THEORY OF COMPUTING, Volume 22 (6), 2026, pp. 1–48
www.theoryofcomputing.org

Training Fully Connected Neural Networks is ∃ℝ-Complete

Daniel Bertschinger Christoph Hertrich* Paul Jungeblut
Tillmann Miltzow† Simon Weber‡

Received June 10, 2024; Revised January 30, 2026; Published September 16, 2026

Abstract. We consider the problem of finding weights and biases for a two-layer fully connected neural network to fit a given set of data points as well as possible, also known as EMPIRICALRISKMINIMIZATION. Our main result is that the associated decision problem is ∃ℝ-complete, that is, polynomial-time equivalent to determining whether a multivariate polynomial with integer coefficients has any real roots. Furthermore, we prove that algebraic numbers of arbitrarily large degree are required as weights to be able to train some instances to optimality, even if all data points are rational. Our result already applies to fully connected instances with two inputs, two outputs, and one hidden layer of ReLU neurons. Thereby, we strengthen a result by Abrahamsen,

An extended abstract of this paper appeared in Advances in Neural Information Processing Systems 36 (NeurIPS 2023).

*Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreements ScaleOpt–757481 and ForEFront–615640)

†Supported by the Netherlands Organisation for Scientific Research (NWO) under project numbers 016.Veni.192.250 and VI.Vidi.213.150.

‡Supported by the Swiss National Science Foundation under project no. 204320.

ACM Classification: Theory of computation: Problems, reductions and completeness; Theory of computation: Machine learning theory; Computing methodologies: Neural networks

AMS Classification: 68Q17, 68T07

Key words and phrases: neural network training, computational complexity, existential theory of the reals, algebraic universality, empirical risk minimization

 DOI: 10.4086/toc.2026.v022a006

Kleist, and Miltzow (NeurIPS'21). A consequence of this is that a combinatorial search algorithm like the one by Arora, Basu, Mianjy, and Mukherjee (ICLR'18) is impossible for networks with more than one output dimension, unless NP = ∃ℝ.

1 Introduction

The use of neural networks in modern computer science is ubiquitous. They are arguably the most powerful tool at our disposal in machine learning [43]. One of the most fundamental algorithmic problems associated with neural networks is to train a neural network given some training data. This is an optimization problem with the goal of minimizing a given loss function. In this paper we study the associated decision problem of determining whether there exist network parameters such that the loss does not exceed a given threshold. For arbitrary network architectures, Abrahamsen et al. [3] showed that this decision problem is ∃ℝ-complete already for two-layer neural networks and linear activation functions. The complexity class ∃ℝ is defined as the family of decision problems that are polynomial-time many-one reducible to determining whether a multivariate polynomial with integer coefficients has a real root. Under the commonly believed assumption that ∃ℝ is a strict superset of NP, this implies that training a neural network is harder than NP-complete problems.

The result of Abrahamsen et al. [3] has one major downside, namely that the network architecture is *adversarial*: the hardness inherently relies on choosing a network architecture that is particularly difficult to train. Their instances could be easily trained if the networks were fully connected. This stems from the fact that they use the identity function as the activation function, which reduces the problem to matrix factorization. While intricate network architectures, e. g., convolutional and residual neural networks, pooling, autoencoders and generative adversarial neural networks are common in practice, they are usually designed in a way that facilitates training rather than making it difficult [43]. We strengthen the result in [3] by showing ∃ℝ-completeness for *fully connected* two-layer neural networks. This shows that ∃ℝ-hardness does not stem from one specifically chosen worst-case architecture, but is inherent in the neural network training problem itself. Although a host of different architectures are used in practice, fully connected two-layer neural networks are arguably the most basic ones, and they are often part of more complicated network architectures [43]. We show hardness even for the case of fully connected two-layer ReLU neural networks with exactly two input and output dimensions.

Remarkably, with only one instead of two output dimensions, the decision problem is in NP. This follows from the combinatorial search algorithm by Arora et al. [6]: the activation patterns of the hidden neurons on the training data together with the signs of the output weights can serve as an NP certificate, since after fixing these discrete choices, the remaining convex optimization problem can be solved in polynomial time.

Our result explains why their algorithm was never successfully generalized to more complex network architectures: adding only a second output neuron significantly increases the computational complexity of the problem, from being contained in NP to being ∃ℝ-complete.

To achieve this result, our reduction follows a new approach compared to the reduction in [3]. Instead of encoding polynomial inequalities into an adversarial network architecture, we make use of the underlying geometry of the functions computed by two-layer neural networks and utilize the fact that their different output dimensions have nonlinear dependencies.

1.1 Outline

We start by formally introducing neural networks, the training problem, and the existential theory of the reals in Section 2. Thereafter, we present our main results in Section 3. In Section 4 we provide different perspectives on how to interpret our findings, including an in-depth discussion of the strengths and limitations. We cover further related work in Section 5. We present the key ideas we use to prove ∃ℝ-hardness in Section 6. Finally, the complete proof details for ∃ℝ-containment, ∃ℝ-hardness, and algebraic universality are contained in Sections 7 to 9, respectively.

2 Preliminaries

2.1 Neural networks

All neural networks considered in this paper have a very simple architecture. For ease of presentation, we do not define neural networks and their training problem in full generality here, but restrict ourselves to the simple architectures we need.

Definition 2.1. A *fully connected two-layer neural network* $N = (S \mathbin{\dot\cup} H \mathbin{\dot\cup} T, E)$ is a directed acyclic graph (the *architecture*) with real-valued edge weights. The vertices are called *neurons*. The set of neurons is partitioned into the set S of *inputs*, the set H of *hidden neurons*, and the set T of *outputs*. All possible edges from S to H, as well as all possible edges from H to T are present. Additionally, each hidden neuron has a real-valued *bias* and an *activation function* ($\mathbb{R} \to \mathbb{R}$).

The probably most commonly used activation function [6, 38, 43] is the *rectified linear unit (ReLU)* defined as

$$\begin{aligned} \mathrm{ReLU}\colon \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \max\{0, x\}. \end{aligned}$$

See Figure 1 for a small fully connected two-layer ReLU neural network.

A neural network architecture N as defined above realizes a function $f(\cdot, \Theta)\colon \mathbb{R}^S \to \mathbb{R}^T$, where Θ denotes assignment of the weights and biases that parameterize the function. For $x \in \mathbb{R}^S$, we define $f(\cdot, \Theta)$ inductively. Each input neuron forwards its component of x to all its outgoing neighbors. Each hidden neuron computes the weighted sum over all incoming values, adds its bias, applies its activation function to this sum, and forwards it to all outgoing neighbors. An output neuron also computes the weighted sum over all incoming values, but it neither adds a bias nor applies any activation function.

For our purposes, we define the following special case of EMPIRICALRISKMINIMIZATION, i. e., a restriction of the general neural network training problem.

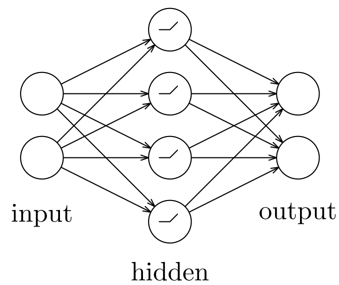


Figure 1: A fully connected two-layer neural network as studied in this paper. The symbol inside the hidden neurons expresses the ReLU activation function.

Definition 2.2. Training a fully connected two-layer neural network is the following decision problem, denoted by TRAIN-F2NN.

Input: A 5-tuple $(N, \varphi, D, \gamma, c)$:

- $N = (S \mathbin{\dot\cup} H \mathbin{\dot\cup} T, E)$ is the fully connected two-layer network architecture.
- $\varphi \colon \mathbb{R} \to \mathbb{R}$ is the activation function for all hidden neurons, computable in polynomial time on a real RAM.
- $D \subseteq \mathbb{Q}^S \times \mathbb{Q}^T$ is the *training data*, i. e., a set of n *data points* of the form $(x; y)$. Here, y is called the *label* of the data point.
- $\gamma \in \mathbb{Q}_{\geq 0}$ is the *target error*.
- $c \colon \mathbb{R}^T \times \mathbb{R}^T \to \mathbb{R}_{\geq 0}$ is an honest[1] *loss function*, computable in polynomial time on a real RAM.

Question: Is there an assignment Θ of weights and biases such that $\sum_{(x;y)\in D} c\big(f(x,\Theta), y\big) \leq \gamma$?

Our $\exists\mathbb{R}$-hardness proof works for any fixed honest loss function, since our reduction only constructs instances with $\gamma = 0$ (exact fit). On the other hand, our $\exists\mathbb{R}$-membership proof only works for activation and loss functions that are polynomial-time computable on a real RAM (see Section 7 for the necessary details).

