_{k,s}$ on the sacrifice that the optimization problem (8) becomes more complicated, shown in (12). In this part, we evaluate the methods of determining the splitting knots β_{ij}, i.e., the method in (11) and the method in (13), whose accuracy are denoted by DAHH and DAHH (with β_{ij}), respectively. Table V lists the average predicted accuracy and its standard variance.

TABLE V: Testing accuracy on the methods of determining the splitting knots β.

Dataset	DAHH	DAHH (with β_{ij})
Strike	**-2.0**±0.1	-1.8±0.1
Bodyfat	**-24.7**±2.3	-23.5±1.1
Spacega	**-15.2**±0.3	**-15.2**±0.2
Abalone	-13.47±0.2	**-13.6**±0.2
California Housing	**-13.5**±0.1	-12.9±0.2
Boston Housing	**-18.3**±1.0	-18.2±0.8
Satimage	**90.8%**±0.3	90.8%±0.2
Letter	**96.6%**±0.1	96.5%±0.1
Shuttle	**99.9%**±0.0	99.5%±0.2

Table V shows that incorporating splitting knots into the optimization problem (8) leads to inferior performance than the method in (11). This can be due to the fact that the stochastic gradient descent algorithm performs poorly in the induced highly non-convex optimization problem (12). The results in Table V verify the effectiveness of the method by pre-allocating the splitting knots as the quantiles in (11). Thus, in this paper, we suggest the method in (11) to select the splitting knots in DAHH, and then apply the stochastic gradient descent algorithm to optimize the weights $w_{k,s}$ in (8), which presents a simple but effective training of DAHH.

2) Model Flexibility: We next present comparison experiments with some related learning methods, including the progressive learning networks (PLN), which incorporates randomness of neurons and uses ReLU activation functions to obtain the network based on forwardwise method with l_1 norm regularization for more compact structures [28], [29]. Another popular forwardwise neural network, i.e., extreme learning machine (ELM), is then incorporated [30]. MLPs with ReLU activations are also presented. The results of PLN and ELM network are fine tuned for each dataset and cited from [28]. For MLP, the single-layered or two-layered structure is chosen for the given simple examples and runs on Tensorflow 1.12.0 platform, where the AdamOptimizer is selected with batch size of 50 and learning rate of 0.01. The number of hidden neurons is set as $\{10, 20, 50, 100, 200, 500\}$ and the best result is reported. For simple examples, we adopt the full skip-layer connection in DAHH with $K_{\text{out}} = 1$. Table VI lists the results over 20 runs, where "Calif H" and "Bost H" are abbreviated for the datasets of California Housing and Boston Housing.

TABLE VI: Testing accuracy on the datasets in Table IV.

Dataset	DAHH	PLN	ELM	MLP
Strike	**-2.0**±0.1	-1.7±0.7	-1.6±0.5	-1.90±0.5
Bodyfat	**-24.7**±2.3	-14.1±0.7	-13.4± 0.6	-24.3±2.1
Spacega	**-15.2**±0.3	-11.6±0.4	-8.5±0.2	-14.8±0.1
Abalone	-13.5±0.2	**-13.8**±0.2	**-13.8**±0.2	**-13.8**±0.3
Calif H	**-13.5**±0.1	-10.6±0.4	-10.8±0.2	-12.9±0.2
Bost H	**-18.3**±1.0	-13.4±0.7	-13.9±0.7	-16.9±1.1
Satimage	**90.8**±0.3	89.9±0.5	84.6±0.5	89.6±0.2
Letter	**96.6**±0.1	95.7±0.2	95.7±0.2	96.2±0.1
Shuttle	**99.9**±0.0	99.8±0.1	99.6±0.1	99.8±0.0

Table VI shows that DAHH outperforms the selected models for most of the given datasets, which shows the flexibility of the proposed DAHH neural network. Since the existing EHH is currently restricted to the LS residual loss, we only present comparisons on regression tasks, where EHH is set to be the same structure budgets of DAHH. The results of EHH are inferior than that of DAHH and MLP, thus we omit the concrete statistical performance of EHH in Table VI for simplification. Table VI indicates that the forward growth in DAHH and the proposed activity ratio r_{act} strategy are effective and works well in enhancing accuracy, and it also reflects that the stochastic gradient descent algorithm can approach a good solution in optimizing DAHH by combining the method of selecting the splitting knots in (11). The programming platforms and algorithms are different, thus the running time is omitted. Since PLN and ELM requires many random neurons with obviously much more number of connecting weights in model representation, the comparison of parameter numbers is omitted. We only compare the number of parameters with the MLPs and the results are shown in Table IX.

