The Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI-25)

On the Hardness of Training Deep Neural Networks Discretely

Ilan Doron-Arad

Technion – Israel Institute of Technology, Haifa, Israel
ilan.d.a.s.d@gmail.com

Abstract

We study *neural network training (NNT)*: optimizing a neural network's parameters to minimize the training loss over a given dataset. NNT has been studied extensively under theoretic lenses, mainly on two-layer networks with linear or ReLU activation functions where the parameters can take any real value (here referred to as *continuous* NNT (C-NNT)). However, less is known about deeper neural networks, which exhibit substantially stronger capabilities in practice. In addition, the complexity of the *discrete* variant of the problem (D-NNT in short), in which the parameters are taken from a given finite set of options, has remained less explored despite its theoretical and practical significance.

In this work, we show that the hardness of NNT is dramatically affected by the network depth. Specifically, we show that, under standard complexity assumptions, D-NNT is not in the complexity class NP even for instances with fixed dimensions and dataset size, having a deep architecture. This separates D-NNT from any NP-complete problem. Furthermore, using a polynomial reduction we show that the above result also holds for C-NNT, albeit with more structured instances. We complement these results with a comprehensive list of NP-hardness lower bounds for D-NNT on two-layer networks, showing that fixing the number of dimensions, the dataset size, or the number of neurons in the hidden layer leaves the problem challenging. Finally, we obtain a pseudo-polynomial algorithm for D-NNT on a two-layer network with a fixed dataset size.

Extended version — http://arxiv.org/abs/2412.13057

1 Introduction

Deep neural networks are among the leading tools in machine learning today, with an astonishingly vast range of applications (Goodfellow, Bengio, and Courville 2016). Neural networks require training: optimizing the weights and biases on a given dataset. While minimizing the generalization error is the primary goal, a fundamental algorithmic challenge is to minimize the training loss on the given dataset.

Neural network training has been studied extensively from a theoretical perspective. NP-hardness proofs have been known for over three decades (Judd 1988; Blum and Rivest 1988; Megiddo 1988) and recent developments obtained strong hardness results for various special cases of the training problem (e.g., (Goel et al. 2021; Boob, Dey, and Lan 2022; Bertschinger et al. 2023; Froese and Hertrich 2024)). For the above results, it sufficed to consider two-layer networks which already capture the hardness of the problem.

On the other hand, less is known about deeper neural networks, which adhere to substantially stronger capabilities in practice (Goodfellow, Bengio, and Courville 2016). Obtaining lower bounds of training as a function of the network depth is considered a prominent question in previous works (e.g., (Goel et al. 2021; Froese and Hertrich 2024)). In addition, the complexity of the *discrete* variant of the problem, in which the parameters are taken from a given finite set of options, has remained less explored despite its theoretical and practical significance.

In this paper, we aim to narrow the gaps described above, by theoretically studying the following questions. (i) In polynomially solvable settings for shallow networks (e.g., fixed dimension and dataset size, etc.), how difficult is training on deeper networks? (ii) What are the complexity differences between training on discrete versus continuous parameter spaces? (iii) In particular, can we obtain a hardness results stronger than NP-completeness for training on a discrete parameter space? such results exist for two-layer networks in the continuous parameter space (Bertschinger et al. 2023; Abrahamsen, Kleist, and Miltzow 2021). We start by formally describing the problem.

Neural Network Training (NNT)

We define the decision problem of whether a given neural network can be optimally trained. This problem is occasionally referred to as *empirical risk minimization (ERM)* but hereafter will be called *neural network training (NNT)*. The definition captures both discrete and continuous parameter spaces with arbitrary activation and loss functions on an arbitrary directed acyclic graph.

Input: We are given a directed acyclic graph $N = (V = \{s\} \cup H \cup T, E)$ (i.e., the network), where s, H, T are the input vertex (source), the *hidden layers* (neurons), and the output vertices, respectively. The *depth* of the network is the longest path from the input to an output and a *layer* is a maximal subset of vertices with the same distance from the

source; the *width*, denoted by k, is the maximum cardinality of any layer. We also have a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, with input $x_i \in \mathcal{X} \subseteq \mathbb{R}^d$ and output $y_i \in \mathcal{Y} \subseteq \mathbb{R}^{m \cdot |T|}$.

We are also given an activation function $\sigma^v : \mathbb{R} \to \mathbb{R}$ for each $v \in H \cup T$ and a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ satisfying $\mathcal{L}(a, b) = 0$ if and only if $a = b$. We assume that the activation functions and the loss function are polynomially computable. Additionally, the input consists of weight spaces $W_e[i] \subseteq \mathbb{R}$ and bias spaces $B_e \subseteq \mathbb{R}$, for all $e \in E$ and dimension $i \in \{1, \ldots, d\}$.[1] Let $W_e = (W_e[i])_{i \in \{1,\ldots,d\}}$ and let $\Theta = (W_e, B_e)_{e \in E}$ be the *parameter space*. Finally, we are given an error margin parameter $\gamma \in \mathbb{R}$.

The Model: Given parameters $\theta = (w_e, b_e)_{e \in E} \in \Theta$, that is, $w_e[i] \in W_e[i]$ and $b_e \in B_e$ for all $e \in E$ and $i \in \{1, \ldots, d\}$, the model $f_\theta(x)$ is computed inductively given an input $x \in \mathcal{X}$. Define $z^s(x) = x$. Moreover, for each $v \in H \cup T$, the output $z^v(x)$ of v is defined by applying the activation function σ^v on the weighted sum of outputs of all vertices with an edge directed towards v: $z^v(x) = \sigma^v\left(\sum_{e=(u,v) \in E} w^e \cdot z^u(x) + b^e\right)$. The output of the network is $f_\theta(x) = (z^t(x))_{t \in T}$.

Objective: Decide if there are parameters $\theta \in \Theta$ that admit a training loss bounded by the error margin parameter: $\sum_{i=1}^{n} \mathcal{L}\left(f_\theta(x_i), y_i\right) \leq \gamma$. We refer to the well-studied special case of NNT, where $W_e, B_e = \mathbb{R}$ for every edge e, as *continuous NNT (C-NNT)*; conversely, if the sets W_e, B_e are finite and are explicitly given as part of the input, the problem is called *discrete NNT (D-NNT)*.