2.2 Existential theory of the reals

The complexity class $\exists\mathbb{R}$ has gained a lot of interest in recent years. It is defined via its canonical complete problem ETR (short for *Existential Theory of the Reals*) and contains all problems that polynomial-time many-one reduce to it. An ETR instance consists of an integer n and a sentence of the form

$$\exists X_1, \ldots, X_n \in \mathbb{R} : \varphi(X_1, \ldots, X_n),$$

where φ is a well-formed and quantifier-free formula consisting of polynomial equations and inequalities in the variables with integer coefficients encoded in binary, and the logical connectives $\{\wedge, \vee, \neg\}$. The goal is to decide whether this sentence is true. As an example,

[1]A loss function is called *honest* if it returns zero if and only if the data is fit exactly.

consider the formula $\varphi(X, Y) :\equiv X^2 + Y^2 \leq 1 \wedge Y^2 \geq 2X^2 - 1$; among (infinitely many) other solutions, $\varphi(0, 0)$ evaluates to true, witnessing that this is a yes-instance of ETR. It is known that

$$\mathsf{NP} \subseteq \exists\mathbb{R} \subseteq \mathsf{PSPACE},$$

and it is widely believed that both inclusions are strict. Here, the first inclusion follows because a SAT instance can easily be expressed as an equivalent ETR instance [86]. The second inclusion was proved by Canny [20].

Note that the complexity of problems involving real numbers has been studied in various contexts. To avoid confusion, let us emphasize that the underlying machine model for $\exists\mathbb{R}$ (over which sentences need to be decided and where reductions are performed) is the standard binary word RAM (or equivalently, a Turing machine), just like for P, NP, and PSPACE. It is *not* the real RAM [34] or the Blum-Shub-Smale model [14].

3 Results

Our main result is the following theorem establishing $\exists\mathbb{R}$-completeness of TRAIN-F2NN even in very restricted cases.

Theorem 3.1. TRAIN-F2NN *is* $\exists\mathbb{R}$*-complete, even if*

- *there are only two input neurons,*
- *there are only two output neurons,*
- *the number of data points is linear in the number of hidden neurons,*
- *the data has only* 13 *different labels,*
- *the target error is* $\gamma = 0$ *(exact fit), and*
- *the* ReLU *activation function is used.*

Let us note that the combination of all restrictions in Theorem 3.1 describes a special case of TRAIN-F2NN. Proving this special case $\exists\mathbb{R}$-hard also implies $\exists\mathbb{R}$-hardness of the general version of TRAIN-F2NN, and even its general case EMPIRICALRISKMINIMIZATION. For example, $\exists\mathbb{R}$-hardness also holds for more input/output neurons, more data points, more labels, arbitrary $\gamma \geq 0$, and other activation functions.

Additionally, our $\exists\mathbb{R}$-hardness reduction implies *algebraic universality*.

Theorem 3.2. *Let* $\alpha \in \mathbb{R}$ *be an algebraic number. Then there exists an instance of* TRAIN-F2NN *fulfilling all the restrictions in Theorem 3.1, which has a solution* Θ *with weights and biases from* $\mathbb{Q}[\alpha]$*, but no solution when the weights and biases are restricted to a field* $\mathbb{F}$ *not containing* α*.*

Here, $\mathbb{Q}[\alpha]$ is the smallest field extension of $\mathbb{Q}$ containing α. This means that there are training instances for which all global optima require irrational weights or biases, even if all data points are integral.

Let us note that algebraic universality and $\exists\mathbb{R}$-completeness are independent concepts. While algebraic universality is known to hold for various $\exists\mathbb{R}$-complete problems [4], this is not automatic: it cannot occur in problems where the solution space is open. On the other hand, reductions to prove algebraic universality do not need to be in polynomial time, i. e., algebraic universality can be established without proving $\exists\mathbb{R}$-hardness at the same time.

4 Discussion

There is already a wide body of literature about the computational hardness of EMPIRICAL-RISKMINIMIZATION, see Section 5 below. In order to clarify our contribution, we use this section to discuss our results from various perspectives, pointing out strengths and limitations.

4.1 Implications of $\exists\mathbb{R}$-completeness

Our definition of $\exists\mathbb{R}$ relies on ETR as a canonical complete problem. While ETR is known to be NP-hard and in PSPACE [20,86], its precise computational complexity is still unknown (and it is considered likely that it is neither NP- nor PSPACE-complete). Therefore, proving a problem to be $\exists\mathbb{R}$-complete only tells us its difficulty relative to ETR. Nevertheless, proving $\exists\mathbb{R}$-completeness is a valuable contribution, even for problems that are known to be NP-hard, because of the following implications.

As pointed out by Schaefer [75], $\exists\mathbb{R}$-completeness of a problem shifts the focus away from the problem itself to its underlying algebraic nature: "Knowing that a problem is $\exists\mathbb{R}$-complete does not tell us more than that it is NP-hard and in PSPACE in terms of classical complexity, but it does tell us where to start the attack: [...] A solution will likely not come out of graph drawing or graph theory but out of a better understanding of real algebraic geometry and logic." [75]

Knowing that a problem is $\exists\mathbb{R}$-complete also hints towards the algorithmic challenges that need to be overcome. While many NP-complete problems can be solved well in practice by extremely optimized off-the-shelf SAT- or MIP-solvers, no such general-purpose tools are available for $\exists\mathbb{R}$-complete problems. In fact, to the best of our knowledge, deciding the solvability of an $\exists\mathbb{R}$-complete problem (and finding the optimum of a related optimization problem) requires algorithms from real algebraic geometry, for example a decision procedure for existentially quantified first-order sentences (after reducing the problem to ETR). However, these algorithms are very inefficient (at least exponential in the number of variables) and therefore infeasible for large instances [70].

However, let us stress that $\exists\mathbb{R}$-completeness does not rule out hope for good heuristics: For the optimization version of the $\exists\mathbb{R}$-complete Art gallery problem (minimizing the number of guards), one can often find near-optimal solutions efficiently in practice using custom heuristics, some of which are also used in combination with integer programming solvers [27]. Under additional assumptions, performance guarantees can be proven [49]. However, these heuristics are specifically tailored towards the Art gallery problem. Identifying reasonable assumptions for other $\exists\mathbb{R}$-complete problems in order to obtain good heuristics (possibly even with performance guarantees) is an important open question.

One meta-heuristic that can often be used to obtain "good" solutions for $\exists\mathbb{R}$-complete problems is gradient descent. A prominent example in our context is training neural networks, where a number of different gradient descent variants powered by backpropagation are nowadays capable of training neural networks containing millions of neurons efficiently and with practically satisfying quality. In general, we do not get any guarantee on the solution quality or running time when using gradient descent, but under the right additional assumptions proving such guarantees is sometimes possible [17]. Gradient descent has also been applied to

∃ℝ-complete problems from other areas, for example *graph drawing* [5]. Still, these gradient descent approaches are tailored towards the specific problem at hand. It would be desirable to have general-purpose solvers, similar to SAT- or MIP-solvers for problems in NP.

4.2 Relation to learning theory

In this paper, we focus on the computational complexity of EMPIRICALRISKMINIMIZATION, that is, minimizing the *training error*. In the machine learning practice, one usually desires to achieve low *generalization error*, which means to use the training data to achieve good predictions on unseen test samples.

To formalize the concept of the generalization error, one needs to combine the computational aspect with a statistical one. There are various models to do so in the literature, the most famous one being *probably approximately correct* (PAC) learnability [84,90]. While EMPIRICALRISKMINIMIZATION and learnability are two different questions, they are strongly intertwined; see Section 5 for related work in this context. Despite the close connections between EMPIRICALRISKMINIMIZATION and learnability, to the best of our knowledge, the ∃ℝ-hardness of the former has no direct implications on the complexity of the latter. Still, since EMPIRICALRISKMINIMIZATION is the most common learning paradigm in practice, our work is arguably also interesting in the context of learning theory.

4.3 Required precision of computation

An implication of ∃ℝ-hardness of TRAIN-F2NN is that for some instances every set of weights and biases exactly fitting the data needs large precision, actually an exponential number of bits to be written down, as, for example, iterated squaring can be encoded into an ETR formula. The algebraic universality of TRAIN-F2NN (Theorem 3.2) strengthens this by showing that exact solutions require algebraic numbers. This restricts the techniques one could use to obtain optimal weights and biases even further, as it rules out numerical approaches (even using arbitrary-precision arithmetic), and shows that symbolic computation is required. In practice, we are often willing to accept small additive errors when computing $f(\cdot,\Theta)$, and therefore do not require Θ to be of such high precision. Rounding the values of Θ (the weights and biases) to the first "few" digits after the binary point may be sufficient. We leave it as an open question to find a suitable model that captures such a discrete version of the training problem and to study the computational complexity of the training problem in that model. In this context, let us point to the following phenomenon that appears with many ∃ℝ-complete problems [34]. While an exact solution x requires high precision, there is an approximate solution $\tilde{x}$ close to x that needs only polynomial precision. However, it is a priori not clear how to exploit this property to obtain useful complexity results or to develop practical heuristics. Moreover, historically, ∃ℝ-completeness seems to be a strong predictor that finding these approximate solutions is difficult in practice [28,34].