Table IX shows that higher accuracy is achieved with less

TABLE VII: The number of modeling parameters.

Dataset	DAHH	MLP
Strike	387	350
Bodyfat	39	150
Spacega	210	350
Abalone	33	90
California Housing	4066	1800
Boston Housing	450	1400
Satimage	1447	21000
Letter	28800	42000
Shuttle	569	320

parameters in DAHH for most of the given datasets. Although the depth of DAHH is set significantly higher than MLPs, the number of parameters remains less than MLPs for most of the datasets. Due to the special network structure, the definitions of network depth differ in DAHH and MLP, since the neurons are connected in a sparse way and the $\min\{\cdot\}$ activation requires no connecting weights in DAHH. Thus, even distinctively deeper DAHH can have less number of parameters than MLP. In fact, for the datasets in Table IX, the MLP with more than two hidden layers performs inferior than the results in Table IX, since the optimization performance can be hindered in deep MLP. In DAHH, the optimization problem maintains convex and the performance of algorithm itself is not impaired by deepening the DAHH. Thus, rather than the network depths, we compare the number of parameters in Table IX.

B. Empirical Study on ANOVA Decomposition

Owing the special network structure, ANOVA decomposition can be conducted to the network output of DAHH, where the impact of all neurons (ANOVA functions) can be detected quantitatively concerning its influence to the corresponding variance of network output, i.e., the ξ value in (23).

We then conduct empirical studies of ANOVA decomposition on the datasets in Table IV. Taking the Bodyfat and California Housing datasets for illustrations, similar with Section V-C, we first shed light on the univariate initial layer ($K = 1$), providing a suggestion for variable selections. Table VIII shows the ξ value based on ANOVA decomposition to reflect the influence of neurons regarding the perturbation to DAHH outputs.

TABLE VIII: The ξ value based on ANOVA decomposition analysis for additive DAHH.

California Housing					
Sort	Variable	ξ	Sort	Variable	ξ
1	x_1	15.51	5	x_5	5.10
2	x_2	9.45	6	x_7	3.02
3	x_8	7.36	7	x_4	1.25
4	x_6	6.64	8	x_3	0.86
Bodyfat					
Sort	Variable	ξ	Sort	Variable	ξ
1	x_7	0.49	8	x_2	0.16
2	x_1	0.38	9	x_6	0.16
3	x_3	0.23	10	x_8	0.16
4	x_{10}	0.19	11	x_5	0.14
5	x_9	0.18	12	x_{13}	0.08
6	x_{11}	0.18	13	x_4	0.07
7	x_{14}	0.18	14	x_{12}	0.04

Table VIII reveals the influence of individual variables to the network output. In California Housing dataset, x_3 (housing median age) has the least influence to the output, while x_1 (longitude), x_2 (latitude) and x_8 (median income) have shown to be more influential. In Bodyfat dataset, x_7 (abdomen circumference), x_1 (desity) and x_3 (weight) are significant to the output. Meanwhile, x_{13} (forearm), x_4 (height) and x_{12} (biceps) have shown to be trivial. Moreover, the accuracy is unaffected and even more accurate when deleting x_{13}, x_4 and x_{12} in Bodyfat, which provides an evidence for the solidity of the aforementioned ANOVA decomposition analysis.

Next, we perform analysis on the DAHH with deeper layers, meaning that the influence of the neurons containing multiple variables can be detected, where ANOVA decomposition can therein provide an alternative to present the interacting relations among variables when necessary. In the California Housing dataset, the ξ value of the 2nd layer involving x_1 (longitude) and x_2 (latitude) are significantly higher, i.e., $z_{2,s}(\tilde{\boldsymbol{x}})$ with $\tilde{\boldsymbol{x}} = [x_1, x_2]^T$, meaning that the location is quite important for the housing price. After the pruning, DAHH has $K = 6$ layers, meaning that most of the variables and their interactions are influential to the network output. This indicates that the factors resulting in the final housing price are a bit complicated. Differently, for Bodyfat dataset, an additive model of DAHH is obtained with the highest accuracy after the tuning, indicating a simpler intrinsic relation among the variables.