Motivation: As gradient decent techniques are a form of C-NNT and ubiquitous in practice, the practical motivation for studying C-NNT is apparent. However, there are also settings in which D-NNT possesses practical significance. *Quantization* techniques, transforming C-NNT into D-NNT, are very common in practice (Courbariaux, Bengio, and David 2015; Rastegari et al. 2016; Zhu et al. 2016; Zhou et al. 2018; Rokh, Azarpeyvand, and Khanteymoori 2023). These techniques enable to reduce memory usage and energy consumption while increasing the speed of both training and deployment. In some cases, quantization even allows better model generalization.

Other practical settings in which D-NNT is useful include model deployment in low-memory devices (Li and Hao 2018), initial parameter values for gradient-based optimization (Hu, Xiao, and Pennington 2020), or even long-term information storage in biological neural networks (Hayer and Bhalla 2005; Miller et al. 2005) (see more details in (Baldassi and Braunstein 2015)).

D-NNT versus C-NNT: One key difference between D-NNT and C-NNT is the over-parametrized setting (where $k > n$). While a C-NNT instance can always optimally fit the dataset in the over-parametrized setting (Zhang et al. 2021), this does not always hold for D-NNT instances. As a naive example, consider a two-layer D-NNT instance with $k = 2$ neurons and one data point $(x = 1, y = 2)$. If the parameter space of the instance contains only zeros, it is impossible to construct a model that optimally fits the dataset. This distinction enables lower bounds and non-trivial algorithms for D-NNT in the over-parametrized setting.

[1] Technically, for edges $e = (u, v) \in E$ where $u \neq s$, there is only one dimension: $W_e = W_e[i] \ \forall i \in \{1, \ldots, d\}$.

Our Results

Hardness of NNT on Deep Networks In the following, we partially answer questions (i)-(iii). As mentioned above, there are lower bounds stronger than NP-completeness for NNT on a two-layer network (Bertschinger et al. 2023; Abrahamsen, Kleist, and Miltzow 2021). However, the intricate part in these hard C-NNT instances is based on the infinite parameter space and infinite precision of real numbers. In contrast, for analogous D-NNT instances on a two-layer network, verifying the correctness of a solution can be done in polynomial time since the entire parameter space is given explicitly in the input. Therefore, D-NNT on shallow networks belongs to NP and a strictly stronger lower bound than NP-completeness for D-NNT would require deeper networks.

We obtain the following lower bound on the running time of D-NNT in *deep* networks. Specifically, we show that D-NNT is unlikely to be in the complexity class NP, making it significantly more challenging than any NP-complete problem (such as, e.g., the closely related Circuit SAT (Lu et al. 2003; Amizadeh, Matusevych, and Weimer 2019)).

Theorem 1.1. *Assume that the* Constructive Univariate Radical Conjecture *(Dutta, Saxena, and Sinhababu 2022) is true and assume* NP $\not\subseteq$ BPP. *Then,* D-NNT $\notin$ NP*, even for instances with input and output dimension* 1 *and weights in* $\{0, 1, -1\}$.

Even though our result considers the discrete version of NNT, we give an easy polynomial reduction from D-NNT to C-NNT. This shows that our lower bound for the discrete problem also applies to its continuous counterpart, albeit for instances with a large output dimension.

Theorem 1.2. *There is a polynomial time reduction from* D-NNT *to* C-NNT.

D-NNT on Two-Layer Networks Our above results shed some light on questions (i) and (iii). In the remainder of this section, we focus on question (ii), discussing D-NNT instances on two-layer neural networks. Our first easy result shows that even in a very restricted scenario, D-NNT is NP-hard. This setting incorporates a dataset of size 1, a regime that is easily solvable in the continuous setting. This gives a another stark distinction between D-NNT and C-NNT.

Theorem 1.3. *Unless* P=NP*, there is no polynomial algorithm that solves* D-NNT *even if there is only one hidden layer, the dataset is of size* $n = 1$*, the dimension is* $d = 1$*, the activation functions are the identity function, and the loss function is the sum of squares.*

In the above result, the hardness of D-NNT follows directly from the maximum number allowed for a weight or a bias. Therefore, henceforth the weights and biases are given in unary; this allows assessing the complexity assuming the parameters are relatively small (polynomial in the

input size). In this setting, we obtain the following strong lower bound on the running time even if the weights are encoded in unary.

Theorem 1.4. *Assuming the* Exponential Time Hypothesis (ETH)*, no algorithm solves* D-NNT *in time* $f(k) \cdot N^{o\left(\frac{k}{\log k}\right)}$ *where* N *is the encoding size of the instance,* k *is the width of the network, and* f *is any computable function. The result holds even for instances with the identity activation function and input dimension* $d = 1$.

While there are ETH-based lower bounds for NNT, we are only aware of such bounds for C-NNT instances with *Rectified Linear Unit (ReLU)* activation functions in which the parameter is d rather than k (Froese, Hertrich, and Niedermeier 2022a). In contrast, in the above result (and also in the next) the activation function used is the identity function. Such a result is unlikely for C-NNT.

Finally, we obtain a hardness result for instances with a single neuron in the hidden layer (width $k = 1$). This result is different from other hardness results with one neuron (Dey, Wang, and Xie 2020; Goel et al. 2021; Froese, Hertrich, and Niedermeier 2022b), since we consider D-NNT rather than C-NNT and our result applies to the identity function as the activation function.

Theorem 1.5. *Assuming* P $\neq$ NP*, there is no polynomial algorithm for* D-NNT *even with one input, one output, one neuron in the hidden layer* $k = 1$*, and the identity activation function.*

We complement the above lower bounds with a pseudo-polynomial algorithm for D-NNT on a two-layer network with general activation functions (on the hidden layer only) and a general loss function. This algorithm studies the *over-parametrized* setting, in which $k > n$. That is, we assume that the number of neurons in the hidden layer is unbounded, the input dimension is unbounded, and the output dimension is one. On the other hand, we assume that the dataset size is a fixed constant. Finally, the running time is pseudo-polynomial in the maximum number that can be computed via the network.

Theorem 1.6. *There is an algorithm that decides* D-NNT *on a two-layer network with output dimension* 1 *in time* $\text{poly}(|I|) \cdot M^{O(n_0)}$*, where* M *is the maximum number that can be computed via the network and* n_0 *is the size of the dataset.*

By Theorem 1.4 and Theorem 1.5, obtaining qualitatively better running time for the above regime is improbable. This algorithm is intended as a theoretic proof-of-concept and is unlikely to perform well in practice.

Discussion

This section describes the implications and limitations of our results and lists some open questions.

