Learning Weights and Thresholds of Threshold Logic Networks

Kevin Kai-Chun Chang and Jason Hao-Yu Wu
Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
{b07901056, b07901067}@ntu.edu.tw

ABSTRACT

Threshold logic network is more viable nowadays due to its compactness and strong bind to neural network applications. However, the problem of weights and thresholds determination still remains open. In this work, we introduce machine learning and propose two approaches, the function-based approach and the network-based approach, to solve the problem. Experimental results show that our method achieves near 80% accuracy in the function-based approach and 70% to 90% accuracy in the network-based approach.

KEYWORDS

Threshold logic network, neural network, machine learning

1 INTRODUCTION

The development of threshold logic started in 1960s. It shows superior compactness, as it costs fewer logic gates than the conventional Boolean logic network does; thus, it makes threshold logic gates (TLG) proper units to construct a logic network. Besides, threshold logic networks (TLN) exhibits strong bind to neural network applications, which makes it become a competitive substitution for the traditional CMOS technologies [1, 2].

However, previous works mostly focus on the threshold logic synthesis problem, where the weights and thresholds are often given in advance. Besides, machine-learning-based approaches are seldom applied on threshold-logic-related problem, while they have been widely adopted in a variety of EDA areas.

Therefore, we attempt to solve the problem of determining weights and thresholds of a given TLN topology, given a function of primary inputs and primary outputs of the TLN. Section 2 details the problem formulation. Besides, as the learnability of the problem has been proved (see Section 2), we propose two ML-based approaches to solve the problem, the function-based approach and the network-based approach.

The function-based approach considers a function each time. The given TLN is first converted to a corresponding neural network, and training is performed on the neural network to fit the given function. Eventually, the weights and thresholds could be extracted from the neural network directly. As for the network-based approach, multiple functions are given and divided into training and testing sets. Baesd on the training set, a deep neural network (DNN) is generated. Then, the DNN model is able to to predict weights and thresholds for the testing functions.

The main contribution of this work is summarized as follows:

- We formulate the **TLN weights and thresholds learning problem**, and we introduce machine-learning-based approaches to solve the problem.
- We analyze the relation between TLNs and neural networks and propose a conversion method. The weights and thresholds of a TLN could be extracted from the corresponding neural network directly.
- We propose a DNN-based training framework and generate a DNN model to predict the thresholds and weights for unseen testing functions.

In the function-based approach, experimental result shows it achieves almost 80% prediction accuracy. The experiment results of the network-based approach shows that our work achieves 90% prediction accuracy on a fully-connected TLN case. For the non-fully-connected case, it also achieves 70% prediction accuracy.

The remaining of this paper is organized as follows. Section 2 gives a detailed definition of TLN, analyzes its PAC-learnability, and formulate the TLN weights and thresholds learning problem. Section 3 detailes the proposed function-based approach. Section 4 describes the proposed network-based approach. Section 5 illustrates the experiment results and the findings. Finally, Section 6 concludes the paper and shows the potential of our work.

2 PRELIMINARIES

2.1 Threshold Logic Network

A threshold logic function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ over Boolean variables $(x_1, x_2, ..., x_n)$ is defined as Equation 1, where $(w_1, w_2, ..., w_n) \in \mathbb{R}^n$ are the weights, and $T \in \mathbb{R}$ is the threshold.

$$f(x_1, x_2, ..., x_n) = \begin{cases} 1, \text{if } \Sigma_{i=1}^{n}(w_i \cdot x_i) \geq T \\ 0, \text{otherwise} \end{cases} \quad (1)$$

A threshold logic gate (TLG) is a logic unit that realizes a threshold logic function. A TLG has n inputs $(x_1, x_2, ..., x_n)$ and 1 output y, where $x_i's$ and y are all Boolean variables. It also holds n weights $(w_1, w_2, ..., w_n)$ and 1 threshold T, where $w_i's$ and T are real values. The value of output y is determined as Equation 1.

One or more TLGs comprise a threshold logic network (TLN) $N = (V, E)$, where V are the vertices and $E \subseteq V \times V$ are the edges. V can be further divided into three disjoint subsets $\{V_I, V_O, V_G\}$. Vertices in V_I are the primary inputs (PIs) which have no fan-ins. Vertices in V_O are the primary outputs (POs) which have no fan-outs. The remaining vertices that belongs to V_G are normal gates. Each vertex in $V_O \cup V_G$ corresponds to a TLG, while each vertex in V_I is just a Boolean value. Besides, N is a direct acyclic graph (DAG) [1].