Although higher accuracy can generally be attained by increasing layers (model flexibility), some datasets which are intrinsically simple in variable interactions can appreciate simpler model structures. For Bodyfat dataset, deeper structure $K \geq 2$ even brings inferior accuracy. For many datasets, not all variables are interacted with each other as MLPs which connect all variables and stack them layerwisely. The depth K of DAHH network can reveal this characteristic, which helps explore the coupling level of variables and bring a more concise structures. Table IX gives the depth of DAHH in Table VI with comparing data dimensionality d.

TABLE IX: Depth K of DAHH for the datasets in Table IV.

Dataset	n_{in}	K
Strike	6	2
Bodyfat	14	2
Spacega	6	2
Abalone	8	2
California Housing	8	6
Boston Housing	13	6
Satimage	36	11
Letter	16	11
Shuttle	9	2

We can see that the depth K of DAHH mainly lies in significantly lower dimensionality for most of the datasets, meaning that all variables are not necessarily interacted. The depth K can be regarded as a view to interpret complexity of the dataset concerning variable coupling levels. An additive PWL system is flexible enough to describe Bodyfat dataset. For Strike, Spacega and Abalone, the models consisting of two-variable coupled components are enough to reveal the intrinsic relations of the datasets, where the number of training data is quite smaller compared to the dimensionality. It is worthy to notice that DAHH can achieve a significantly high accuracy with merely 2 variables coupled in Shuttle dataset which contains over 14000 training data, revealing the simple intrinsic relations among variables. Next, we simply adopt $K = 8$ for both datasets of Bodyfat and California Housing to check the relative contribution of different layers $\#k, k = 1, \ldots, K$ in DAHH concerning the ξ value based on ANOVA decomposition.

TABLE X: The ξ value based on ANOVA decomposition of multi-layered DAHH.

	Layer $\#k$							
	1	2	3	4	5	6	7	8
Bodyfat	2.19	8.52	2.45	0.35	0	0	0	0
Cali H	18.3	39.1	31.1	13.7	5.6	0.2	0	0

Table X shows that ANOVA decomposition can reflect the relative contribution of different layers when given a DAHH, i.e., the relative contribution of different levels of variable coupling. In Table X, even set with a higher depth, DAHH still prunes these neurons of high levels of variable coupling, indicating a simpler modeling description for Bodyfat, while a relatively deeper DAHH is preferred for California Housing.

The above analysis of ANOVA decomposition can give a suggestion to reveal the relative importance of different individual variables, and also the relative importance of different layers (different variable interactions). These analyzing results facilitating the interpretation of DAHH and its structures. Therefore, besides being a prediction model, DAHH itself can also perform variable analysis and selections. Although the aforementioned analysis are not necessarily the ground truth always with guarantee, it can be regarded as an attempt to interpret the "box" of DAHH neural networks.

C. Performance Evaluation on Benchmarks

In this subsection, we test several popular benchmarks of regression and classification in Table XI. The selected benchmarks have different characteristics and suit different learning methods. Thus, the selected state-of-the-art methods can differ slightly for each dataset in the comparisons. Since the methods are varied and implemented in different platforms, we omit the running time comparisons in this paper.

For complicated datasets, the neurons from shallow layers can pertain less influence to the output. To make the network more concise, we adopt the partial skip-layer connections with $K_{\text{out}} = K - 1$, meaning that only the last two layers are connected to the network output. In the selected benchmarks, $K_{\text{out}} = K - 1$ and $K_{\text{out}} = 1$ achieve comparable accuracy, but $K_{\text{out}} = K - 1$ has less parameters and simpler structure, which proves the effectiveness of the partial-skip layer connection.

1) Regression of Year Prediction MSD: Year Prediction MSD dataset is a popular benchmark for large scale multivariate regression [27]. This dataset contains the songs between 1922 and 2011, and aims to predict the year in which a song was released based on the audio features. Since PLN and MLE do not pertain good performance in this dataset, we compare DAHH with MLP and the variational inference for neural network (VI), which is an improved stochastic variational network method with very good performance in the Year Prediction MSD dataset [31], [32]. The method of scalable predictive uncertainty estimation using deep ensembles is proposed in [33], which is another state-of-the-art method for this dataset. For MLP, we choose the candidate structure as the single hidden layer with neurons numbers of $\{500, 800, 1000, 1200, 1500, 2000\}$ and two hidden layers of $\{[500, 500], [800, 800], [500, 1000], [1000, 500], [1000, 1000]\}$, where the l_2 regularization and dropout are introduced to fine tune the model. We then report the best performance of MLP. In DAHH, we set $n_k = 20000$ neurons as the budgets in each layer, and the backward pruning is also used. The fine tuned results of VI and deep ensembles methods are cited from [32], [33]. Table XI illustrates the comparisons.