Hardness of NNT on Deep Networks: Theorem 1.1 indicates that even very simplified D-NNT instances on a deep network architecture are strictly harder than any NP-complete problem. This result implies that, unlike other NNT lower bounds, the depth of the network can be isolated as the sole factor contributing to the hardness of training. This is in contrast to other stronger than NP-completeness lower bounds that are applied to two-layer networks (Bertschinger et al. 2023; Abrahamsen, Kleist, and Miltzow 2021).

Several questions remain open here. It would be interesting to obtain even stronger hardness results for deep networks exploiting more general instances with larger dimensions and unbounded dataset size. In addition, our reduction to C-NNT (Theorem 1.2) augments the output dimension; we would like to reproduce the lower bound of Theorem 1.1 for C-NNT with a smaller number of outputs, which would require a larger dataset and potentially more expressive activation functions.

Discrete vs. Continuous NNT: One key distinction between D-NNT and C-NNT, exemplified in our results, is that D-NNT is often intractable even for instances considered to be naive in the continuous parameter space (e.g., as in the over-parametrized setting (Zhang et al. 2021)). On the other hand, D-NNT instances can be solved by exhaustive enumeration, unlike C-NNT. Despite the above distinctions, the theoretical dynamics between the discrete and continuous settings are not fully understood and should be a subject to future works. This question is especially interesting due to the large body of work on quantized models (Courbariaux, Bengio, and David 2015; Rastegari et al. 2016; Zhu et al. 2016; Zhou et al. 2018; Rokh, Azarpeyvand, and Khanteymoori 2023). Our results give indications for the hardness of training quantized models in the worst case. However, in practical settings the quantization is not adversarial and can be chosen in various ways. Thus, the lower bounds in our paper do not give a direct implication for practical quantization schemes.

Handling Large Numbers: The crux in the hardness of deep networks presented in Theorem 1.1 is based on representing fast-increasing numbers in the network. Note that this does not depend on the maximum absolute parameter value, which is 1 in our reduction. Thus, our results provides additional theoretical angle on numerical instability observed in the area of deep neural networks (e.g., (Hochreiter 1998; Zheng et al. 2016; Sun et al. 2022)). Nevertheless, in practice, numbers used in deep networks do not usually grow as fast as in our lower bound, designed solely to explore the limitations of deep learning in theory.

It is intriguing to study further the complexity of the problem allowing the running time to pseudo-polynomially depend on the maximum number that can be computed via the network. Indeed, in Theorem 1.6 we obtain an algorithm with such a running time in the over-parametrized discrete setting (on two-layer networks). Pseudo-polynomial lower bounds, especially for deeper networks, would be interesting here. In addition, Theorem 1.4 describes a scenario in which the encoding size is determined by the maximum weight in the instance. It would be interesting to improve the lower bound to $f(k) \cdot N^{\Omega(k)}$ under the same conditions.

Number of Inputs: This paper studies NNT only on instances with a single source (which can be multidimensional though). Even with one input, NNT is computationally challenging, with various lower bounds (see the previous work section). Nonetheless, it would be interesting to explore NNT with multiple inputs in a simplified environment.

Activation Functions: In Theorem 1.1, we use as a proof-of-concept activation functions that perform as division and multiplication operators. Even though these activation functions are non-standard, they can be efficiently encoded and computed, consequently fitting to polynomial reductions. It would be interesting to use standard activation functions such as ReLU or linear activation functions for obtaining lower bounds in deep networks.

Approximation Algorithms: This paper focuses on obtaining exact theoretical lower bounds for NNT. However, in most practical settings approximate solutions may suffice. We remark that for instances with parameter error $\gamma = 0$, e.g., Theorem 1.3 and Theorem 1.5, there cannot be any multiplicative approximation ratio. Thus, the best we can expect is an additive approximation (Goel et al. 2021). It remains an open question whether Theorem 1.1 can be adapted to rule out multiplicative or additive approximation for (relatively easy) deep networks. We remark however that using simple scaling, the lower bound described in Theorem 1.1 can be applied for arbitrarily large values of γ.

Connection to Learning: In practice, the goal of training is to minimize the generalization loss on unseen data. Conversely, this paper focuses on loss minimization on a static dataset. While there are rooted connections between training and learning (e.g., (Shalev-Shwartz and Ben-David 2014; Goel et al. 2017, 2021)), it would be interesting to find non-trivial assumptions on the data distribution and design analogous lower bound to Theorem 1.1 in a learning perspective.

Scaling of Hyper-Parameters The width plays a pivotal role in Theorem 1.3 and cannot be reduced using our reduction. On the other hand, in Theorem 1.4, the width k roughly expresses the number of constraints and the weight space n describes the size of the alphabet of a *2-constraint satisfaction problem (2-CSP)* instance (Marx 2007). Observe that Theorem 1.4 focuses on the *parameterized* regime where $n \gg k$; however, our reduction can also yield $\Omega\left(n^k\right)$ ETH-based lower bounds for the regime $n = O(1)$. Finally, in Theorem 1.5, the size of the dataset and the dimension can scale up to a polynomial factor, based on the hardness of Set Cover (Raz and Safra 1997; Dinur and Steurer 2014).

Previous Work

Due to the immense body of work on neural network training, we limit this section to purely theoretical results. There has been a long line of work on training neural networks with continuous weights (C-NNT), with NP-hardness proofs for various special cases (Shalev-Shwartz and Ben-David 2014; Blum and Rivest 1988). As a fundamental special case, most research focused on (fully connected) networks with two layers leading to a comprehensive understanding of such instances. The parameters n, k, d, and γ mentioned in the following results are analogous to the parameters in our definition of the NNT problem.

A significant portion of previous work studied NNT with ReLU activation functions. Under the above restrictions, the problem was proven to be NP-Hard (Dey, Wang, and Xie 2020; Goel et al. 2021) even for $k = 1$. For the same regime, a stronger lower bound of $n^{\Omega(d)}$ on the running time has been given by (Froese, Hertrich, and Niedermeier 2022b) assuming ETH (see Conjecture 2.1). For the setting where the dimension d is a fixed constant, (Froese and Hertrich 2024) showed NP-hardness with input dimension $d = 2$. (Goel et al. 2021) Showed that for error margin parameter $\gamma = 0$ the problem is NP-Hard if and only if $k \geq 2$; also, in the paper they give lower bounds for approximation algorithms. For $k = 2$ and $\gamma = 0$ there is also an NP-Hardness result of (Boob, Dey, and Lan 2022).