2.2 PAC-Learnability Analysis

To analyze if a problem is solvable through learning process, we often rely on the probably approximately correct (PAC) learning model [3]:

A concept class C is PAC-learnable over the instance space X if $\exists$ an algorithm L s.t. $\forall$ concept $c \in C$, $\forall$ fixed probability distribution D over X, and $\forall 0 \leq \epsilon, \delta \leq 1/2$, if L is given access to the procedure $EX(c, D)$ which returns an example $< x, c(x) >$ each time, then with a probability at least $1 - \delta$, L outputs a hypothesis concept $h \in C$ s.t. $error(h) \leq \epsilon$.

Then, the PAC-learnability can be connected with the Vapnik-Chervonenkis-dimension (VC-dimension) through the following theorem [3–5], as the analyses on the VC-dimensions are well-developed:

Theorem 1: A concept class C is PAC-learnable if and only if its VC-dimension is finite.

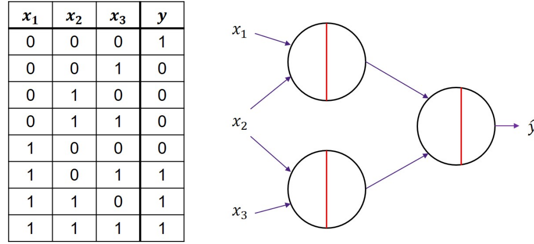
There have been several works studying on the VC-dimension of learning on TLNs [6–8]. According to [6], for a TLN with a total of W variable weights and thresholds, its VC-dimension is bounded by $6Wlog_2W$. In [7], the author also shows the VC-dimension of learning on TLN is bounded by $O(WlnN)$, where N is the number of TLGs.

As a consequence, the VC-dimension of learning on TLNs is finite. Thus, the problem is PAC-learnable, and it's theoretically feasible to adopt a machine-learning-based approach to solve the TLN weights and thresholds learning problem.

2.3 Problem Formulation

Given the threshold logic network topology $N = (V, E)$, in which the weights and thresholds of TLGs in N are undetermined, and one or more functions. The ***TLN weights and thresholds learning problem*** is to find a set of weights and thresholds for the TLGs in N such that the output values of POs (i.e., V_O) are aligned with the given function outputs under all the PI combinations. To evaluate a solution, the accuracy can be defined as the ratio of correct output values. In this work, the given functions are permissible to the TLN N; that is, there must exist a set of weights and thresholds such that all the TLN output values are aligned with the given function outputs.

Figure 1 illustrates an example for the TLN weights and thresholds learning problem. A TLN N with 3 PIs, 1 PO, and 2 normal gates is given. A function f is specified with a truth table. The problem is to determine the 6 weights and 3 thresholds for the 3 TLGs in N, such that the output value $\hat{y}$ is aligned with the given output y under each input combination (x_1, x_2, x_3). If the output values $\{\hat{y}\}$ are $\{1, 1, 0, 0, 0, 1, 1, 1\}$ under $(x_1, x_2, x_3) = \{(0, 0, 0), (0, 0, 1), ..., (1, 1, 1)\}$, respectively, the accuracy of the solution is 7/8=87.5%.



x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 1: An example for the TLN weights and thresholds learning problem.

3 FUNCTION-BASED APPROACH

3.1 TLGs and Neural Neurons

In Section 2, the funtion of a TLG is defined as Equation 1. Through transposition of terms, the output function of a TLG could be rewrited as Equation 2, where $H(\cdot)$ represents the unit step function.

$$y = \begin{cases} 1, if \Sigma_{i=1}^{n}(w_i \cdot x_i) \geq T \\ 0, otherwise \end{cases}$$
$$\iff y = \begin{cases} 1, if \Sigma_{i=1}^{n}(w_i \cdot x_i) - T \geq 0 \\ 0, otherwise \end{cases}$$
$$\iff y = H(\Sigma_{i=1}^{n}(w_i \cdot x_i) - T) \tag{2}$$