TABLE XI: Descriptions of the selected benchmarks.

Data set	Task	N_{train}	N_{test}	d	l
Year Prediction MSD	R	463715	51630	90	1
MNIST	C	60000	10000	784	10
NORB	C	24300	24300	2048	5

TABLE XII: Testing NME on Year Prediction MSD dataset.

	DAHH (K)	VI	Deep Ensembles	MLP
NME	**-47.04** (20)	-46.89	**-47.04**	-46.98

Table XII shows that DAHH still pertains advantages in accuracy for MSD dataset and obtains the same best result with the state-of-the-art method of Deep Ensembles. The depth $K = 20$ indicates that at most 20 variables interacted in the neurons of DAHH are capable of describing the model and achieving a fairly good performance, while all variables interact with each other in each neuron of MLP.

2) Image Classification of MNIST and NORB: MNIST and NORB are the benchmarks for image classification [20], [34]. MNIST dataset contains the handwritten digits from 0 to 9 by a matrix with 28×28 gray scale pixels. NORB dataset contains images of 50 different 3-D toy objects belonging to 5 distinct categories, where the images have $2 \times 32 \times 32$ pixels. Except MLP, we also compare with the broad learning (BL) network, which designs an incremental learning neural network in a forward wise way and has shown obvious better performance than PLN in both MNIST and NORB [35]. Rather than ELM, we employ multiple-layered ELM (MLELM) for better performance, and the MLP is also considered [36]. Table XIII shows the results, where the BL, MLELM and MLP are fine tuned and cited from [35], [36]. For both MNIST and NORB, DAHH is set with $K = 6$ also with the l_1 norm regularization to delete redundant neurons in the network.

Table XIII shows that DAHH achieves the highest accuracy in NORB dataset and MLELM shows to outperform DAHH slightly in MNIST, which verifies the effectiveness of DAHH in dealing with image datasets. It worths to be noticed that the depth $K = 6$ of DAHH are significantly shallower than

TABLE XIII: Testing classification accuracy on MNIST and NORB datasets.

	DAHH (K)	BL	MLELM	MLP
MNIST	98.83% (6)	98.74%	**99.04%**	97.39%
NORB	**90.54%** (6)	89.27%	88.91%	84.20%

the dimensionality of the original feature space, i.e., $d = 784$ and $d = 2084$, meaning that at most 6 variables interacted in the neurons of DAHH are capable of achieving competitive performance for the datasets containing $d = 784$ and $d = 2084$ variables. This characteristic tells that the interaction among variables is not complicated, which facilitates simplicity in model representation and interpretation of the coupling levels among variables.

BL and MLELM achieve comparable accuracy with DAHH networks, but the numbers of network weights in BL and MLELM are tremendous, since there are many random fully connected neurons requiring quite a lot memory in storage. The whole network of DAHH only needs the output connecting weights. We then present the number of weights in the networks of BL, MLELM and DAHH.

The deep structures of BL are set as 100-11000 and 1000-9000 for MNIST and NORB respectively. In [35], it shows that BL achieves comparable results with MLELM, but has much less number of weights than MLELM. Thus, we only need to compare the model size of BL with that of DAHH.The connecting weights of BL and MLELM require huge storage for model representations. After the pruning, only 260k weights are contained in DAHH for MNIST and 140k weights for NORB. Table XIV demonstrates the comparison on the numbers of weights contained in the models, where the numerical operator represents the concrete calculation on the number of weights in BL.

TABLE XIV: Comparisons on the number of weights.

	BL	DAHH
MNIST	$784 \cdot 100 + 100 \cdot 11000 + 11100 \cdot 10$	$\underline{260k}$
NORB	$784 \cdot 9000 + 9000 \cdot 11000 + 20000 \cdot 10$	$\underline{140k}$

Table XIV demonstrates that the sparse connections in DAHH network contribute to less memory storage in model representation but maintains competitive performance, which verifies the conciseness and the flexibility of the proposed DAHH neural network.