Interestingly, even for two-layer networks with ReLU activation functions, the C-NNT problem is $\exists\mathbb{R}$-Complete (the *existential theory of the reals*) (Bertschinger et al. 2023; Abrahamsen, Kleist, and Miltzow 2021). This complexity class, originating in (Schaefer 2009), is conjectured to be distinct from NP.

On the algorithmic front, (Pilanci and Ergen 2020) showed a polynomial time algorithm for fixed dimensions with a regularized objective. There are also algorithms with exponential running time (as a function of the number of neurons and the output dimension) (Arora et al. 2018; Froese, Hertrich, and Niedermeier 2022a). These results interestingly show that under restrictions on the output dimension and depth of the network, the problem is in NP. For the study of linear activation functions, (Khalife, Cheng, and Basu 2024) give an algorithm polynomial in the dataset size, but exponential in the network size and the dimension.

Organization: In Section 2 we give some preliminary definitions and notations. In Section 3 we give our lower bounds for deep networks and in Section 4 our lower bounds for two-layer networks. Finally, Section 5 presents the pseudo-polynomial algorithm. Due to space constraints, some of the proofs are delegated to the full version of the paper.

2 Preliminaries

Notations Given a number $n \in \mathbb{N}$, let $\mathsf{poly}(n)$ be a polynomially bounded expression of n. Given a vector $x \in \mathbb{R}^d$, for all $i \in \{1, \ldots, d\}$ we use $x[i]$ to denote the i-th entry of x. Given an instance I of an optimization/decision problem, we use $|I|$ to denote the encoding size of I. As the notations of NNT are quite cumbersome, we use the notations $I = (N = (V, E), \mathcal{D}, (\sigma^v)_{v \in V \setminus \{s\}}, \mathcal{L}, \Theta = (W_e, B_e)_{e \in E}, \gamma)$ given in the formal definition in the introduction as the canonical notation for an NNT instance. We sometimes omit the specific declaration of some objects when clear from the context.

Complexity The complexity class NP consists of all decision problems for which a given solution (certificate) can be verified in polynomial time by a deterministic Turing ma-

chine. In addition, the complexity class P consists of all decision problems that can be decided by a deterministic Turing machine in polynomial time. *Bounded-error probabilistic polynomial time (BPP)* is the complexity class consisting of all decision problems decidable by a probabilistic Turing machine with success probability at least $\frac{2}{3}$ in polynomial time. It is known that P $\subseteq$ BPP and P $\subseteq$ NP; it is often conjectured that P = BPP and P $\neq$ NP .

A complexity conjecture used in Theorem 1.4 is the *Exponential-Time Hypothesis (ETH)* (Impagliazzo and Paturi 2001). This conjecture claims that there is no subexponential algorithm for the 3-SAT problem. Formally,

Conjecture 2.1. Exponential-Time Hypothesis (ETH) *There is a constant $\beta > 0$ such that there is no algorithm that given a* 3-SAT *formula Φ with n variables and m clauses can decide whether Φ is satisfiable in time $O\left(2^{\beta \cdot n}\right)$.*

3 Proof of Theorem 1.1

In this section, we give the proof of Theorem 1.1. The proof is based on a reduction from a decision problem involving *straight-line programs (SLP)*.

Definition 3.1. *A* straight-line program (SLP) *P is a sequence of univariate integer polynomials $(a_0, a_1, \ldots, a_\ell)$ such that $a_0 = 1$, $a_1 = x$ and $a_i = a_j \oplus a_k$ for all $2 \leq i \leq \ell$, where $\oplus \in \{+, -, *\}$ and $j, k < i$. We use $\tau(f)$ to denote the minimum length of an SLP that computes a univariate polynomial f.*

We use a conjecture proposed in (Dutta, Saxena, and Sinhababu 2022). A *radical* $\mathsf{rad}(f)$ of a non-zero integer polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$ is the product of the irreducible integer polynomials dividing f.

Conjecture 3.2. Constructive univariate radical conjecture: *For any polynomial $f \in \mathbb{Z}[x]$, we have $\tau(\mathsf{rad}(f)) \leq \mathsf{poly}(\mathsf{rad}(f))$. Moreover, there is a randomized polynomial time algorithm which, given an* SLP *of size ℓ computing a polynomial f, constructs an* SLP *for $rad(f)$ of size $\mathsf{poly}(s)$ with success probability at least $1 - \frac{1}{\Omega(s^{1+\varepsilon})}$ for some $\varepsilon > 0$.*

Note that integers are a special case of univariate polynomials. Hence, integers can be computed via SLPs. This leads to the following problem originating in (Allender et al. 2009).

Problem 3.3. PosSLP: *Given an* SLP *P computing an integer n_P, decide if $n_P > 0$.*

In a PosSLP instance, we are given only the compact representation of the SLP. Thus, the variables are not explicitly computed but are defined inductively over the definition of previously defined variables. We will use a strong lower bound on PosSLP obtained by (Bürgisser and Jindal 2024).

Theorem 3.4. *(Bürgisser and Jindal 2024): If Conjecture 3.2 is true and* PosSLP $\in$ BPP*, then* NP $\subseteq$ BPP*.*

We can now prove Theorem 1.1. The crux is to use a neural network to efficiently compute the SLP variables, even though the weight space has to be calculated a priori. In the proof, we create 2 neurons for computing each operation of the SLP. The depth of the neural network will be $\Theta(\ell)$. The parameter space and activation function are efficiently encoded such that the output of each layer is the i-th component of the given PosSLP instance. Importantly, the reduced D-NNT instance is *restricted*: there is exactly one option for each parameter.

Lemma 3.5. *There is a polynomial time reduction that given a* PosSLP *P computes a* restricted D-NNT *instance I such that P is a* YES *instance if and only if I is a* YES *instance.*

Proof. Let $P = (a_1, \ldots, a_\ell)$ be a PosSLP instance. Since in PosSLP we compute an integer, only a special case of a univariate polynomial, we may assume without the loss of generality that we are given an SLP formula P such that $a_0 = 1$ and $a_i = a_j \oplus a_k$ for all $1 \leq i \leq \ell$, where $\oplus \in \{+, -, *\}$ and $j, k < i$. Based on P, define a D-NNT instance I as follows. Define two neurons h_i, h'_i for each $0 \leq i \leq \ell$, where $s = h_0$ is the input vertex. Also, define the output vertex as $t = h_\ell$ and let V be the set of vertices of the network. For the following, fix some $1 \leq i \leq \ell$ and $j, k < i$ such that $a_i = a_j \oplus a_k$.

