AN UPPER-BOUND ON THE REQUIRED SIZE OF A NEURAL NETWORK CLASSIFIER

Hossein Valavi, Peter J. Ramadge

Department of Electrical Engineering, Princeton University

ABSTRACT

There is growing interest in understanding the impact of architectural parameters such as depth, width, and the type of activation function on the performance of a neural network. We provide an upper-bound on the number of free parameters a ReLU-type neural network needs to exactly fit the training data. Whether a net of this size generalizes to test data will be governed by the fidelity of the training data and the applicability of the principle of Occam's Razor. We introduce the concept of s-separability and show that for the special case of $(c-1)$-separable training data with c classes, a neural network with $(d+2c)$ parameters can achieve 100% training classification accuracy, where d is the dimension of data. It is also shown that if the number of free parameters is at least $(d+2p)$, where p is the size of the training set, the neural network can memorize each training example. Finally, a framework is introduced for finding a neural network achieving a given training error, subject to an upper-bound on layer width.

Index Terms— Neural Networks, Deep Learning

1. INTRODUCTION

Selecting architectural parameters of a neural network (e.g., the depth and width of the layers) has been a long-standing and challenging problem. It is an important problem for theoretical reasons, and also for guiding the design of neural networks for a wide range of practical applications [1, 2, 3, 4, 5]. The standard method for selecting neural network (N.N.) parameters has been based on several empirical evaluations on a given dataset. Training is done on a few different candidates and the one that generalizes best is picked as the final classifier. However, estimating the required size of a neural network before training is of great importance. Of course it is also important that the selected architecture is easily trainable and that it generalizes well. Earlier results [6, 7, 8, 9, 10, 11, 12, 13] have shown that a two layer neural network with very wide hidden layer is a universal function approximator. However, such neural networks are not of practical interest due to the very large width of the hidden layer. Recently, neural networks have been designed and trained with more than a thousand layers to attain higher performance on several applications [3]. Typically, such neural networks have far more parameters than the size of the training data. This raises the question: are all of these parameters really needed. This question has multiple dimensions: are the parameters needed to obtain an accurate classifier on the training data, or to provide faster training, or better generalization, and so on. Our goal is restricted to investigating how many free parameters are needed to fit the training data exactly. This provides an upper-bound on the number of free parameters a neural network needs to classify the training data well. The guiding principle is Occam's razor: a parsimonious neural network is more likely to generalize well to testing data. To provide an upper bound, we introduce a network construction algorithm that can exactly fit the training data provided the architectural parameters are chosen appropriately. In detail, we first show that for a dataset containing c classes, and satisfying a property we call s-separable, a neural network with only 2 layers and $(d+2c)$ parameters can perfectly fit the dataset, where d is the dimension of data. An algorithm for the non s-separable case is also discussed (§3). These results provide an upper-bound on the number of parameters a neural network needs to classify a training data. Section 3 introduces the construction algorithm. Two immediate applications are the implementation of any arbitrary quantizer and any desired truth-table. Finally, we show how an l layer N.N. can be formed to precisely map each training data to its corresponding label.

2. PRIOR WORK

We focus on finding an upper-bound on the size (i.e., number of free-parameters) for a N.N. that can exactly fit a training dataset. Prior work has studied this problem through functional analysis ([6, 7, 8, 9, 10, 11, 12, 13]), or via evaluating the expressivisity in terms of the number of linear regions the final decision boundary can form ([14, 15, 16, 17, 18]), or via studying the trajectory length of the output layer ([19]). In contrast we address the issue by using a network construction algorithm. This is inspired by the recent work presented in [20]. This work shows that conventional complexity metrics (such as VC-dimension or Rademacher complexity) are unable to adequately characterize the complexity of a neural net. Further, the authors prove that a 2 layer N.N. with p neurons in the hidden layer, can fit a random noise training set of p examples . Our goal is to analyze the expressive power of a neural net for classification. We aim to understand how many parameters are needed to classify the training data exactly and how to bound the architectural parameters of such a neural net. That particular network may not generalize, but