On the other hand, the output function of a neuron in neural networks can be represented by Equation 3, where w_i's are the neuron weights, x_i's are the inputs, and b is the bias.

$$y = \Sigma_{i=1}^{n}(w_i \cdot x_i) + b \tag{3}$$

In deep neural networks, the outputs of neurons are often further fed through a non-linear activation function such as a logistic function, shown in Equation 4.

$$y = L(\Sigma_{i=1}^{n}(w_i \cdot x_i) + b) \tag{4}$$

$$\text{, where } L(x) = \frac{1}{1 + e^{-ax}} \tag{5}$$

When the coefficient a in Equation 5 becomes large, $L(x)$ could approximate the unit step function $H(x)$, as shown in Figure 2. Therefore, Equation 2 and Equation 4 have the same form.

Based on the similarity, a TLG could be converted to a neuron with activation by setting $-T$ as the neuron bias and w_i's as the neuron weights. Figure 3 illustrates an example of the conversion

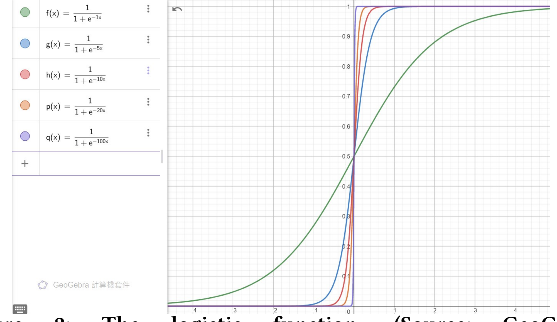


Figure 2: The logistic function. (Source: GeoGebra. https://www.geogebra.org/)

between a TLG and a neuron with activation. If all the TLGs in a TLN are converted to neurons with activation, the whole TLN becomes a neural network with logic values, i.e., a binarized neural network (BNN) [9].

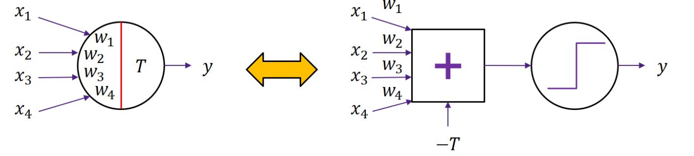


Figure 3: The conversion between a TLG and a neuron with activation.

3.2 Our Algorithm Flow

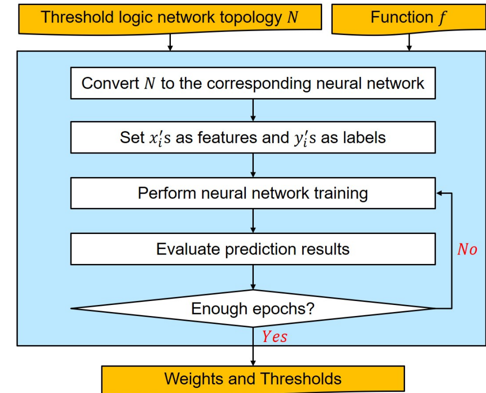


Figure 4: The algorithm flow of the function-based approach.

Inspired by the relation between TLGs and neurons, we propose the function-based approach to solve the TLN weights and thresholds learning problem. Figure 4 shows the algorithm flow. Given the TLN topology $N = (V, E)$ and **one** target function f, we first convert N to the corresponding neural network. To enhance the learnability and the training efficiency, we avoid adding activation layers at the output of the neural network. That is, only TLGs in V_G are converted to a neuron with activation, while TLGs in V_O are converted to a single neuron.

Second, for each input combination $(x_1, x_2, ..., x_n)$ of f, $(x_1, x_2, ..., x_n)$ is set as the input feature and the corresponding function output y is converted to the label y' as Equation 6. Since there is no activation layer at the last stage, the conversion is required to get aligned with the behaviors of TLNs.

$$y' = \begin{cases} 1, \text{if } y = 1 \\ -1, \text{if } y = 0 \end{cases} \tag{6}$$

Third, we perform neural network training. In each epoch, the weights and biases of neurons are extracted, and the weights and thresholds of N are set accordingly. Then, the accuracy of the solution could be evaluated. The training process continues until the specified number of epochs is reached. Finally, the weights and thresholds of the TLN N are obtained.