Define the edge $e_{i,j} = (h_j, h_i)$. Moreover, define the edge $e_{i,k}$ as follows. If $\oplus \in \{+, -\}$, define $e_{i,k} = (h_k, h_i)$; otherwise (if $\oplus = *$), define $e_{i,k} = (h'_k, h_i)$. Finally, define the edge $e'_{r,r} = (h_r, h'_r)$ for all $0 \leq r \leq \ell$. Let E be the entire set of edges; so far we have defined the network $N = (V, E)$. Define the parameter space $\Theta = (W_e, B_e)_{e \in E}$ as follows. Let $B_e = \{0\}$ for all edges $e \in E$ (i.e., the bias of any edge must be zero) and define the weights as follows. Define $W_{e_{i,j}} = \{1\}$, set $W_{e_{i,k}} = \{1\}$ if $\oplus \in \{+, *\}$, and $W_{e_{i,k}} = \{-1\}$ if $\oplus = -$. Finally, let $W_{e'_{r,r}} = \{1\}$ for all $0 \leq r \leq \ell$. Note that the above weights induce a restricted D-NNT instance.

The activation function $\sigma^{h_i} : \mathbb{R} \to \mathbb{R}$ is defined by the following cases. If $\oplus \in \{+, -\}$, define $\sigma^{h_i}(\alpha) = \alpha$ for all $\alpha \in \mathbb{R}$. If $\oplus = *$, then consider the following construction before the definition of σ^{h_i}. For any $\alpha \in \mathbb{R}$, let β, λ be the unique decomposition of α into an integer $\beta \in \mathbb{Z}$ and $\lambda \in [0, 1)$ such that $\alpha = \beta + \lambda$; then, define $\sigma^{h_i}(\alpha) = \beta \cdot \lambda \cdot 10^{\mathsf{len}(\lambda)}$, where $\mathsf{len}(\lambda)$ is the number of decimal digits in λ after the decimal point. This function will be applied only to numbers for which λ can be represented using a finite number of decimal digits. For example, if $\alpha = 2.55$ then $\beta = 2$, $\lambda = 0.55$, and $\mathsf{len}(\lambda) = 2$; therefore, $\sigma^{h_i}(\alpha) = 2 \cdot 0.55 \cdot 10^2 = 110$.

Define $\sigma^{h'_i} : \mathbb{R} \to \mathbb{R}$ such that $\sigma^{h'_i}(\alpha) = \alpha \cdot 10^{-\mathsf{num}(\alpha)}$, where $\mathsf{num}(\alpha)$ is the total number of digits in α before the decimal point. This function will be applied only to integers. Note that $\sigma^{h'_i}$ converts an integer to a number between 0 and 1, for example, $\sigma^{h'_i}(138) = 0.138$. Note that all activation functions can be computed in polynomial time. Define a simple dataset containing merely a single pair $\mathcal{D} = \{(x, y)\}$, where $x, y = 1$, i.e., both the input and output are one-dimensional. Let $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be the loss function where $\mathcal{L}(\alpha, y) = 0$ if $\alpha = 1 = y$, else $\mathcal{L}(\alpha, y) = 1$ if $\alpha > 0$, and otherwise $\mathcal{L}(\alpha, y) = 2$. We finally set $\gamma = 1$ as the error margin parameter. The reduction can be computed in time $\mathsf{poly}(\ell)$: there are $O(\ell)$ vertices and edges, the weights are bounded, while the activation function, dataset, as well the

loss function, can be computed and encoded in polynomial time and space, respectively. Therefore, the correctness of the lemma follows by showing that for each $0 \leq i \leq \ell$ it holds that

$$z^{h_i}(x) = \sigma^{h_i}\left(\sum_{e=(u,h_i)\in E} w^e \cdot z^u(x) + b^e\right) = a_i. \quad (1)$$

We prove (1) by induction on i. For the base case, let $i = 0$. Then, by definition $z^{h_i}(x) = z^s(x) = x = 1 = a_0 = a_i$. Now, let $1 \leq i \leq \ell$ and assume that for all $0 \leq r \leq i-1$ it holds that $z^{h_r}(x) = a_r$. By the definition of a_i, there are $0 \leq j, k \leq i-1$ such that $a_i = a_j \oplus a_k$ for $\oplus \in \{+, -, *\}$. Then, by the induction hypothesis, it holds that $z^{h_j}(x) = a_j$ and $z^{h_k}(x) = a_k$. We consider three cases depending on $\oplus$.

- $\oplus = +$. Then, it holds that $z^{h_i}(x) = 1 \cdot z^{h_j}(x) + 1 \cdot z^{h_k}(x) = a_j \oplus a_k = a_i$.
- $\oplus = -$. Then, we have $z^{h_i}(x) = 1 \cdot z^{h_j}(x) - 1 \cdot z^{h_k}(x) = a_j \oplus a_k = a_i$.
- $\oplus = *$. Then, recall that $z^{h_k}(x) = a_k$; thus, $z^{h'_k}(x) = \sigma^{h'_k}(z^{h_k}(x)) = a_k \cdot 10^{-\mathsf{num}(a_k)}$. Note that a_k is an integer (this can be easily proven by induction); thus, $z^{h'_k}(x) \in [0, 1)$ and $z^{h'_k}(x)$ can be represented using a finite number of decimal digits. Therefore, it holds that $\sigma^{h_i}\left(z^{h_j}(x) + z^{h'_k}(x)\right) = a_j \cdot a_k \cdot 10^{-\mathsf{num}(a_k)} \cdot 10^{\mathsf{len}\left(a_k \cdot 10^{-\mathsf{num}(a_k)}\right)} = a_j \cdot a_k = a_i$. The second equality holds by the following. If $a_k = 0$, the equality is immediate; otherwise, since a_k is an integer, the number of digits after the decimal point of $a_k \cdot 10^{-\mathsf{num}(a_k)}$, that is $\mathsf{len}\left(a_k \cdot 10^{-\mathsf{num}(a_k)}\right)$, is exactly $\mathsf{num}(a_k)$.

By the above, the network output is a_ℓ. By the definition of the loss function, there are $\theta = (w_e, b_e)_{e\in E} \in \Theta$ such that $\mathcal{L}(f_\theta(x), y) \leq \gamma$ if and only if $a_\ell = n_P > 0$. This gives the statement of the lemma. □

Using the above lemma, we state and prove a lower bound for restricted D-NNT.