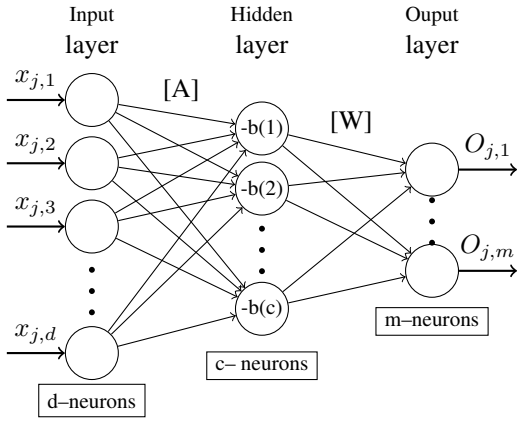

Fig. 1: Constructed neural network with one hidden layer of size c that achieves 100% training data classification accuracy. $A \in \mathbb{R}^{d\times c} = a\mathbf{1}^T, a \in \mathbb{R}^d, \mathbf{1} \in \mathbb{R}^c$ and $b \in \mathbb{R}^c$ encode the first layer's parameters. $W \in \mathbb{R}^{c\times m}$ encodes the parameters of the last layer.

under the principle of Occam's razor, it bounds of the size of neural network that is likely to generalize well.

3. NETWORK CONSTRUCTION

This section introduces an algorithm for constructing a neural network classifier that maps each example in a labeled training set to its corresponding label. This is a preliminary result. As two immediate applcations, we show how to apply this construction to implement any quantizer, and to implement any truth-table. Using N.N.s to implement truth tables is a classical problem (see [21], [22], [23], [24]).

3.1. s-Separable Training Data

Let $\{(x_i, y_i)\}_{i=1}^p$ be a dataset with $x_i \in \mathbb{R}^d$ and $y_i \in [1:c]$. We say that the dataset is s-separable if there exist a vector a in $\mathbb{R}^d$ such that projecting the data onto a creates $(s+1)$ intervals each containing points with the same label. Note that s can be larger than $(c-1)$; which means all data from one particular class need not be in the same interval. This is a broader form of linear separability for a c class dataset.

Lemma 3.1. *Consider a N.N. with m output neurons, and let f be an injective mapping from the class labels into R^m. If the training dataset is $(s-1)$-separable, then there exists a neural network with depth 2 and $(d+(m+1)s)$ parameters that maps training examples in class j to $f(j)$.*

Proof. We first prove the result for $(c-1)$-separable data. W.l.o.g, assume the first k_1 examples belong to class 1, the next k_2 to class 2, and the last k_c to class c. $(c-1)$-separability implies there exist vectors $a \in \mathbb{R}^d, b \in \mathbb{R}^c$ such that:

$$\begin{aligned} b(1) < a^T x_1 < a^T x_2 < \ldots < a^T x_{k_1} < b(2) \\ < a^T x_{k_1+1} < \ldots < a^T x_{k_2} < b(3) \\ < \ldots < b(c) < a^T x_{k_{c-1}+1} < \ldots < a^T x_p \end{aligned} \tag{1}$$

We prove that if (1) is satisfied, then the parameters of the neural network in Fig. 1 can be selected to attain 100% training classification accuracy. Let the vector h_j represent the output of the first hidden layer (after applying ReLU) when x_j is applied at the input (i.e., $h_j = \rho((a^T x_j)\mathbf{1} - b)$), and $H \in \mathbb{R}^{p\times c}$ be the matrix with h_j^T on its j-th row. It suffices to find a matrix $W \in \mathbb{R}^{c\times m}$ such that $HW = Y$ where $Y \in \mathbb{R}^{p\times m}$ represents the corresponding labels (e.g., one-hot encoding of the class-labels). Let $f(j) \in \mathbb{R}^m$ be the vector representation of the label of the i-th class. Then, matrix Y can be written as :

$$Y = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \ldots & 1 \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix} \begin{bmatrix} - - f(1)^T - - \\ - - f(2)^T - - \\ \vdots \\ - - f(c)^T - - \end{bmatrix} \tag{2}$$