Bienstock et al. [11] use the above idea to discretize the weights and biases to show that, in principle, arbitrary neural network architectures can be trained to approximate optimality via linear programs whose size is linear in the size of the data set, but exponential in the architecture size.

4.4 Number of input neurons

In practice, neural networks are often trained on high-dimensional data, thus having only two input neurons is even more restrictive than the practical setting. Note that we easily obtain hardness for higher input dimensions by simply placing all data points of our reduction into a two-dimensional subspace. The precise complexity of training fully connected two-layer neural networks with only one-dimensional input and multi-dimensional output remains unknown. While this setting does not have practical relevance, we are still curious about this open question from a purely mathematical perspective.

4.5 Number of output neurons

If there is only one output neuron instead of two, then the problem is known to be NP-complete as long as there are at least two input neurons [6, 35]. Our reduction can easily be extended to the case with more than two output neurons by padding all output vectors with zeros. Thus, the complexity classification is complete with respect to the number of output neurons.

4.6 Number of hidden neurons

Consider a situation where the number m of hidden neurons is larger than the number n of data points. If there are no two contradicting data points $(x_1; y_1)$ and $(x_2; y_2)$ with $x_1 = x_2$ but $y_1 \neq y_2$, then we can always fit all data points exactly [94]. Thus, we need at least a linear number of data points in terms of m for $\exists\mathbb{R}$-hardness. Our result is (asymptotically) tight in this aspect. Note that by adding additional data points, the ratio between n and m can be made arbitrarily large. Thus, our reduction holds also for all settings in which m is (asymptotically) much smaller than n.

4.7 Number of output labels

The number of labels used in our reduction is just 13. Requiring only a small constant number of different labels shows the relevance of our result to practice, where the number of data points often largely exceeds the number of labels, for instance, in classification tasks.

If all labels are contained in a one-dimensional affine subspace, then the problem is in NP, as they can be projected down to one-dimensional labels and the problem can be solved with the algorithm by Arora et al. [6]. As any two labels span a one-dimensional affine subspace, the problem can only be $\exists\mathbb{R}$-hard for at least three affinely independent output labels.

We think it is not particularly interesting to close the gap between 3 and 13 output labels, but it would be interesting to investigate the complexity of the problem when output labels have more structure. For example, in classification tasks one often uses *one-hot encodings*, where the output dimension equals the number of classes and all labels have the form $(0, \ldots, 0, 1, 0, \ldots, 0)$. Note that at least three output dimensions are needed in this case to obtain three different labels.

4.8 Target error

For simplicity, we only prove hardness for the case with target error $\gamma = 0$. However, it is generally not required to fit the data exactly in real-world applications. It is not too difficult to see that hardness can be proved analogously for larger values of γ: simply use the same reduction and add several inconsistent data points that necessarily incur an error of at least γ. The precise choice of these inconsistent data points heavily depends on the loss function. In conclusion, for different values of γ, we can still construct instances that prove $\exists\mathbb{R}$-completeness of the decision problem.

4.9 Activation function

The ReLU activation function is currently the most commonly used activation function [6, 38, 43]. Our methods are adaptable to other piecewise linear activation functions, such as *leaky* ReLUs. Having said that, our methods are *not* applicable to other types of activation functions, such as *Sigmoid*, *soft* ReLU or step functions. We want to point out that TRAIN-F2NN (and even EMPIRICALRISKMINIMIZATION) is in NP if a step function is used as the activation function [57]. Concerning the Sigmoid and soft ReLU functions, it is not even clear whether the EMPIRICALRISKMINIMIZATION is decidable, as trigonometric functions and exponential functions are not computable on the real RAM [34,72]. While smoother activation functions might support the success of heuristics like gradient descent, we would intuitively assume that finding exact global optima is even more difficult with smooth activation functions.

4.10 Other architectures

We consider fully connected two-layer networks as the most important base case, but we are also interested in complexity results for other network architectures. Specifically, deeper fully connected neural networks and convolutional neural networks are interesting. We find it hard to imagine that the worst-case complexity of EMPIRICALRISKMINIMIZATION becomes easier for more complicated architectures, but it remains an important open problem to prove or disprove this. In particular, $\exists\mathbb{R}$-completeness for deeper networks would strengthen our result and show that $\exists\mathbb{R}$-completeness is a robust phenomenon, in other words, independent of a choice of a specific network type.

4.11 Lipschitz continuity

The set of data points created in the proof of Theorem 3.1 is intuitively very tame. Formally, this is captured by proving that for all yes-instances constructed by our reduction there exists a solution Θ such that $f(\cdot, \Theta)$ is Lipschitz continuous for a small Lipschitz constants L. Lipschitz continuity is also related to *overfitting* and *regularization* [44], the purpose of the latter being to prefer simpler functions over more complicated ones. Being Lipschitz continuous with a small Lipschitz constant essentially means that the function is relatively flat. It is particularly remarkable that we can show hardness even for small Lipschitz constants, since Lipschitz continuity has been a crucial assumption in several recent results about training and learning ReLU networks, for example in [11, 18, 23, 39].

5 Related work

5.1 Complexity of neural network training

It is well known that minimizing the training error of a neural network is a computationally difficult problem for a large variety of activation functions and architectures [84].

Closest to our work is the recent $\exists\mathbb{R}$-completeness result by Abrahamsen et al. [3] for two-layer neural networks. In contrast to our work, they use the identity activation function and rely on particularly difficult-to-train architectures, both qualities being uncommon in practice. Zhang [96] sketched a similar result already in 1992: Training neural networks with *real-valued* data points is $\mathsf{NP}_{\mathbb{R}}$-complete, again with the identity activation function and with an adversarial architecture. $\mathsf{NP}_{\mathbb{R}}$ is a complexity class in the BSS-model of computation [14], but a suitable discretization (considering the so-called *constant-free Boolean part*) yields $\exists\mathbb{R}$-completeness in today's language.

For ReLU networks, NP-hardness, parameterized hardness and inapproximability results have been established even for the simplest possible architecture consisting of only a single ReLU neuron [15, 30, 36, 41]. While all these results require non-constant input-dimension, Froese and Hertrich [35] show that it is still NP-hard to train a two-layer ReLU network with two input neurons and one output neuron. On the positive side, the seminal algorithm by Arora et al. [6] solves EmpiricalRiskMinimization for two-layer ReLU networks with one output neuron to global optimality, placing the problem in NP. It was later extended to a more general class of loss functions by Froese et al. [36]. The running time is exponential in the number of neurons in the hidden layer and in the input dimension, but polynomial in the number of data points if the former two parameters are considered to be constant. This NP-containment of EmpiricalRiskMinimization with one-dimensional output is in sharp contrast to our $\exists\mathbb{R}$-completeness result for Train-F2NN with two-dimensional outputs. Brand et al. [16] identify conditions under which the algorithm by Arora et al. [6] *can* be generalized to deeper architectures, but these are very particular cases that one would not expect to occur in practice.

While minimizing training and generalization errors are different problems, the hardness of the former also imposes challenges on the latter. Strategies to circumvent hardness from the perspective of learning theory include allowing improper learners, restricting the type of weights allowed in a neural network, or imposing assumptions on the underlying distribution. For example, Chen et al. [23] show fixed-parameter tractability of learning a ReLU network under several assumptions, including Gaussian data and Lipschitz continuity of the network. We refer to [8, 22, 31, 39, 40, 42] as a non-exhaustive list of other results about (non-)learnability of ReLU networks in different settings. Generally, identifying the exact boundary between tractable and intractable cases is an interesting direction for future work.

5.2 Expressivity of ReLU networks

It is essential for our reduction to understand the classes of functions representable by certain ReLU network architectures. So-called *universal approximation theorems* state that a single hidden layer (with arbitrary width) is already sufficient to approximate every continuous function on a bounded domain with arbitrary precision [25,52]. However, deeper networks require much fewer neurons to reach the same expressive power, yielding a potential theoretical explanation of the dominance of deep networks in practice [6,33,46,48,59,68,71,74,89,92]. Other related work includes counting and bounding the number of linear regions [47,65,66,69,71,83,88], classifying the set of functions *exactly* representable by different architectures [6,7,29,45,50,51,67,95], or analyzing the memorization capacity of ReLU networks [91,93,94]. Huchette et al. [53] provide a survey on the interactions of neural networks and polyhedral geometry, including implications on training, verification, and expressivity.