Lemma 3.6. *Assume that Conjecture 3.2 is true and assume that* NP $\not\subseteq$ BPP. *Then,* restricted D-NNT *is not in* BPP.

Proof. Assume that Conjecture 3.2 is true and assume that NP $\not\subseteq$ BPP. Using Lemma 3.5, a BPP algorithm for restricted D-NNT implies a BPP algorithm for PosSLP. Hence, restricted D-NNT is not in BPP, or we would reach a contradiction to Theorem 3.4. □

From Lemma 3.6 and the definition of the complexity class NP, we can finally prove Theorem 1.1. The proof is given in the full version of the paper.

4 Lower Bounds for Two-Layer Networks

In this section, we give the high level ideas in the proofs of our lower bounds for D-NNT on two-layer networks. The full proofs are given in the full version of the paper.

We start by explaining the proof idea of Theorem 1.3. The proof is based on a simple reduction from the *subset sum* problem, known to be NP-Hard (e.g., (Kleinberg and Tardos 2006)). In the subset sum problem, we are given a collection $A = \{a_1, \ldots, a_n \in \mathbb{N}\}$ and a target number $T \in \mathbb{N}$. The goal is to decide if there is a subset $S \subseteq A$ such that $\sum_{a\in S} a = T$.

In the proof, we give a reduction from subset sum to D-NNT on a network with one hidden layer where each edge corresponds to a single item a_i from a subset sum instance. The weights correspond to either taking the item or discarding it. Finally, the loss function guarantees that the overall sum at the output of the network reaches exactly to the target value. The complete proof is given in the full version of the paper.

We now give the proof of Theorem 1.4. The proof is based on a reduction from *binary constraint satisfaction problem (2-CSP)*. The input to a 2-CSP instance is a tuple $\Gamma = (G, \Sigma, C)$ such that $G = (V, E)$ is a constraint graph, Σ is an alphabet, and $C = \{C_{(u,v)}\}_{(u,v)\in E}$ are constraints such that for all $(u, v) \in E$ it holds that $C_{(u,v)} \subseteq \Sigma \times \Sigma$ and $C_{(u,v)} \neq \emptyset$. An *assignment* to Γ is a function $\phi : V \to \Sigma$. An assignment ϕ is called *feasible* if for all $(u, v) \in E$ it holds that $(\phi(u), \phi(v)) \in C_{(u,v)}$. The goal is to decide if there is a feasible assignment. We will use the following result of (Marx 2007).

Theorem 4.1. *(Marx 2007) Assuming* ETH*, for any computable function f there is no algorithm that decides* 2-CSP *in time $f(k) \cdot N^{o\left(\frac{k}{\log k}\right)}$, where k is the number of constraints and N is the size of the alphabet.*

Using the above, we can now prove the theorem. In the proof, we reduce 2-CSP instance Γ to a D-NNT instance with a neuron for each variable and an output vertex for each constraint of Γ; thus, the width is in the worst case the number of constraints. We encode weights in the network such that a constraint e is satisfied if and only if the output corresponding to e is within a restricted range of values.

To prove Theorem 1.5, we use the classic *exact set cover* problem. In the exact set cover problem, we are given a universe $U = \{u_1, \ldots, u_n\}$ of elements, a collection of sets $\mathcal{S} \subseteq 2^U$, and a cardinality bound $K \in \mathbb{N}$. The goal is to decide if there are K sets $S_1, \ldots, S_K \in \mathcal{S}$ such that for all $u \in U$ there is *exactly* one set $S_i, 1 \leq i \leq K$, such that $u \in S_i$. In our reduction, we create a simple D-NNT instance with a single input, output, and neuron. Each data point (x_i, y_i) corresponds to an element in the exact set cover instance. In addition, we have an extra data point used to guarantee a selection of at most K sets. The dimensions correspond to sets, so the j-th entry in x_i is 1 if and only if element i is in dimension j. The weights of the edges are also binary and correspond to a selection of sets. The complete proof is given in the full version of the paper.

5 A Pseudo-Polynomial Algorithm

In this section, we describe a pseudo-polynomial algorithm for D-NNT on a two-layer network as described in Theorem 1.6. The proofs from this section are deferred to the full version of the paper. For the remainder of this section, fix a D-NNT instance $I = (N = (V = \{s\} \cup$

$H \cup \{t\}, E), \mathcal{D}, (\sigma^h)_{h\in H}, \mathcal{L}, \Theta = (W_e, B_e)_{e\in E}, \gamma)$ with one hidden layer H, one input vertex s, one output t, with activation functions only on the hidden layer.

Let d be the input dimension (with output dimension being one), let $H = \{h_1, \ldots, h_k\}$ be the neurons in the hidden layer, and let $n_0 = |\mathcal{D}|$ be the size of the dataset. Let $e_1, \ldots, e_k$ be the edges connecting the input vertex s to the k neurons. With a slight abuse of notation, for all $a \in \{1, \ldots, k\}$ let $W_{e_a}[d+1] = B_{e_a}$; in addition, for all $i \in \{1, \ldots, n_0\}$ let $x_i[d+1] = 1$ (these notations slightly simplify our claims). We remark that the network is not necessarily fully-connected and that the activation and loss functions can be arbitrary.

As we consider a pseudo-polynomial time algorithm, we may assume without the loss of generality that $W_e, B_e \subset \mathbb{N}$ for every $e \in E$ (otherwise, simply scale all numbers in the input). Let $W_{\max}$ be the largest number that can be computed via the network with or without applying the activation and loss functions. Moreover, let $M = d \cdot W_{\max} \cdot k$. Observe that M is pseudo-polynomial in the input size $|I|$.

We define a dynamic program DimDP with an entry for each number $1, \ldots, M$, representing the value obtained at the input to the activation function on each neuron $q \in \{1, \ldots, k\}$, for each data point $(x, y) \in \mathcal{D}$, and dimensions $1, \ldots, j$ in the q-th neuron. The entries describe selections of prefixes of the parameters that compute a specific vector of outputs, one output for each data point. For simplicity, we first describe a simplified version of the dynamic program, and of a second dynamic program that will be introduced later, returning only boolean values; we later expand these programs to return selected parameters themselves using simple backtracking.