Denote the first $p \times c$ matrix by E and the second matrix with $f(j)^T$ on its $j-$th row by F. Matrix E has orthogonal columns and $E^T E = D$, where $D \in \mathbb{R}^{c\times c}$ is a diagonal matrix with k_j on its j-th diagonal entry. Hence, it suffices to find a matrix W such that $HW = EF$ or equivalently, $E^T HW = DF$. The matrix $E^T H$ can be written as:

$$E^T H = \begin{bmatrix} (h_1 + h_2 + \ldots + h_{k_1})^T \\ (h_{k_1+1} + \ldots + h_{k_1+k_2})^T \\ \vdots \\ (h_{k_{c-1}+1} + \ldots + h_p)^T \end{bmatrix} \tag{3}$$

Referring to inequality (1), $E^T H$ will be a $c \times c$ lower triangular matrix with positive diagonal entries, and hence is invertible. Setting $W = (E^T H)^{-1} DF$ will create a neural network that perfectly maps each training data onto its corresponding m-dimensional label. The number of parameters of the N.N. is $(d+c+(c\times m))$ for (a, b, W), respectively. For the general case of s-separable dataset with $(s > c-1)$, matrix Y will not be arranged in the order shown in (2). However, a permutation matrix P can be found such that PY has the desired form. Then the problem simplifies to finding a matrix W such that $PHW = PY$, which can be solved similarly. □

If the MNIST digits dataset [25], was $(c-1)$-separable (it is not), the design method outlined above would yield a N.N. with a hidden layer of 10 neurons and an output layer of 10 neurons. Such a two layer network trained using the Adam optimization algorithm [26] can fit 95% of the training data. Moreover, this trained network achieves 7.4% testing

error. These are not spectacular numbers. But the experiment suggests that a simple architecture can achieve a great deal, and the bulk of the complexity is needed to reduce the last 10% of training and testing error.

Corollary 3.1. *Let N be a ReLU type N.N. with L hidden layers. If there exists an injective mapping from the training data to the output of the penultimate hidden layer, then the parameters of N can be selected such that each training example is mapped to its corresponding label.*

Proof. Let f represent the corresponding injective mapping and consider the last two hidden layers. Since f is injective, training data get mapped to distinctive outputs at the penultimate hidden layer. Now considering $\{(f(x_j), y_j), j \in [1:p]\}$ as the new training set and using Lemma 3.1, there exist a N.N. that can map all $f(x_j)$ to y_j, for $j \in [1:p]$. □

As an example, we show how to use Lemma 3.1 to implement an arbitrary quantizer using a neural net. Quantizers have numerous applications in signal processing.

Corollary 3.2. *Let $a \in \mathbb{R}^d$, and $\text{span}(a)$ be divided into n disjoint intervals. Let f be a mapping from $\text{span}(a)$ to $\mathbb{R}$. If f maps the elements in each interval to the same real number, then there exists a N.N. with $(d + 2n)$ parameters that maps each x in the i-th interval of $\text{span}(a)$ to $f(i)$.*

Proof. Since it is assumed that all datapoints that fall in each interval get mapped to the same real number via f, the problem can be viewed as classification of a $(n-1)$-separable dataset. The construction algorithm introduced in the Lemma 3.1 can be utilized to create the desired neural net. □

As a second example, we show how to implement an arbitrary b-bit truth table with a neural net. This result will be used in later sections of the paper. Unlike conventional methods that break down the implementation of a truth-table into a combination of primitive logic gates ([21], [22], [23], [24]), we show an implementation based on Lemma 3.1.