4 NETWORK-BASED APPROACH

4.1 Our Algorithm Flow

Different from the function-based approach, we consider a bunch of permissible functions of the given TLN N at the same time. 80% of the functions are used as the training data, while the remaining functions are for testing. Based on the training data, a deep neural network (DNN) is generated, where the input features are the given function outputs Y and the output predictions are the weights along with the thresholds for N. Then, the DNN model is applied to predict the weights and thresholds for each testing function.

4.2 The Training Process

Fig 5 shows the training process of the network-based approach along with an example function. In each epoch, one or several functions are sampled from the training data. For each function f, the output values of f are concatenated as a vector Y. For instance, the vector Y of the function in Figure 5 is $(1, 0, 0, 0, 0, 1, 1, 1)$. Y is then fed into the DNN as the input feature. The DNN model would output a set of weights and thresholds in accordance with f. After that, the weights and thresholds of N are set as the predicted values, all the input combinations of f are propagated, and the TLN output values Y' are obtained. The evaluation process compute the mean-square error (MSE) between Y and Y', which is later fed back to the DNN for parameter updatings.

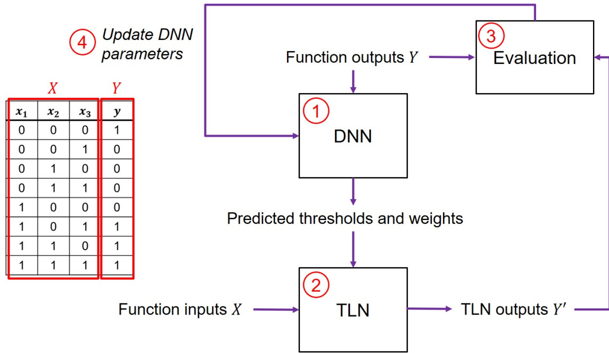


Figure 5: The training process of the network-based approach.

The DNN model consists of three fully connected linear layers, with dimensions $(2^{|V_I|} * |V_O|, 512)$, $(512, 512)$, $(512, W)$ where W is the total number of weights in N.

For the TLN propagation part in the training process, to compute the output of each gate, the weighted sum of the inputs and the threshold are compared. However, the comparison is not differentiable. Consequently, its gradient is undefined and the backward propagation of the DNN is blocked. Therefore, again, we adopted the logistic function to replace the direct comparison. Equation 7 shows the output value of a TLG in N.

$$Output = \frac{1}{1 + e^{-a(\Sigma_{i=1}^{n}(w_i \cdot x_i) - T)}} \tag{7}$$

4.3 The Testing Process

For each testing function f, the output values Y are fed into the DNN model. The model would predict the weights and thresholds for f. The accuracy could be evaluated as described in Section 2.

5 EXPERIMENTAL RESULTS

The function-based approach and the network-based approach are both implemented by the Python3 programming language with the PyTorch library. The training and evaluation processes are conducted on the Google Colab platform, with the Intel Xeon CPU @ 2.20GHz, 12GB RAM, and Nvidia K80 or Tesla T4 GPU.

5.1 Data Generation

For TLN topology generation, we first specify the number of PIs, the number of layers, and the number of TLGs in each layer. Then, there are two modes to add edge connections. In the “random” mode, a probability p is given. Between each pair of vertices in adjacent layer, an edge is added with the probability p. Note that $p = 1$ corresponds to the special case of fullt-connected. The other is the “fixed-input” mode, where the number of inputs n of each TLG is fixed. For each TLG g, n distinct TLGs from the previous layer are selected and edges are added between these TLGs and g. All the TLGs except for the POs have at least one fan-out edge.

To collect permissible functions for each generated TLN, we randomly set the weights and thresholds within [-1,1]. Then, we enumerate all the input combinations, propagate the input values through the net, and collect the outputs. As all the possible real-value weights and thresholds could be normalized to the range [-1,1], the method might be able to collect all the permissible functions.

In the following experiments, we focus on two TLNs. Case 1 is a 2-layer and fully-connected network, while Case 2 is generated based on the “fixed-input” mode. Table 1 details their statistics.

Table 1: Case statistics.