5.3 Existential theory of the reals

The complexity class ∃ℝ gains its importance from numerous important decision problems that have been shown to be complete for this class in recent years. The name ∃ℝ was introduced by Schaefer [75] who also pointed out that several NP-hardness reductions from the literature actually implied ∃ℝ-hardness. For this reason, several important ∃ℝ-completeness results were obtained before the need for a dedicated complexity class became apparent. A compendium for the class ∃ℝ was written by Schaefer et al. [79].

Common features of ∃ℝ-complete problems are their continuous solution space and the nonlinear relations between their variables. Important ∃ℝ-completeness results include the realizability of abstract order types [64,86] and geometric linkages [76], as well as the recognition of many types of geometric intersection graphs [21,56,58,61,62]. More results appeared in the graph drawing community [32,54,60,77,78], regarding polytopes [73], the study of Nash-equilibria [9,12,37,80], matrix factorization [24,82,85], or continuous constraint satisfaction problems [63]. In computational geometry, we would like to mention the art gallery problem [2,87] and covering polygons with convex polygons [1].

Recently, the community started to pay more attention to higher levels of the *real polynomial hierarchy*, which also capture several interesting algorithmic problems [13,19,26,32,55,81].

6 Proof ideas

To prove ∃ℝ-hardness, we reduce from ETR-INV, a restricted version of ETR. An instance of ETR-INV consists of real variables and a conjunction of constraints in these variables. Each constraint is either an addition constraint $X + Y = Z$, or an inversion constraint $X \cdot Y = 1$. Given such an instance, we construct a TRAIN-F2NN instance that models the variables, and in which exact fitting of all the data points corresponds to simultaneously satisfying all addition and inversion constraints.

6.1 Variables

A natural candidate for encoding variables are the weights and biases of the neural network. However, those did not prove to be suitable for our purposes. The main problem with using the parameters of the neural network as variables is that the same function can be computed by many neural networks with different combinations of these parameters. We are not aware of an easy way to normalize the parameters.

To circumvent this issue, we work with the functions representable by fully connected two-layer neural networks directly. We frequently make use of the geometry of the plots of these functions. For now, it is only important to understand that each hidden ReLU neuron encodes a continuous piecewise linear function with exactly two pieces (both of constant gradient). These two pieces are separated by a so-called *breakline*. Now, if we have m hidden neurons, their individually encoded functions add up such that the function computed by the whole neural network is a continuous piecewise linear function with at most m breaklines. All cells of the function between the breaklines have constant gradient.

To keep things simple for now, let us first consider a neural network with only one input and one output neuron. We place a series of data points $(x_i; y_i) \in \mathbb{R} \times \mathbb{R}$ as seen in Figure 2. All continuous piecewise linear functions $f(\cdot, \Theta)$ computed by a neural network with only four hidden neurons (i. e., only four breaklines and therefore at most five pieces) that fit these data points exactly must be very similar. In fact, they can only differ in one degree of freedom, namely the slope of the piece going through the data point p. In our construction, this slope represents the value of a variable. The whole set of data points enforcing this configuration is called a *variable gadget*.

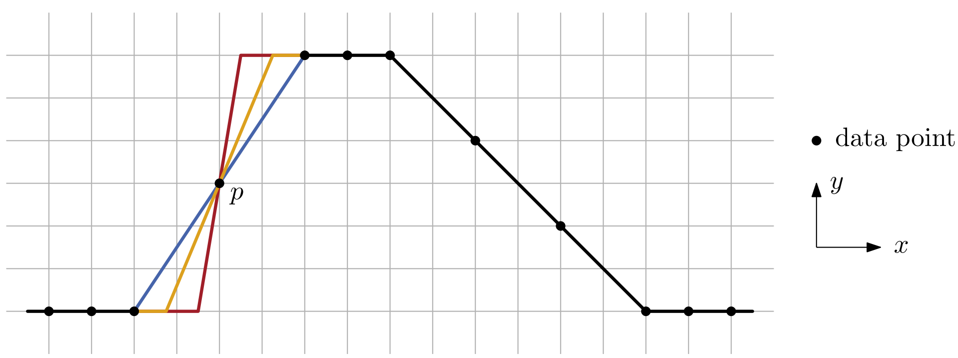


Figure 2: The value of $f(\cdot, \Theta)$ is fixed (black part), except for the segment through data point p. The red, orange and blue segments are just three out of uncountably many possibilities. Its slope can be used to encode a real-valued variable.

6.2 Linear dependencies

The key insight for encoding constraints between variables is that we can relate the values of several variable gadgets by a data point: By placing a data point p at a location where several

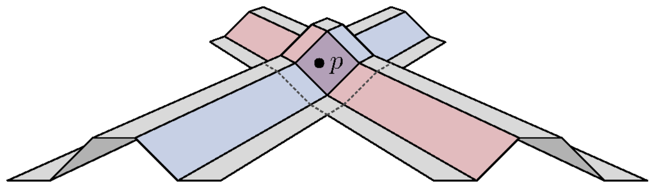


Figure 3: Two intersecting variable gadgets. The slopes of the blue and the red region encode the values. Point p lies in the intersection of both and can encode a linear relationship between them.

variable gadgets overlap, each of the variable gadgets contributes its part towards fitting p. The exact contribution of each variable gadget depends on its slope. Consequently, if one variable gadget contributes more, the others have to lower their contribution by the same amount. This enforces linear dependencies between different variable gadgets and can be used to design *addition* and *copy gadgets*.

We need a second input dimension in order to intersect multiple variable gadgets. This extends each variable gadget into a *stripe* in $\mathbb{R}^2$, with Figure 2 showing only an orthogonal cross-section of this stripe. See Figure 3 for two intersecting variable gadgets. Much of the technical difficulties lie in the subtleties to enforce the presence of multiple (possibly intersecting) gadgets using a finite number of data points.

An addition gadget encodes a linear relation between three variables. In $\mathbb{R}^2$, three lines (or stripes) usually do not intersect in a single point. Thus, we have to carefully place the gadgets to guarantee such a single intersection point. To this end, we create copies of the variable gadgets involved (copying is a linear relation between just two variables, thus "easy"). These copies can then be positioned (almost) freely.

6.3 Inversion

We are not able to encode nonlinear constraints within only a single output dimension [6]. By adding a second output dimension, the neural network now represents two functions $f^1(\cdot, \Theta)$ and $f^2(\cdot, \Theta)$. Consequently, we are allowed to use data points with two different output labels, one for each output dimension.

One important observation is that the locations of the breaklines of the function $f = f(\cdot, \Theta) = (f^1(\cdot, \Theta), f^2(\cdot, \Theta))$ are independent of the weights of the edges in the second layer of the neural network. Thus, both functions f^1 and f^2 have the same breaklines. Still, setting some weights to zero may *erase* a breakline in one of the functions.

An *inversion gadget* (realizing the constraint $X \cdot Y = 1$) also corresponds to a stripe in $\mathbb{R}^2$. For simplicity, we only show a cross-section here, see Figure 4. In each output dimension individually, the inversion gadget acts exactly like a variable gadget. The inversion gadget can therefore be understood as a variable gadget that carries two values.

We prove that by allowing only five breaklines in total, a function f can only fit all data points exactly if f^1 and f^2 share three of their four breaklines (while both having one "exclusive" breakline each, which is erased in the other dimension). This enforces a nonlinear dependency between the slopes of f^1 and f^2. By choosing the right parameters, this nonlinear relation models an inversion constraint.

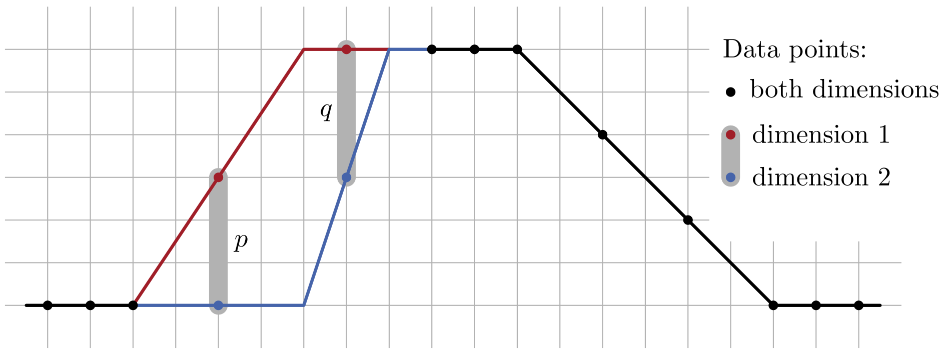


Figure 4: Data points p and q have different labels in the two output dimensions, enforcing that the slopes of the red and the blue pieces are related via a nonlinear dependency.