Corollary 3.3. *Any b-bit truth table (with 2^b or fewer entries) can be implemented with a ReLU-type N.N. of depth 2.*

Proof. A function f can be found to map entries of a b-bit truth table to $\mathbb{R}^{2^b}$. By doing so, implementing any truth-table simplifies to a classification problem, i.e., find a classifier that maps the vectors in $\mathbb{R}^{2^b}$ to the outputs in the truth-table. Suppose in total, there are p entries that generate c distinct outputs. Further, suppose entries are ordered such that the first k_1 entries generate result 1, the next k_2 entries generate result 2, etc., and the last k_c entries generate result c. One f that generates a $(c-1)$-separable dataset, is $f(j) = je_j, j \in [1:p]$, where e_j is the j-th standard basis. By Lemma 3.1, there is a 2 layer ReLU-type N.N. that implements f. □

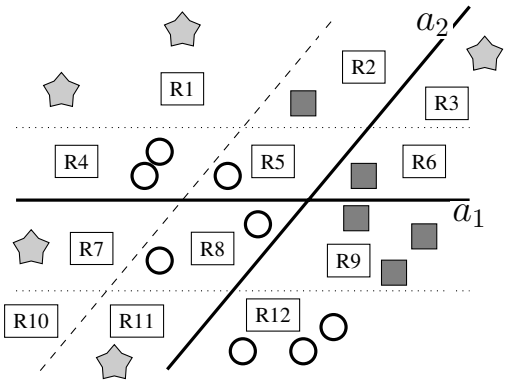

Fig. 2: Different non-overlapping 2D regions created by diving the span of two vectors $a_1, a_2 \in \mathbb{R}^2$ into 3 and 4 intervals, respectively.

3.2. Beyond s-Separable Data

For the broader case of non s-separable training data, Lemmas 3.2 and 3.3 introduce a 4 and an L-layer construction.

Lemma 3.2 (Width)**.** *Let $\{a_i\}_{i=1}^n$ be linearly independent vectors in $\mathbb{R}^d$, and divide $\text{span}(a_i)$ into k_i disjoint intervals, $i \in [1:n]$. This partitions $\mathbb{R}^d$ into $\prod_{i=1}^n k_i$ regions. Let f map $\mathbb{R}^d$ into $\mathbb{R}^m$ with c distinct vectors in its range. If f maps all points in a region to the same value, then there exists a neural network with the depth of 4 and $(d+n+2(c+\sum k_i))$ parameters that implements f.*

This lemma sets an upper-bound on the width of a 4-layer N.N that can exactly classify the training data. The width of the hidden layers is determined by the number of projection vectors, number of thresholds, and the number of classes.

Proof. Perfect classification is possible based on the intervals of $\text{span}(a_i)$, $i \in [1:n]$, and a truth-table. Lemma 3.1 can be utilized to implement each classifier in parallel. This will form the first and second hidden layers. The third and fourth layers are formed based on Corollary 3.3 for implementing a truth-table. The above procedure and the desired N.N. are depicted in Fig. 2 and Fig. 3, respectively. □

The lemma below gives a construction for an L-layer N.N. that exactly classifies the training data. The width of the j-th hidden layer is at most k_j+t, where t is the number of training examples classified incorrectly by the previous hidden layers.

Lemma 3.3 (Depth)**.** *Let $\{a_i\}_{i=1}^L$ be a linearly independent set in $\mathbb{R}^d$. For $i \in [1:L]$, divide $\text{span}(a_i)$ into k_i disjoint intervals, such that if a region in the resulting partition of $\mathbb{R}^d$ contains training examples, these examples have the same label. Then there exists a depth L neural network mapping each training example to its corresponding label.*

Proof. We first prove that there exists an L layer N.N. using the sign activation function that achieves 100% training accuracy. Then, we show how to do this using ReLU N.N.s. The

N.N. is formed in L hierarchical steps; corresponding to L hidden layers. In the first step, each training example is projected onto $\text{span}(a_1)$ and classified w.r.t. k_1 intervals along $\text{span}(a_1)$. We say that such an interval is homogeneous if all projected vectors in that interval have the same label. If all k_1 intervals along $\text{span}(a_1)$ are homogeneous, Lemma 3.1, gives a 2-layer N.N. with one hidden layer consisting of k_1 neurons that can classify all training examples correctly.