D. Performance Analysis and Further Discussions

The aforementioned experiments present comprehensive evaluation of DAHH on different benchmark datasets for various tasks. It is worth noticing that DAHH can achieve competitive performance even for the image datasets of MNIST and NORB, where no extra feature extractors are used, such as the convolutional neural network (CNN) [20].

Generally, in order to achieve fairly good performance in image classification, CNNs should be taken to extract features and then connect to other classifiers, so as to capture the spatial information and local patterns. Therefore, the models using CNNs as feature extractors commonly appear significantly superior results than the ones without using CNNs. However, in such cases, the resulting high accuracy for the tested image datasets mainly owns to the contribution of CNNs, and the evaluation on the effectiveness of the connected classifier is weakened. Therefore, we directly apply our proposed DAHH to the image datasets of MNIST and NORB in Table XIII without using any feature extractor nor data augmentation, where DAHH still shows competitive performance, which can better verify the flexibility and effectiveness of the proposed model. To better state this point, we test on MNIST with CNN and one of the state-of-the-art deep models, i.e., LeNet-5, which is built on the delicate integration of multiple CNNs and MLPs. In addition, we also test a three-layered CNN, whose architectures is set as $32 \times 3 \times 3$, $64 \times 3 \times 3$ and $128 \times 3 \times 3$, each of which is followed by a max pooling of 2×2. Then, we simply replace the initial layer in DAHH with such three-layered CNN as feature extractors, followed by a simple DAHH configured as $[1000, 1000]$, i.e., C-DAHH. Similarly, we also try to different structures of MLP with neurons of $[100]$, $[500]$ and $[1000]$ in each layer, concatenated after the CNN by reporting the best performance, i.e., C-MLP. Table XV shows the comparisons, where the result of LeNet-5 is fine tuned and cited from [37].

TABLE XV: Comparisons on the testing accuracy of MNIST with and without CNNs.

MLP	DAHH	CNN	C-MLP	C-DAHH	LeNet-5
97.39%	98.83%	99.03%	99.16%	99.18%	**99.40%**

We can see that the incorporation of CNNs significantly improves the performance of both DAHH and MLP. The performance of C-DAHH shows slightly better prediction than C-MLP, though the number of weights in C-DAHH is much less than C-MLP. C-DAHH is slightly inferior than that of the state-of-the-art deep NN like LeNet, where LeNet-5 also incorporates layers of batch normalization to enhance the performance and has significantly more parameters, while C-DAHH and C-MLP do not posses such strategy and have relatively simpler architectures. Thus, C-DAHH can still be regarded to remain a competitive result with state-of-the-art NN of LeNet-5.

CNNs have natural advantages in image classfication, and thus we mainly consider them as feature extractors in such cases. In this paper, we shed core light on the flexibility of the proposed DAHH on general tasks.

Besides, different trails are given to utilize the explicit structure information of DAHH, i.e., theoretical analysis in Section V and numerical experiments in Section VI-B. For some specific applications, interpretation analysis can be appreciated to detect more properties, such as the importance of different model components towards prediction outputs. However, the current analysis has limitations for more complex scenarios, where variables are differently or inexplicitly coupled, and its validity cannot always be guaranteed with prior knowledge regarding the ground truth, yet it can still be regarded as an attempt to look inside the box of DAHH network.

Based on the above discussions, image-specified cases remain as challenges. Thus, more sophisticated integration of CNN and DAHH is worth rigorous investigations, where the skip-layer connection and the nested pooling operator of $\min\{\cdot\}$ and $\max\{\cdot\}$ can be promising to construct novel network integrations. Meanwhile, how to better utilize and interpret of the special structure of DAHH is also worthy of further trials, and its explicit geometrical explanation may facilitate theoretical insight towards generic deep NNs with PWL activations.

VII. Conclusion

Starting from the tree structure of AHH, DAHH is proposed as a specialized PWL neural network together with effective training method. Through the forward growth, the network structure is efficiently and adaptively determined, and the output connection can be flexibly chosen with skip-layer connections. Under such structure framework, DAHH can be trained by back-propagation algorithm, which is capable of tackling large scale problems. Instead of a black box, the explicit expression of each neuron can be easily obtained and facilitate the application of ANOVA decomposition, revealing the coupling level and interactions among variables. Theoretical analysis are also discussed regarding the approximation, explicit domain partition and locally linear expressions. Numerical experiments verify the effectiveness and flexibility of the proposed DAHH.