6.4 Reduction

Let us illustrate the reduction by giving a simple example. Note that this is not yet the complete picture. We start with an ETR-INV instance, for example, deciding whether the following sentence

$$\exists X_1, X_2, X_3, X_4 \in \mathbb{R} : (X_1 + X_2 = X_3) \wedge (X_1 + X_3 = X_4) \wedge \\ (X_1 \cdot X_4 = 1) \wedge (X_4 \cdot X_3 = 1)$$

is true. This instance has four variables, X_1, X_2, X_3, X_4, and four constraints: two additions and two inversions. Recall that every gadget corresponds to a stripe in the input space $\mathbb{R}^2$. See Figure 5 for the following construction (the stripes are drawn as lines for better readability).

- We add a variable gadget for each of the variables. All of these are placed such that their corresponding stripes are parallel and do not overlap, see the horizontal lines in Figure 5.
- We introduce three more variable gadgets for each addition constraint, one per variable involved. These are placed such that they have a common intersection point while also intersecting their corresponding variable gadget. In Figure 5, see the two bundles to the left. A data point at the triple intersection enforces the addition constraint, while data points labelled $\bullet_=$ encode that the values of the two intersecting variable gadgets are equal.
- Last, we add an inversion gadget for each inversion constraint and place it such that it intersects the variable gadgets of the two variables involved. See the two dashed lines in Figure 5. Data points labelled $\bullet^{=_1}$ ($\bullet^{=_2}$) enforce equality only in the first (second) output dimension.

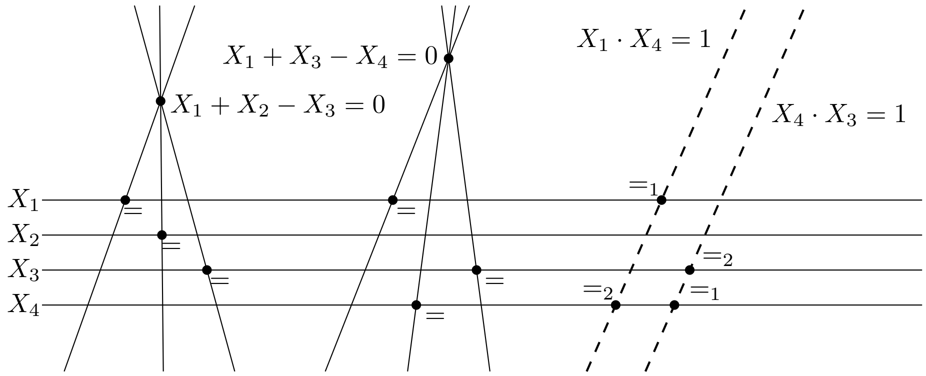


Figure 5: Overview of the global arrangement of the gadgets.

To see that the above reduction is correct, assume first that the ETR-INV instance is true. Then there exist real values for the four variables and these values can be used as the slopes of their corresponding variable gadgets. By the correctness of the individual gadgets (that we prove below), it follows that each data point is fit exactly.

Conversely, if all data points are fit exactly, then the correctness of the gadgets implies that the slopes of the variable gadgets give a solution to the ETR-INV instance.

7 ∃ℝ-membership

∃ℝ-membership is already proven by Abrahamsen et al. [3].

Proposition 7.1 ([3, Section 2])**.** TRAIN-F2NN ∈ ∃ℝ.

For the sake of completeness, while not being too repetitive, we briefly summarize their argument: ∃ℝ-membership is shown by describing a so-called *polynomial-time real verification algorithm* (see [34] for the formal details). The input of such an algorithm is a TRAIN-F2NN instance I, as well as a witness Θ consisting of real-valued weights and biases. Instance I consists of a network architecture, data points D and a target error γ. The algorithm has to verify that the neural network parameterized by Θ fits all data points in D with a total error at most γ. The underlying model of computation is the *real RAM*, that is, an extension of the classical word RAM by registers that can store arbitrary real numbers. Arithmetic operations ("+", "−", "·", "÷") on real registers take constant time.

The real verification algorithm given in [3] loops over all data points in D and evaluates the function realized by the neural network for each of them individually. As each hidden neuron uses a polynomial-time computable activation function, each such evaluation takes only polynomial time in the size of the network. Proposition 7.1 follows, since we are also guaranteed that the loss function can be computed in polynomial time on a real RAM.

8 ∃ℝ-hardness

This section is devoted to proving ∃ℝ-hardness of TRAIN-F2NN. Our reduction is mostly geometric, so we start by reviewing the underlying geometry of two-layer neural networks in Section 8.1. This is followed by a high-level overview of the reduction in Section 8.2, before we describe the gadgets in detail in Section 8.3. Finally, in Section 8.4, we combine the gadgets into the proof of Theorem 3.1.

8.1 Geometry of two-layer neural networks

Our reduction constructs a neural network that has just two input neurons, two output neurons, and m hidden neurons. Thus, for a given assignment Θ of weights and biases, it realizes a function $f(\cdot,\Theta)\colon \mathbb{R}^2 \to \mathbb{R}^2$. In this section, we build a geometric understanding of $f(\cdot,\Theta)$, in particular, we study the geometry of the plot of $f(\cdot,\Theta)$. For further results in this direction, we point the interested reader to [6, 29, 50, 67, 95] that investigate the set of functions exactly represented by different architectures of ReLU networks.

The i-th hidden ReLU neuron v_i realizes a function

$$\begin{aligned} f_i\colon \mathbb{R}^2 &\to \mathbb{R} \\ (x_1,x_2) &\mapsto \mathrm{ReLU}(a_{1,i}x_1 + a_{2,i}x_2 + b_i), \end{aligned}$$

where $a_{1,i}$ and $a_{2,i}$ are the edge weights from the first and second input neuron to v_i and b_i is its bias. Note that f_i is a continuous piecewise linear function: If $a_{1,i} = a_{2,i} = 0$, then f_i is constant, $f_i = \mathrm{ReLU}(b_i) = \max\{b_i, 0\}$. Otherwise, the domain $\mathbb{R}^2$ is partitioned into two half-planes, touching along a so-called *breakline* given by the equation $a_{1,i}x_1 + a_{2,i}x_2 + b_i = 0$. The two half-planes are (see Figure 6)

- the *inactive region* $\{(x_1,x_2) \subseteq \mathbb{R}^2 \mid a_{1,i}x_1 + a_{2,i}x_2 + b_i \le 0\}$, in which f_i is the constant 0, and
- the *active region* $\{(x_1,x_2) \subseteq \mathbb{R}^2 \mid a_{1,i}x_1 + a_{2,i}x_2 + b_i > 0\}$, in which f_i is positive and has a constant gradient.

Now let $c_{i,1}$ and $c_{i,2}$ be the weights of the edges connecting v_i with the first and second output neuron, and let $f(\cdot,\Theta) = (f^1(\cdot,\Theta), f^2(\cdot,\Theta))$. For $j \in \{1,2\}$, the function $f^j(\cdot,\Theta) = \sum_{i=1}^m c_{i,j} \cdot f_i(\cdot,\Theta)$ is a weighted linear combination of the functions computed at the hidden neurons. We make three observations.

- Each function computed by a hidden ReLU neuron has at most one breakline. Thus, the domain of $f^j(\cdot,\Theta)$ is partitioned into the cells of a line arrangement containing at most m breaklines. Apart from that, $f^j(\cdot,\Theta)$ has a constant gradient inside each cell.
- Let b_i be the breakline produced by a hidden neuron v_i in $f^j(\cdot,\Theta)$. Its position is solely determined by $a_{\cdot,i}$ and b_i. In particular, it is independent of $c_{i,j}$. Thus, the sets of breaklines of $f^1(\cdot,\Theta)$ and $f^2(\cdot,\Theta)$ are both subsets of the same set of (at most m) breaklines determined by the hidden neurons.

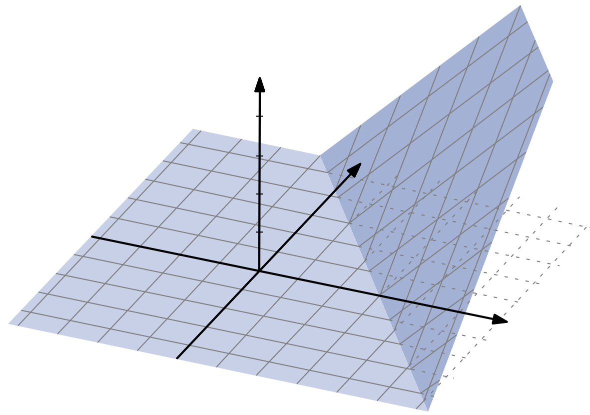


Figure 6: A continuous piecewise linear function computed by a hidden ReLU neuron. It has exactly one breakline; the flat part is the inactive region, whereas the sloped part is the active region.