Otherwise, let $I_1, \ldots, I_{q_1}$ denote the homogeneous intervals and $m_j = |I_j|$ denote the number of projected training examples in I_j, $j \in [1\!:\!q_1]$. Let $r_1 = p - \sum_{j=1}^{q_1} m_j$. W.l.o.g., let $\{a_1^T x_1, \ldots, a_1^T x_{m_1}\} \in I_1$, $\{a_1^T x_{m_1+1}, \ldots, a_1^T x_{m_1+m_2}\} \in I_2$, etc, and $\{a_1^T x_{r_1+1}\}, \ldots, \{a_1^T x_p\}$ be in non-homogeneous intervals. Using the notation from Lemma 3.1, we form the first hidden layer of the N.N. using $h_1 = (q_1 + r_1)$ neurons, setting $A = a_1 1^T$ and $b = (b_1, \ldots, b_{h_1})$ such that:

$$
\begin{aligned}
& b_1 < a_1^T x_1 < \ldots < a_1^T x_{m_1} < \\
& b_2 < a_1^T x_{m_1+1} < \ldots < a_1^T x_{m_1+m_2} < \\
& \quad \vdots < \quad \vdots \quad < \\
& b_{q_1+1} < a_1^T x_{r_1+1} < b_{q_1+2} < \ldots < b_{h_1} < a_1^T x_p
\end{aligned}
$$

By satisfying the above, the first layer with sign activation maps the original data in $\mathbb{R}^p$ to $\mathbb{R}^{h_1}$, with each $x \in I_j$ mapped to $(e_1 + \cdots + e_j)$, $j \in [1\!:\!q_1]$, with the remaining examples mapped to $(e_1 + \cdots + e_{q_1+1}), \ldots, (e_1 + \cdots + e_{h_1})$, where e_j is the j-th standard basis in $\mathbb{R}^{h_1}$.

In the second construction phase, $\{e_1, (e_1+e_2), \ldots, (e_1+\cdots+e_{h_1})\}$ in the first hidden layer, are used as the new training examples and the above procedure is repeated for the new projection vector a_2. This can be repeated until either a stopping criterion is reached or all resulting sub-intervals are homogeneous. Either case requires at most L steps. For a ReLU-type N.N., a similar method can be used; the difference being that elements in an homogeneous interval I_j are going to be mapped to a scalar multiple of $(e_1 + \cdots + e_j)$. An extra hidden layer, using lemma 3.3, can be added to map those elements back to $(e_1 + \cdots + e_j)$. Alg. 1 illustrates the N.N. construction for binary classification. □

Computational results on MNIST [25], show that a ReLU N.N. with 3 hidden layers, each consisting of 10 neurons, can correctly classify $\sim 95.8\%$ of the training data; about 400 more training examples than a similar 2 layer net, with almost equivalent testing accuracy. Suggesting that refinement of the previously incorrectly classified training examples generalizes to the testing data.

4. CONCLUSION

Motivated by results in [20], we have given a construction method for a neural network classifier based on a family of vector projections and thresholding. This classifier can exactly classify training data, or stop training once the training error is below a desired error upper bound. By appealing to principle of Occam's Razor, this provides an upper-bound on the size of a neural network classifier that we reasonably expect to generalize well. There are still open questions on the best ways to selecting the projection vectors, the threshold values, and obtaining exact generalization bounds. These questions are the subject of on-going research.