	Case 1	Case 2
level	3	5
Gate Count per Level	4 3 2	8 6 4 2 1
$\lvert V_I \rvert$	4	8
$\lvert V_O \rvert$	2	1
Total Number of Weight	18	26
Total Number of Threshold	5	13
mode	random	fixed-inputs

5.2 Function-Based Approach

As Case 2 is not fully-connected, the conversion to the corresponding neural network requires lots of manual effort. Thus, we currently focus on the Case 1. In the following experiments, 30 functions are randomly sampled for training, the batch size is set as 4, and the Adam optimizer is applied.

Table 2 shows the result with the coefficient of the logistic function a set as 1000, and Table 3 shows the result with the coefficient of the logistic function a set as 10000. There are four observations about the results:

First, as the number of epochs increases, the avg. loss decreases and the avg. accuracy increases, which implies the training procedure is effective.

Second, comparing the loss and accuracy in the two tables, it can be observed that the accuracy of $a = 1000$ outperforms that of $a = 10000$ in the early stage of the training process (#epochs = 100, 300, 500); however, the latter one outperforms the former one as the number of epochs increases and achieves a better accuracy eventually. As a higher a approximate the logistic function more precisely, it’s reasonable that the final accuracy of $a = 10000$ outperforms that of $a = 1000$. However, a higher a also implies a sharper logistic function, and thus the learnability of the neural network decreases. Therefore, the $a = 10000$ case requires more training epochs to achieve high-quality results.

Third, the avg. final accuracy is near 80%. However, the difference of accuracy between individual function is large. Some of the

functions could achieve more than 90% accuracy, while some of the functions only achieve below 50% accuracy. Thus, the training process is not quite stable. The main reason is that the number of data in the function-based approach for each function is only $2^{|V_I|}$ (i.e., 16 for case 1); therefore, it's easy to get stuck at some local optimal points. Moreover, the neural network architectures are determined by the structure of the given TLNs, which are often too simple and "shallow" to fit the given functions.

Fourth, the accuracy curve is usually not smooth in comparison with the loss curve. Figure 6 shows an example. The small size of training data and the limited neural network structure may also explain the phenomenon. Besides, it also reflects that the TLN weights and thresholds learning problem itself is somehow "discrete". Considering a TLG with two inputs. The threshold T and the weight w_2 are fixed at 0, while the inputs x_1 and x_2 are set to 1. Now, consider the relation of the output y to the weight w_1. y changes from 0 to 1 at the point $w_1 = 0$; at any other values of w_1, the output y remains the same. Similarly, for the whole TLN, the values of POs only change at some discrete points. As a result, the accuracy curve becomes unsmooth and the learning difficulty stays high.

Table 2: The experimental results of the function-based approach with a = 1000 on Case 1.

#Epochs	Avg. Loss	Avg. Accuracy
100	1.165	0.550
300	1.002	0.594
500	0.882	0.641
1000	0.698	0.654
5000	0.508	0.758

Table 3: The experimental results of the function-based approach with a = 10000 on Case 1.

#Epochs	Avg. Loss	Avg. Accuracy
100	1.272	0.476
300	1.075	0.538
500	0.928	0.582
1000	0.704	0.678
5000	0.468	0.771
8000	0.452	0.777

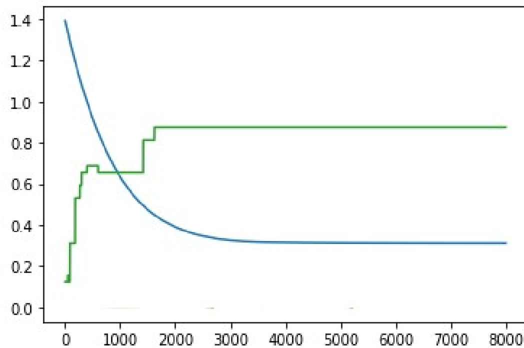


Figure 6: The loss and accuracy curves of one of the training function.

5.3 Network-Based Approach

To make the model training easier, all the thresholds are fixed as 0 during the training and testing processes. That is, we only predict weights in our experiments.

For case1, we assign the coefficient a in the logistic function, the number of epochs, and the training data size as the control variable, respectively. Fig 7 shows the effect of increasing the coefficient a. With larger a, the loss would decrease faster, and the accuracy becomes higher. The reason is that higher coefficients approximate the original unit step function better.