- Even if all $f_i(\cdot, \Theta)$ have a breakline, their sum $f^j(\cdot, \Theta)$ at each output neuron might have fewer breaklines: It is possible for a breakline to be *erased* by setting $c_{i,j} = 0$. Other possibilities are that several breaklines contributed by different hidden neurons cancel each other (producing no breakline) or lie on top of each other (combining multiple breaklines into one). In our reduction we deliberately erase some breaklines in some output dimensions, i. e., we make use of the $c_{i,j} = 0$ trick. However, we avoid the other two cases of breaklines combining/canceling.

Combining the above observations yields a stronger statement: For each output neuron and breakline ℓ, the change of gradient of $f^j(\cdot, \Theta)$ along ℓ is constant (see also [29]). Based on this, we distinguish two types of breaklines.

Definition 8.1. A breakline ℓ is *concave* (*convex*) in $f^j(\cdot, \Theta)$ if the restriction of $f^j(\cdot, \Theta)$ to any two neighboring cells separated by ℓ is concave (convex).

The *type* of a breakline is an ordered pair $(t_1, t_2) \in \{\wedge, 0, \vee\}^2$ where t_1 and t_2 are symbols describing whether the breakline is concave ($\wedge$), erased (0), or convex ($\vee$) in $f^1(\cdot, \Theta)$ and $f^2(\cdot, \Theta)$, respectively.

By now, we have gained a geometric understanding of $f(\cdot, \Theta)$, the continuous piecewise linear function computed by a ReLU neural network with two input and two output neurons. However, not every continuous piecewise linear function can be computed by such a neural network. For the correctness of our reduction, we need a sufficient condition for this.

Lemma 8.2. *A continuous piecewise linear function $f \colon \mathbb{R}^2 \to \mathbb{R}^2$ whose breaklines form a line arrangement with m lines can be realized by a fully connected two-layer neural network with m hidden neurons if the following two conditions hold.*

- *In at least one cell of $\mathcal{L}$ the value of f is the constant* $(0, 0)$.

- *For each breakline $\ell \in \mathcal{L}$, the change of the gradient of f along ℓ is constant in both output dimensions.*

Proof. We can use the following construction. Add one hidden neuron per breakline, oriented such that its inactive region is the half-plane containing the $(0, 0)$-cell. The position solely depends on the weights of the first layer and the bias. The weights of the second layer are then chosen to produce the right change of gradient in each output dimension. It is easy to see that the sum of all these neurons computes f. □

We refer to [29] for a precise characterization of the functions representable by two-layer neural networks with m hidden neurons.

8.2 Preparing the reduction

We show $\exists\mathbb{R}$-hardness of TRAIN-F2NN by giving a polynomial-time reduction from ETR-INV to TRAIN-F2NN. ETR-INV is a variant of ETR that is frequently used as a starting point for $\exists\mathbb{R}$-hardness proofs in the literature [2, 3, 32, 60].

Formally, ETR-INV is a special case of ETR in which the quantifier-free part φ of the input sentence $\Phi :\equiv \exists X_1, \ldots, X_n \in \mathbb{R} \colon \varphi(X_1, \ldots, X_n)$ is a conjunction (only $\wedge$ is allowed) of constraints, each of which is either of the form $X + Y = Z$ or $X \cdot Y = 1$.

A promise version of ETR-INV allows us to assume that either there is no solution or there is a solution with all variables in $\left[\frac{1}{2}, 2\right]$. Our reduction starts from this promise version.

Theorem 8.3 ([2, Theorem 3.2]). ETR-INV *is $\exists\mathbb{R}$-complete.*

Furthermore, the next result shows that ETR-INV exhibits the same algebraic universality that we seek for TRAIN-F2NN.

Theorem 8.4 ([4]). *Let α be an algebraic number. Then there exists an instance of* ETR-INV*, which has a solution in $\mathbb{Q}[\alpha]$, but no solution when the variables are restricted to a field $\mathbb{F}$ that does not contain α.*

The reduction starts with an ETR-INV instance Φ and outputs an integer m and a set of n data points such that there is a fully connected two-layer ReLU neural network N with m hidden neurons exactly fitting all data points ($\gamma = 0$) if and only if Φ is true. Recall that the neural network N defines a continuous piecewise linear function $f(\cdot, \Theta) \colon \mathbb{R}^2 \to \mathbb{R}^2$.

We define several *gadgets* representing the variables as well as the linear and inversion constraints of the ETR-INV instance Φ. Strictly speaking, a gadget is defined by a set of data points that need to be fit exactly. These data points serve two tasks. First, most of the data points are used to enforce that $f(\cdot, \Theta)$ has m breaklines with predefined orientations and at almost predefined positions. Second, the remaining data points enforce relationships between the exact positions of different breaklines.

Globally, our construction yields $f(x, \Theta) = (0, 0)$ for "most" $x \in \mathbb{R}^2$. Each gadget consists of a constant number of parallel breaklines (enforced by data points) that lie in a *stripe* of constant width in $\mathbb{R}^2$. The value of $f(\cdot, \Theta)$ may be non-zero only within these stripes. The "semantics" of a gadget[2] is fully determined by the distances between its parallel breaklines. Thus, each gadget can be translated and rotated arbitrarily without affecting its semantics.

8.2.1 Abstractions

Describing all gadgets purely by their data points is tedious and obscures the relatively simple geometry enforced by these data points. We therefore introduce two additional constructs, namely *data lines* and *weak data points*, that simplify the presentation. In particular, data lines impose breaklines, which in turn are needed to define gadgets. Weak data points allow us to have features that are only active in one output dimension. How these constructs can be realized with carefully placed data points is deferred to Sections 8.3.6 and 8.3.7, after we have introduced all other gadgets. The realization of weak data points in in Section 8.3.6 requires an additional gadget.

- A *data line* $(\ell; y)$ consists of a line $\ell \subseteq \mathbb{R}^2$ and a label $y \in \mathbb{R}^2$. We say that a data line is fit if $f(\ell, \Theta) = \{y\}$, i. e., the neural network maps every point on it to y.

  As soon as we consider several gadgets, their corresponding stripes in $\mathbb{R}^2$ might intersect. We do not require that the data lines are fit correctly inside these intersections. This is justified because, as we are going to see below, each data line is realized by finitely many data points on it. We make sure that their coordinates do not lie in any of the intersections.

- A *weak data point* relaxes the notion of a regular data point and prescribes only a lower bound on the label. For example, we denote by $(x; y_1, \geq y_2)$ that $f^1(x, \Theta) = y_1$ and $f^2(x, \Theta) \geq y_2$. Weak data points can have such an inequality label in the first, the second, or both output dimensions.

8.3 Gadgets and constraints

We describe all gadgets in isolation first. The interaction of two or more gadgets is considered only where it is necessary. In particular, we assume that $f(x, \Theta)$ is zero everywhere for $x \in \mathbb{R}^2$ outside the outermost breaklines enforced by the gadgets. After all gadgets have been introduced, we describe the global arrangement of the gadgets in Section 8.4. Recall that, since each gadget can be freely translated and rotated, we can describe the positions of all its data lines and (weak) data points relative to each other. Since all data lines of a gadget are parallel, we can describe their relative positions solely by the distance between them.

Not all gadgets make use of the two output dimensions. Some gadgets have the same labels in both output dimensions for all of their data lines, and thus look the same in both output dimensions. For these gadgets, we simplify the usual notation of $(y_1, y_2) \in \mathbb{R}^2$ to single-valued labels $y \in \mathbb{R}$. In our figures, data points and functions looking the same in both output dimensions are drawn in black, while features only occurring in one dimension are drawn in different colors to distinguish them from each other.

[2]For a variable gadget, its “semantics” is the real number represented by it.

To clarify our exposition, let us define some terms we will use. Let $(\ell_1; y_i), \ldots, (\ell_k; y_k)$ be parallel data lines. Further, let $\ell \subseteq \mathbb{R}^2$ be an oriented line intersecting all ℓ_i. Without loss of generality, we assume that ℓ intersects ℓ_i before ℓ_j if and only if $i < j$. A *cross-section* through $(\ell_1; y_i), \ldots, (\ell_k; y_k)$ is defined as follows. For each data line $(\ell_i; y_i)$, the cross-section contains a data point $p_i = (x_i; y_i) \in \mathbb{R} \times \mathbb{R}^2$, where x_i is the oriented distance between the intersections of ℓ_1 and ℓ_i with ℓ. Two data points p_i and p_j in the cross-section are *consecutive* if $|i - j| = 1$. If ℓ is perpendicular to all ℓ_i, then the cross-section is *orthogonal*. The intersection of a breakline with ℓ is a *breakpoint*.