Formally, define $\mathsf{DimDP} : \{0, \ldots, M\}^{n_0} \times \{1, \ldots, d+1\} \times \{1, \ldots, k\} \to \{\mathsf{true}, \mathsf{false}\}$ inductively as follows. Fix some $q \in \{1, \ldots, k\}$; the definition of the dynamic program is analogous to all such values of q. For each $m \in \{0, \ldots, M\}^{n_0}$, define $\mathsf{DimDP}[m, q, 1] = \mathsf{true}$ if and only if there is $w \in W_{e_q}[1]$ such that $w \cdot x_i[1] = m[i]$ for all $i \in \{1, \ldots, n_0\}$; otherwise, set $\mathsf{DimDP}[m, q, 1] = \mathsf{false}$. Then, for all $j \in \{2, \ldots, d+1\}$ and $m \in \{0, \ldots, M\}^{n_0}$ define $\mathsf{DimDP}[m, q, j] = \mathsf{true}$ if and only if there are $w \in W_{e_q}[j]$ and $m_1, m_2 \in \{0, \ldots, M\}^{n_0}$ such that $\mathsf{DimDP}[m_1, q, j-1] = \mathsf{true}$, $m_1 + m_2 = m$, and for all $i \in \{1, \ldots, n_0\}$ it holds that $w \cdot x_i[j] = m_2[i]$. The following result summarizes the correctness of DimDP.

Lemma 5.1. *For all $m \in \{0, \ldots, M\}^{n_0}$, $q \in \{1, \ldots, k\}$, and $j \in \{1 \ldots, d+1\}$ it holds that $\mathsf{DimDP}[m, q, j] = \mathsf{true}$ if and only if there are $\left(w[r] \in W_{e_q}[r]\right)_{r\in\{1,\ldots,j\}}$ such that for all $i \in \{1, \ldots, n_0\}$ it holds that $\sum_{r=1}^{j} x_i[r] \cdot w[r] = m[i]$.*

Once we have computed the above dynamic program DimDP, we compute a second dynamic program $\mathsf{FinalDP} : \{0, \ldots, M\}^{n_0} \times \{0, \ldots, k\} \to \{\mathsf{true}, \mathsf{false}\}$, where entry $\mathsf{FinalDP}[m, q]$ describes whether the total weight arriving to the output on input x_i only from neurons $h_1, \ldots, h_q$ is precisely $m[i]$. Formally, for all $m \in \{0, \ldots, M\}^{n_0}$ define $\mathsf{FinalDP}[m, 0] = \mathsf{true}$ if and only if $m = (0)_{i\in\{1,\ldots,n_0\}}$. For all $q \in \{1, \ldots, k\}$ and $m \in \{0, \ldots, M\}^{n_0}$ define $\mathsf{FinalDP}[m, q] = \mathsf{true}$ if and only if there are $m_1, m_2, m_3 \in \{0, \ldots, M\}^{n_0}$ such that $\mathsf{FinalDP}[m_1, q-1] = \mathsf{true}$, $m_1 + m_3 = m$, $\mathsf{DimDP}[m_2, q, d+1] = \mathsf{true}$, and $m_3 = \sigma^{h_q}(m_2)$. The following result summarizes the correctness of FinalDP.

Lemma 5.2. *For all $m \in \{0, \ldots, M\}^{n_0}$ and $q \in \{0, \ldots, k\}$ it holds that $\mathsf{FinalDP}[m, q] = \mathsf{true}$ if and only if there are $(w_{e_a}[r] \in W_{e_a}[r])_{r\in\{1,\ldots,d\}, a\in\{1,\ldots,q\}}$ and $(b_{e_a} \in B_{e_a})_{a\in\{1,\ldots,q\}}$ such that for all $i \in \{1, \ldots, n_0\}$ it holds that $\sum_{a=1}^{q} \sigma^{h_a}\left(\sum_{r=1}^{d} x_i[r] \cdot w_{e_a}[r] + b_{e_a}\right) = m[i]$.*

We slightly alter the above dynamic programs so that instead of only returning true or false, they also retrieve an actual set of parameters corresponding to the entry. Namely, for all $m \in \{0, \ldots, M\}^{n_0}$, $q \in \{1, \ldots, k\}$, and $j \in \{1, \ldots, d+1\}$ the dynamic program $\mathsf{DimDP}[m, q, j]$ keeps a set of parameters $\left(w[r] \in W_{e_q}[r]\right)_{r\in\{1,\ldots,j\}}$ such that for all $i \in \{1, \ldots, n_0\}$ it holds that $\sum_{r=1}^{j} x_i[r] \cdot w[r] = m[i]$ (recall that $x_i[d+1] = 1$). Similarly, for all $m \in \{0, \ldots, M\}^{n_0}$ and $q \in \{0, \ldots, k\}$ the dynamic program $\mathsf{FinalDP}[m, q]$ keeps a set of parameters $(w_{e_a}[r] \in W_{e_a}[r])_{r\in\{1,\ldots,d\}, a\in\{1,\ldots,q\}}$ and $(b_{e_a} \in B_{e_a})_{a\in\{1,\ldots,q\}}$ such that for all $i \in \{1, \ldots, n_0\}$ it holds that $\sum_{a=1}^{q} \sigma^{h_a}\left(\sum_{r=1}^{d} x_i[r] \cdot w_{e_a}[r] + b_{e_a}\right) = m[i]$. The parameters can be selected efficiently using standard backtracking.

We now define a model based on the above dynamic programs. Select a set of parameters $\theta \in \Theta$ such that $\sum_{i=1}^{n} \mathcal{L}(f_\theta(x_i), y_i)$ is minimized over all possible parameter sets corresponding to $\mathsf{FinalDP}[m, k]$, for all $m \in \{0, \ldots, M\}^{n_0}$ such that $\mathsf{FinalDP}[m, k] = \mathsf{true}$. Using the above, we obtain an algorithm deciding I by checking whether the training loss over the selected model surpasses the threshold. The pseudocode of the algorithm is given in Algorithm 1. The next lemma analyzes the running time and correctness of the algorithm.

Algorithm 1: Dynamic Programming Algorithm

Input: a D-NNT instance I
Output: true if I is a Yes-instance

1: Compute the dynamic program DimDP.
2: Compute the dynamic program FinalDP.
3: Compute the model f_θ based on DimDP and FinalDP.
4: **return true** if and only if $\sum_{i=1}^{n_0} \mathcal{L}(f_\theta(x_i), y_i) \leq \gamma$.