References

[1] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural networks*, vol. 2, no. 3, pp. 183–192, 1989.

[2] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[7] L. Breiman, "Hinging hyperplanes for regression, classification, and function approximation," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 999–1013, 1993.

[8] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[10] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.

[11] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[12] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep learning with s-shaped rectified linear activation units," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[13] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[14] S. Wang and X. Sun, "Generalization of hinging hyperplanes," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4425–4431, 2005.

[15] J. Xu, X. Huang, and S. Wang, "Adaptive hinging hyperplanes and its applications in dynamic system identification," *Automatica*, vol. 45, no. 10, pp. 2325–2332, 2009.

[16] Q. Tao, J. Xu, J. A. Suykens, and S. Wang, "Fast adaptive hinging hyperplanes," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 1482–1487.

[17] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[18] J. Xu, Q. Tao, Z. Li, X. Xi, J. A. Suykens, and S. Wang, "Efficient hinging hyperplanes neural network and its application in nonlinear system identification," 2020.

[19] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[20] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] P. Julian, 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*, vol. 46, no. 4, pp. 463–480, 1999.

[22] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in neural information processing systems*, 2014, pp. 2924–2932.

[23] R. Winder, "Partitions of n-space by hyperplanes," *SIAM Journal on Applied Mathematics*, vol. 14, no. 4, pp. 811–818, 1966.

[24] P. Baldi and R. Vershynin, "The capacity of feedforward neural networks," *Neural networks*, vol. 116, pp. 288–311, 2019.

[25] J. H. Friedman *et al.*, "Multivariate adaptive regression splines," *The annals of statistics*, vol. 19, no. 1, pp. 1–67, 1991.

[26] I. Lind and L. Ljung, "Regressor and structure selection in narx models using a structured anova approach," *Automatica*, vol. 44, no. 2, pp. 383–395, 2008.

[27] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[28] S. Chatterjee, A. M. Javid, M. Sadeghi, P. P. Mitra, and M. Skoglund, "Progressive learning for systematic design of large neural networks," *arXiv preprint arXiv:1710.08177*, 2017.

[29] X. Liang, A. M. Javid, M. Skoglund, and S. Chatterjee, "Distributed large neural network with centralized equivalence," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 2976–2980.

[30] H. Guang-Bin, C. Lei, and S. Chee-Kheong, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans Neural Netw*, vol. 17, no. 4, pp. 879–892, 2006.

[31] A. Graves, "Practical variational inference for neural networks," in *Advances in neural information processing systems*, 2011, pp. 2348–2356.

[32] C. Louizos and M. Welling, "Deep bayesian neural nets as deep matrix gaussian processes," 2016.

[33] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems*, 2017, pp. 6402–6413.

[34] Y. LeCun, F. J. Huang, L. Bottou *et al.*, "Learning methods for generic object recognition with invariance to pose and lighting," in *CVPR (2)*. Citeseer, 2004, pp. 97–104.

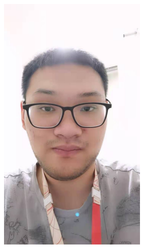
[35] C. P. Chen and Z. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 1, pp. 10–24, 2017.

[36] C. Erik, G. Huang, L. Liyanaarachchi *et al.*, "Extreme learning machines [trends & controversies]," *IEEE Intelligent Systems*, vol. 28, no. 6, pp. 30–59, 2013.

[37] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "Addernet: Do we really need multiplications in deep learning?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1468–1477.

Qinghua Tao received the B.S. degree from Central South University, China, in 2014, and the Ph.D. degree from Tsinghua University, China, in 2020. She is currently a Post-Doctoral Researcher with ESAT-STADIUS, KU Leuven, Belgium. Her research interests include machine learning, dynamic systems and optimization, especially for the analysis and applications of piecewise linear neural networks.

Jun Xu received the B.S. degree in control science and engineering from the Harbin Institute of Technology, China, in 2005, and the Ph.D. degree in control science and engineering from Tsinghua University, China, in 2010. She is currently an Associate Professor with the School of Mechanical Engineering and Automation, Harbin Institute of Technology, Shenzhen, China. Her research interests include piecewise linear functions and their applications in machine learning, as well as nonlinear system identification and control.