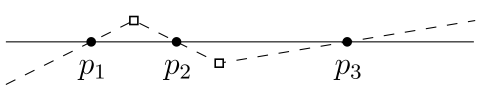
We draw cross-sections by projecting a data point $(x_i; y_i) \in \mathbb{R} \times \mathbb{R}^2$ into a two-dimensional coordinate system, marking x_i along the abscissa and y_i along the ordinate. If a y_i behaves differently in the two output dimensions, then we draw it twice and distinguish the two dimensions by color.

Observation 8.5. *Let f be a continuous piecewise linear function exactly fitting three consecutive data points, p_i, p_{i+1} and p_{i+2}, in a cross-section of a gadget. The following hold for each output dimension.*

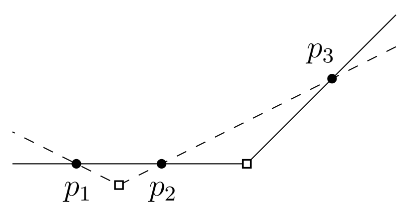
(i) If the three points are collinear, then f has either no breakpoint strictly between p_i and p_{i+2}, or at least two.

(ii) If the three points are not collinear, then f has some breakpoint b strictly between p_i and p_{i+2}. Furthermore, if p_{i+2} lies above (below) the ray from p_i through p_{i+1}, then b is convex (concave, resp.).

Observation 8.5 is the key to prove that data lines enforce breaklines of a certain type, with a prescribed orientation and (almost) fixed position. It is illustrated in Figure 7.



(a) If the points are collinear, then there is either no breakpoint or there are at least two.



(b) If the points are not collinear, then we need a breakpoint of a certain type, here convex.

Figure 7: Three consecutive points, p_1, p_2 and p_3, in a cross-section and possible continuous piecewise linear functions fitting them (solid and dashed).

8.3.1 Variable gadget

A *variable gadget* consists of twelve parallel data lines, $\ell_1, \ldots, \ell_{12}$, numbered from one side to the other, in the figures from left to right. Further, there is a weak data point q between ℓ_3 and ℓ_4. For all of these, the following table lists their relative distance to ℓ_1 and their label (note that both output dimensions have the same label).

	ℓ_1	ℓ_2	ℓ_3	q	ℓ_4	ℓ_5	ℓ_6	ℓ_7	ℓ_8	ℓ_9	ℓ_{10}	ℓ_{11}	ℓ_{12}
distance to ℓ_1	0	1	2	$3+\frac{2}{3}$	4	6	7	8	10	12	14	15	16
label	0	0	0	≥ 2	3	6	6	6	4	2	0	0	0

See Figure 8 for an orthogonal cross-section through a variable gadget.

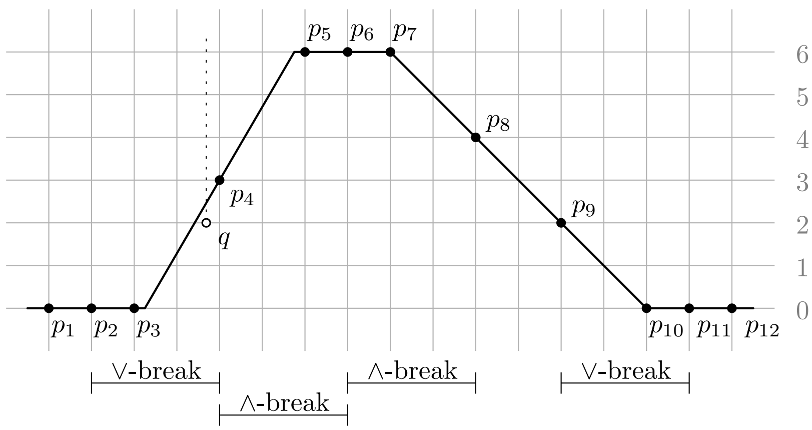


Figure 8: Orthogonal cross-section of a variable gadget. The bars below the cross-section indicate non-collinear triples used in the proof of Lemma 8.6. For example, there needs to be a convex breakpoint strictly between p_2 and p_4.

Lemma 8.6. *A continuous piecewise linear function f that fits $\ell_1, \ldots, \ell_{12}$ and q exactly must have at least four breaklines. If it has exactly four breaklines, then they must all be parallel to the data lines. In this case, let b_1, b_2, b_3 and b_4 be the breaklines, numbered from left to right. We have the following in each output dimension.*

- *f is 0 everywhere to the left of b_1 and to the right of b_4.*
- *f is 6 everywhere between b_2 and b_3.*
- *b_3 lies on ℓ_7 and b_4 lies on ℓ_{10}.*
- *The* slope *of the variable gadget, i. e., the norm of the gradient between b_1 and b_2, is at least $\frac{3}{2}$ and at most 3.*

Before we prove Lemma 8.6, let us describe the functionality of a variable gadget: The slope s_X of a variable gadget for a variable X is in $\left[\frac{3}{2}, 3\right]$. In order to represent values in $\left[\frac{1}{2}, 2\right]$, we say that a slope s_X encodes the value $X = s_X - 1$.

Proof of Lemma 8.6. We first prove that four breaklines are indeed necessary to fit all data lines exactly. Every orthogonal cross-section contains four non-collinear triples of consecutive points: (p_2, p_3, p_4), (p_4, p_5, p_6), (p_6, p_7, p_8) and (p_9, p_{10}, p_{11}). They pairwise share at most one point, so by Observation 8.5(ii), four breakpoints are indeed required. Since the data lines are parallel to each other, all orthogonal cross-sections look the same, and each breakpoint corresponds to a breakline that is parallel to the data lines. For the rest of the proof, we denote the breaklines by b_1, b_2, b_3 and b_4, numbered from left to right.

In the following, we consider each of the non-collinear triples individually, to further locate the positions of the breaklines. The following observations are all due to Observation 8.5.

- The non-collinear triple (p_2, p_3, p_4) implies that b_1 must be between ℓ_2 and ℓ_4. The collinear triple (p_1, p_2, p_3) enforces that b_1 is to the right of ℓ_3 and that f is 0 everywhere to the left of b_1.
- The non-collinear triple (p_4, p_5, p_6) implies that b_2 must be between p_4 and p_6. The collinear triple (p_5, p_6, p_7) enforces that b_2 is to the left of ℓ_5 and that f is 6 everywhere to the right of b_2.
- The non-collinear triple (p_6, p_7, p_8) implies that b_3 must be between ℓ_6 and ℓ_8. The collinear triples (p_5, p_6, p_7) and (p_7, p_8, p_9) leave ℓ_7 as the only remaining position for b_3. The collinear triple (p_5, p_6, p_7) further implies that f must be 6 everywhere to the left of b_3.
- The non-collinear triple (p_9, p_{10}, p_{11}) implies that b_4 must be between ℓ_9 and ℓ_{11}. The collinear triples (p_8, p_9, p_{10}) and (p_{10}, p_{11}, p_{12}) leave ℓ_{10} as the only remaining position for b_4. The collinear triple (p_{10}, p_{11}, p_{12}) further implies that f must be 0 everywhere to the right of b_4.

As b_1 must be on ℓ_3 or to its right and b_2 must be on p_5 or to its left, the slope is at least $\frac{3}{2}$. Last, weak data point q enforces that the slope is at most 3. $\square$

8.3.2 Measuring a value from a variable gadget

Consider a variable gadget for a variable X with slope s_X. We call the two parallel lines with distance 1 to ℓ_4 its *measuring lines*. More precisely, we distinguish between the *lower measuring line* (the one towards ℓ_3) and the *upper measuring line* (the one towards ℓ_5). Since the slope of the variable gadget is in $\left[\frac{3}{2}, 3\right]$, both measuring lines are always inside or at the boundary of the sloped part (in other words, between breaklines b_1 and b_2).

Assume that the variable gadget is fit exactly. Then, at any point p on ℓ_4, the variable gadget contributes 3 to $f(p, \Theta)$. It follows that a point p_u on the upper measuring line contributes $3 + s_X$ to $f(p_u, \Theta)$. Similarly, a point p_l on the lower measuring line contributes $3 - s_X$ to $f(p_l, \Theta)$. See Figure 9 for a visualization.

8.3.3 Linear constraints: addition and copying

Until this point, we considered individual gadgets separately from each other. As soon as we have two or more gadgets, their corresponding stripes may intersect, leading to interference of the gadgets inside these intersections. We exploit this to encode linear constraints.