Lemma 5.3. *Algorithm 1 correctly decides the* D-NNT *instance and can be computed in time $M^{O(n_0)} \cdot d \cdot k$.*

The above directly implies the proof of the theorem.

Proof of Theorem 1: Follows from Lemma 5.3. □

References

Abrahamsen, M.; Kleist, L.; and Miltzow, T. 2021. Training Neural Networks is ER-complete. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 18293–18306. Curran Associates, Inc.

Allender, E.; Bürgisser, P.; Kjeldgaard-Pedersen, J.; and Miltersen, P. B. 2009. On the Complexity of Numerical Analysis. *SIAM J. Comput.*, 38(5): 1987–2006.

Amizadeh, S.; Matusevych, S.; and Weimer, M. 2019. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*.

Arora, R.; Basu, A.; Mianjy, P.; and Mukherjee, A. 2018. Understanding Deep Neural Networks with Rectified Linear Units. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Baldassi, C.; and Braunstein, A. 2015. A max-sum algorithm for training discrete neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2015(8): P08008.

Bertschinger, D.; Hertrich, C.; Jungeblut, P.; Miltzow, T.; and Weber, S. 2023. Training Fully Connected Neural Networks is ∃R-Complete. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Blum, A.; and Rivest, R. 1988. Training a 3-node neural network is NP-complete. *Advances in neural information processing systems*, 1.

Boob, D.; Dey, S. S.; and Lan, G. 2022. Complexity of training ReLU neural network. *Discret. Optim.*, 44(Part): 100620.

Bürgisser, P.; and Jindal, G. 2024. On the Hardness of PosSLP. In Woodruff, D. P., ed., *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, 1872–1886. SIAM.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28.

Dey, S. S.; Wang, G.; and Xie, Y. 2020. Approximation Algorithms for Training One-Node ReLU Neural Networks. *IEEE Trans. Signal Process.*, 68: 6696–6706.

Dinur, I.; and Steurer, D. 2014. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 624–633.

Dutta, P.; Saxena, N.; and Sinhababu, A. 2022. Discovering the Roots: Uniform Closure Results for Algebraic Classes Under Factoring. *J. ACM*, 69(3).

Froese, V.; and Hertrich, C. 2024. Training neural networks is np-hard in fixed dimension. *Advances in Neural Information Processing Systems*, 36.

Froese, V.; Hertrich, C.; and Niedermeier, R. 2022a. The computational complexity of ReLU network training parameterized by data dimensionality. *Journal of Artificial Intelligence Research*, 74: 1775–1790.

Froese, V.; Hertrich, C.; and Niedermeier, R. 2022b. The Computational Complexity of ReLU Network Training Parameterized by Data Dimensionality. *J. Artif. Intell. Res.*, 74: 1775–1790.

Goel, S.; Kanade, V.; Klivans, A. R.; and Thaler, J. 2017. Reliably Learning the ReLU in Polynomial Time. In Kale, S.; and Shamir, O., eds., *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, volume 65 of *Proceedings of Machine Learning Research*, 1004–1042. PMLR.

Goel, S.; Klivans, A. R.; Manurangsi, P.; and Reichman, D. 2021. Tight Hardness Results for Training Depth-2 ReLU Networks. In Lee, J. R., ed., *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Goodfellow, I. J.; Bengio, Y.; and Courville, A. C. 2016. *Deep Learning*. Adaptive computation and machine learning. MIT Press.

Hayer, A.; and Bhalla, U. S. 2005. Molecular switches at the synapse emerge from receptor and kinase traffic. *PLoS computational biology*, 1(2): e20.

Hochreiter, S. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02): 107–116.

Hu, W.; Xiao, L.; and Pennington, J. 2020. Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Impagliazzo, R.; and Paturi, R. 2001. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2): 367–375.

Judd, S. 1988. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4(3): 177–192.

Khalife, S.; Cheng, H.; and Basu, A. 2024. Neural networks with linear threshold activations: structure and algorithms. *Mathematical Programming*, 206(1): 333–356.

Kleinberg, J. M.; and Tardos, É. 2006. *Algorithm design*. Addison-Wesley. ISBN 978-0-321-37291-8.

Li, Q.; and Hao, S. 2018. An optimal control approach to deep learning and applications to discrete-weight neural networks. In *International Conference on Machine Learning*, 2985–2994. PMLR.

Lu, F.; Wang, L.-C.; Cheng, K.-T.; and Huang, R.-Y. 2003. A circuit SAT solver with signal correlation guided learning. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, 892–897. IEEE.

Marx, D. 2007. Can you beat treewidth? In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, 169–179. IEEE.

Megiddo, N. 1988. On the complexity of polyhedral separability. *Discrete & Computational Geometry*, 3: 325–337.

Miller, P.; Zhabotinsky, A. M.; Lisman, J. E.; and Wang, X.-J. 2005. The stability of a stochastic CaMKII switch: dependence on the number of enzyme molecules and protein turnover. *PLoS biology*, 3(4): e107.

Pilanci, M.; and Ergen, T. 2020. Neural Networks are Convex Regularizers: Exact Polynomial-time Convex Optimization Formulations for Two-layer Networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 7695–7705. PMLR.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542. Springer.

Raz, R.; and Safra, S. 1997. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 475–484.

Rokh, B.; Azarpeyvand, A.; and Khanteymoori, A. 2023. A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Transactions on Intelligent Systems and Technology*, 14(6): 1–50.

Schaefer, M. 2009. Complexity of Some Geometric and Topological Problems. In Eppstein, D.; and Gansner, E. R., eds., *Graph Drawing, 17th International Symposium, GD 2009, Chicago, IL, USA, September 22-25, 2009. Revised Papers*, volume 5849 of *Lecture Notes in Computer Science*, 334–344. Springer.

Shalev-Shwartz, S.; and Ben-David, S. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Sun, Y.; Lao, D.; Sundaramoorthi, G.; and Yezzi, A. 2022. Surprising instabilities in training deep networks and a theoretical analysis. *Advances in Neural Information Processing Systems*, 35: 19567–19578.

Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; and Vinyals, O. 2021. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3): 107–115.

Zheng, S.; Song, Y.; Leung, T.; and Goodfellow, I. 2016. Improving the robustness of deep neural networks via stability training. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 4480–4488.

Zhou, Y.; Moosavi-Dezfooli, S.-M.; Cheung, N.-M.; and Frossard, P. 2018. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Zhu, C.; Han, S.; Mao, H.; and Dally, W. J. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*.