Zhen Li received the B.S. degree in automation from Changzhou University, Changzhou, China, in 2017, and the M.S. degree in control science and engineering from Harbin Institute of Technology, Shenzhen, China. His current research interests include, ensemble and boosting machine learning methods, piecewise linear neural networks and their applications.

Na Xie received the B.S. degree from Zhejiang University, China, in 2006, and the Ph.D. degree from Tsinghua University, China, in 2011. From 2009 to 2010, she was a visiting scholar in Cambridge University, the UK. She is currently an Associate Professor with School of Management Science and Engineering, Central University of Finance and Economics, China. Her research interests include the investment and financing of infrastructures, intelligent transportation investing policy, and the applications of machine learning and operational research methods, especially from economy aspects.

Shuning Wang received the B.S. degree in electrical engineering from Hunan University, Changsha, China, in 1982, and the M.S. and Ph.D. degrees in system engineering from Huazhong University of Science and Technology, Wuhan, China, in 1984 and 1988, respectively. He was an Associate Professor from 1992 to 1993 and a Full Professor from 1994 to 1995 with the Institute of Systems Engineering, Huazhong University of Science and Technology, Hubei, China. Since 1996, he has been a Full Professor with the Department of Automation, Tsinghua University, Beijing, China. He was a Visiting Scholar with the College of Engineering, University of California at Riverside, Riverside, in 1994, and a Visiting Fellow with the Department of Electrical Engineering, Yale University, New Haven, CT, from 2001 to 2002. His current research interests include developing practical methods for nonlinear system identification, control and optimization via piecewise-linear approximation.

Xiaoli Li received the B.E. and M.E. degrees from Dalian University of Technology in 1994 and 1997, respectively, and a Ph.D. degree from Northeastern University, China, in 2000. From 2000 to 2003, he was a postdoctoral research fellow at Tsinghua University, China, and at the University Libre de Bruxelles, Belgium. Now, he is a professor at Beijing University of Technology. His research interests include intelligent control, multiple model control, adaptive control, and robust control.

Johan A.K. Suykens (SM'05–F'15) was born in Willebroek, Belgium, in May 18, 1966. He received the M.S. degree in electro-mechanical engineering and the Ph.D. degree in applied sciences from Katholieke Universiteit Leuven, Leuven, Belgium, in 1989 and 1995, respectively. In 1996, he was a Visiting Post-Doctoral Researcher with the University of California at Berkeley, Berkeley, CA, USA. He has been a Post-Doctoral Researcher with the Fund for Sci- entific Research FWO Flanders, Belgium. He is currently a Professor (Hoogleraar) with KU Leuven, Leuven. He is currently serving as the Program Director of Master AI at KU Leuven. He is the 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), a coauthor of the book, Cellular Neural Networks, Multi-Scroll Chaos and Synchronization (World Scientific), and an Editor of the books, Nonlinear Modeling: Advanced Black-Box Techniques (Kluwer Academic Publishers) and Advances in Learning Theory: Methods, Models and Applications (IOS Press).

Dr. Suykens has been an elevated IEEE Fellow 2015 for developing least squares support vector machines. He has been awarded an ERC Advanced Grant 2011 and 2017. He was a recipient of the International Neural Networks Society INNS 2000 Young Investigator Award for significant contributions in the field of neural networks. He received the IEEE Signal Processing Society 1999 Best Paper (Senior) Award and several best paper awards at international conferences. In 1998, he organized an International Workshop on Nonlinear Modeling with Time-Series Prediction Competition. He has served as Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS from 1997 to 1999 and 2004 to 2007 and the IEEE TRANSACTIONS ON NEURAL NETWORKS from 1998 to 2009. He has served as the Director and an Organizer of the NATO Advanced Study Institute on Learning Theory and Practice (Leuven, in 2002), a Program Co-Chair for the 2004 International Joint Conference on Neural Networks and the 2005 International Symposium on Nonlinear Theory and its Applications, an Organizer of the 2007 International Symposium on Synchronization in Complex Networks, a Co-Organizer of the NIPS 2010 workshop on Tensors, Kernels and Machine Learning, and the chair of the International Workshop on Advances in Regularization, Optimization, Kernel Methods, and Support Vector Machines: Theory and Applications (ROKS) 2013.