Table 4: The accuracy with different coefficient a.

a	accuracy
1	0.877
2	0.886
10	0.906

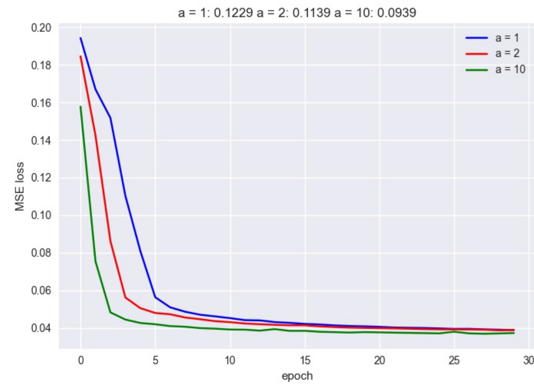


Figure 7: The MSE error with different coefficient a.

Fig 8 shows the trend of loss and error rate of Case 1. The error rate is stuck at about 0.09 after the second epoch, though the MSE loss keeps decreasing. That is, the variation of the accuracy is not very smooth and even not completely aligned with the loss. To further reduce the error rate, we may modify the model training recipe such as the model architecture, the size of batches, the optimizer, etc.

Figure 8: Loss and error rate for Case 1.

As for case 2, Figure 9 shows the trend of loss and error rate of Case 2. Since Case 2 is a more "generalized" network in comparison with Case 1, the training difficulty becomes much higher. The error rate is stuck at about 0.3 even if we modify the number of training functions, the learning rate, and the coefficient a. To further reduce the error rate, more sophisticated training techniques may be required.

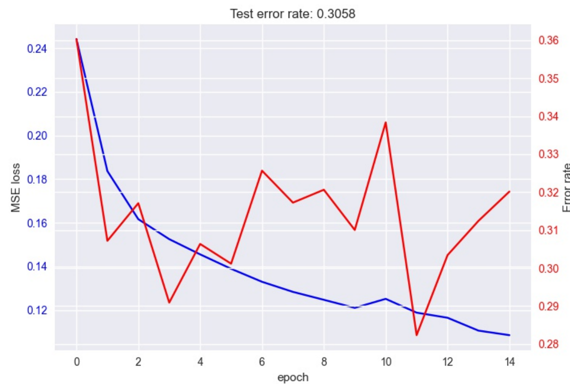


Figure 9: Loss and error rate for Case 2.

6 CONCLUSION

In this work, we propose two machine-learning-based approaches for the TLN weights and thresholds learning problem. Experimental results show that our methods achieve high prediction accuracy, especially on the fully-connected TLNs.

Future work includes thorough experiments on different test cases, such as various TLN topologies and the prediction on thresholds in the network-based approach. Our training process may also be improved to achieve better accuracy and efficiency for both approaches, and the information of the network topology could be considered during the training process. Besides, only permissible functions are considered in our experiments. We may include non-permissible functions and observe their behaviors in the future.

REFERENCES

[1] Nian-Ze Lee, Hao-Yuan Kuo, Yi-Hsiang Lai, and Jie-Hong R. Jiang. Analytic approaches to the collapse operation and equivalence verification of threshold logic circuits. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2016.

[2] Yung-Chih Chen, Li-Cheng Zheng, and Fu-Lian Wong. Optimization of threshold logic networks with node merging and wire replacement. *ACM Trans. Des. Autom. Electron. Syst.*, 24(6), oct 2019.

[3] Michael J Kearns and Umesh Vazirani. *An introduction to computational learning theory.* MIT press, 1994.

[4] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[5] Vladimir Pestov. Pac learnability versus vc dimension: a footnote to a basic result of statistical learning. In *The 2011 International Joint Conference on Neural Networks*, pages 1141–1145. IEEE, 2011.

[6] Martin Anthony. Boolean functions and artificial neural networks. 2003.

[7] Wolfgang Maass. Bounds for the computational power and learning complexity of analog neural nets. *SIAM Journal on Computing*, 26(3):708–732, 1997.

[8] Chao Zhang, Jie Yang, and Wei Wu. Binary higher order neural networks for realizing boolean functions. *IEEE Transactions on Neural Networks*, 22(5):701–713, 2011.

[9] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.