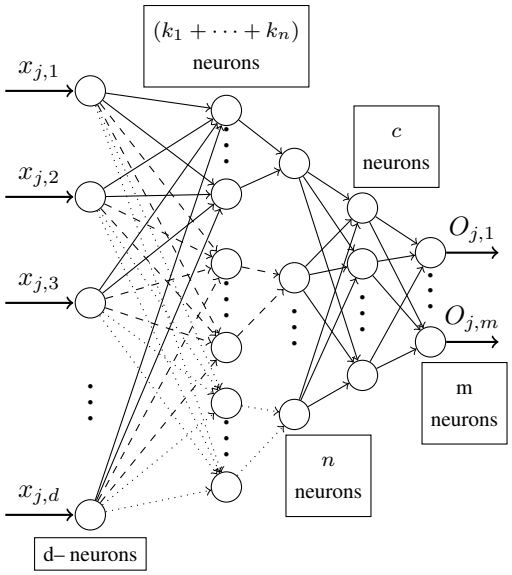

Fig. 3: A 4-layer neural network that maps each $x \in \mathbb{R}^d$ in the j-th high-dimensional region to $f(j)$.

Algorithm 1 N.N. Arch. Design for binary classification

Input: Data matrix $X \in \mathbb{R}^{p \times d}$, labels $Y \in \mathbb{R}^{p \times m}$ and given training error ERR.

Initialize:
- $l = 1$
- Perform SVM to get $a_1 \in \mathbb{R}^d$
- Set k_1 thresholds to get $b_1 \in \mathbb{R}^{k_1}$
- $err = \#$ points in multi-label intervals
- Form the 1^{st} layer of N.N.

while $err \leq ERR$
- X : Remaining data not classified correctly
- Select $a_l \in \mathbb{R}^d$.
- Project X onto a_l
- Select k_l thresholds to find $b_l \in \mathbb{R}^{k_l}$.
- Form l-th layer of the N.N.
- l++

5. REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[2] X. Pan and V. Srikumar, "Expressiveness of rectifier networks," in *International Conference on Machine Learning*, 2016, pp. 2427–2435.

[3] 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.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[5] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[6] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[7] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, no. 5, pp. 551–560, 1990.

[8] K. Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, no. 8, pp. 1069–1072, 1993.

[9] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

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

[11] M. Leshno, V. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993.

[12] S. Ferrari and R. Stengel, "Smooth function approximation using neural networks," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 24–38, 2005.

[13] R. Jones, Y. Lee, C. Barnes, G. Flake, K. Lee, P. Lewis, and S. Qian, "Function approximation and time series prediction with neural networks," *Neural Networks*, pp. 649–665, 1990.

[14] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *International Conference on Algorithmic Learning Theory*. Springer, 2011, pp. 18–36.

[15] R. Pascanu, G. Montufar, and Y. Bengio, "On the number of response regions of deep feed forward networks with piece-wise linear activations," *arXiv preprint arXiv:1312.6098*, 2013.

[16] G. 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.

[17] A. Daniely, R. Frostig, and Y. Singer, "Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity," in *Advances In Neural Information Processing Systems*, 2016, pp. 2253–2261.

[18] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," in *Conference on Learning Theory*, 2016, pp. 907–940.

[19] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," *arXiv preprint arXiv:1606.05336*, 2016.

[20] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv preprint arXiv:1611.03530*, 2016.

[21] I. Aleksander and H. Morton, *An Introduction to Neural Computing*, vol. 3, Chapman & Hall London, 1990.

[22] S. Chakradhar, V. Agrawal, and M. Bushnell, "Neural net and boolean satisfiability models of logic circuits," *IEEE Design & Test of Computers*, vol. 7, no. 5, pp. 54–57, 1990.

[23] I. Taha and J. Ghosh, "Symbolic interpretation of artificial neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 3, pp. 448–463, 1999.

[24] L. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*, Cambridge University Press, 2002.

[25] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[26] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.