Let $\mathcal{A}$ and $\mathcal{B}$ be disjoint subsets of the variables. We can enforce a linear constraint of the form $\sum_{A\in\mathcal{A}} A = \sum_{B\in\mathcal{B}} B$ using just one additional data point p. In particular, we care about the following two special cases.

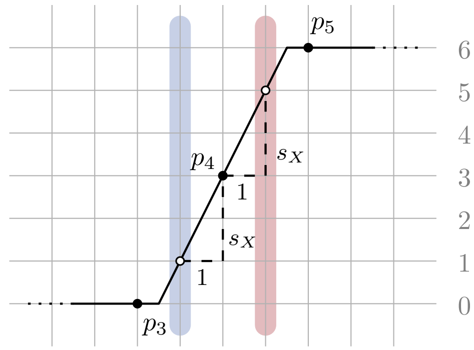


Figure 9: Partial cross-section of a variable gadget with slope $s_X = 2$, so $X = 1$. The lower and upper measuring lines are drawn in blue and red. This variable gadget contributes $3 - s_X$ to the lower and $3 + s_X$ to the upper measuring line.

- To copy a value from one variable X to another variable Y, we model $X = Y$ by $\mathcal{A} = \{X\}$ and $\mathcal{B} = \{Y\}$.
- To encode the addition $X + Y = Z$, we set $\mathcal{A} = \{X, Y\}$ and $\mathcal{B} = \{Z\}$.

For all variables in $\mathcal{A}$, the data point p must be on the upper measuring line of their corresponding variable gadget. Similarly, for variables in $\mathcal{B}$, the data point p must be on the lower measuring line. This requires a placement of the gadgets such that all required measuring lines intersect in a single point, where we can place p. This is trivial for $|\mathcal{A}| + |\mathcal{B}| = 2$, as it only requires the variables gadgets involved to be non-parallel, see Figure 10. For $|\mathcal{A}| + |\mathcal{B}| \geq 3$, this is more involved. We can use the equality constraint $X = Y$ to copy the value of a variable onto additional variable gadgets, which can then be positioned freely to obtain the required intersections, see Figure 11. We discuss the global layout to achieve this in more detail in Section 8.4 below.

Lemma 8.7. *Let $\mathcal{A}$ and $\mathcal{B}$ be disjoint subsets of the variables. A data point p with label $4|\mathcal{A}| + 2|\mathcal{B}|$ placed on the upper measuring line of each $A \in \mathcal{A}$ and the lower measuring line of each $B \in \mathcal{B}$ enforces the linear constraint $\sum_{A \in \mathcal{A}} A = \sum_{B \in \mathcal{B}} B$.*

Proof. Let s_X be the slope of the variable gadget for variable X. For each $A \in \mathcal{A}$, the data point p is placed on the upper measuring line of A's variable gadget, so A contributes $3 + s_A$ to $f(p, \Theta)$. Similarly, for each variable $B \in \mathcal{B}$, the data point p is placed on the lower measuring line of B's variable gadget, so B contributes $3 - s_B$ to $f(p, \Theta)$.

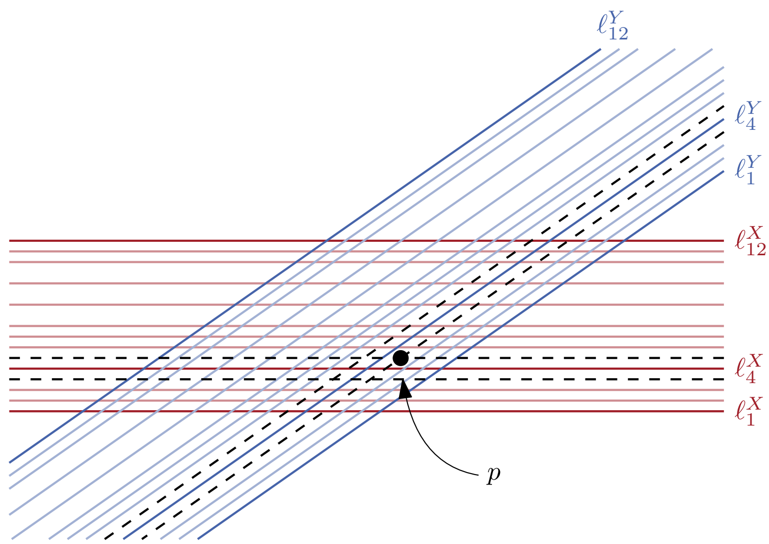


Figure 10: Top-down view on the intersection of two variable gadgets corresponding to two variables X (red) and Y (blue). The dashed lines are their measuring lines. The point p is placed at the intersection of the upper measuring line for X and lower measuring line for Y, and receives label 6 to enforce the constraint $X = Y$.

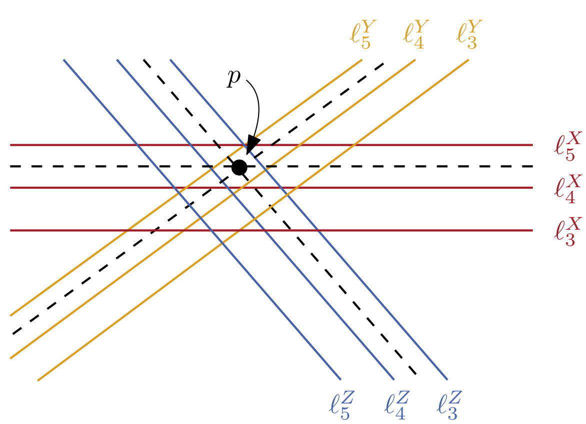


Figure 11: Top-down view of the intersection of three variable gadgets corresponding to variables X (red), Y (orange), and Z (blue). The dashed lines are the upper measuring lines for X and Y and the lower measuring line for Z, intersecting in a single point p with label 10. This realizes the constraint $X + Y = Z$.

The overall contribution of all variables involved adds up to

$$\begin{aligned} f(p,\Theta) &= \sum_{A\in\mathcal{A}}(3+s_A) + \sum_{B\in\mathcal{B}}(3-s_B) \\ &= \sum_{A\in\mathcal{A}}(4+A) + \sum_{B\in\mathcal{B}}(2-B) \\ &= 4|\mathcal{A}| + 2|\mathcal{B}| + \sum_{A\in\mathcal{A}} A - \sum_{B\in\mathcal{B}} B, \end{aligned}$$

where we used that $X = s_X - 1$ to get from the first to the second line. Setting the label of p to $4|\mathcal{A}| + 2|\mathcal{B}|$ yields that p is fit exactly if and only if $\sum_{A\in\mathcal{A}} A = \sum_{B\in\mathcal{B}} B$. □

As already mentioned above, we do not require Lemma 8.7 in its full generality, but only two special cases: The only linear constraint in an ETR-INV instance is the addition $X + Y = Z$. Additionally, we also need the ability to copy values in our reduction, i. e., constraints of the form $X = Y$.

Corollary 8.8. *To encode the addition constraint $X + Y = Z$ of* ETR-INV*, data point p has label* 10*. For the copy constraint $X = Y$, the data point has label* 6*.*

8.3.4 Inversion gadget

In essence, an *inversion gadget* is the superposition of two variable gadgets. By using data lines with different labels in the two output dimensions, it can represent two real variables at once. However, their values have a non-linear dependency.

Formally, the inversion gadget consists of 13 parallel data lines $\ell_1, \ldots, \ell_{13}$, numbered from one side to the other, in the figures from left to right. The following table lists their relative distance to ℓ_1 and their labels.

	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6	ℓ_7	ℓ_8	ℓ_9	ℓ_{10}	ℓ_{11}	ℓ_{12}	ℓ_{13}
distance to ℓ_1	0	1	2	4	7	9	10	11	13	15	17	18	19
label in dim. 1	0	0	0	3	6	6	6	6	4	2	0	0	0
label in dim. 2	0	0	0	0	3	6	6	6	4	2	0	0	0

See Figure 12 for an orthogonal cross-section through an inversion gadget.

Lemma 8.9. *A continuous piecewise linear function that fits $\ell_1, \ldots, \ell_{13}$ exactly must have at least five breaklines. If it has exactly five breaklines, then they must all be parallel to the data lines. In this case, let b_1, b_2, b_3, b_4 and b_5 be the breaklines, numbered from left to right. The following hold.*

- *f is $(0, 0)$ everywhere to the left of b_1 and to the right of b_5.*
- *In output dimension 1, f is 6 everywhere between b_2 and b_4.*
- *In output dimension 2, f is 0 everywhere to the left of b_2 and 6 everywhere between b_3 and b_4.*

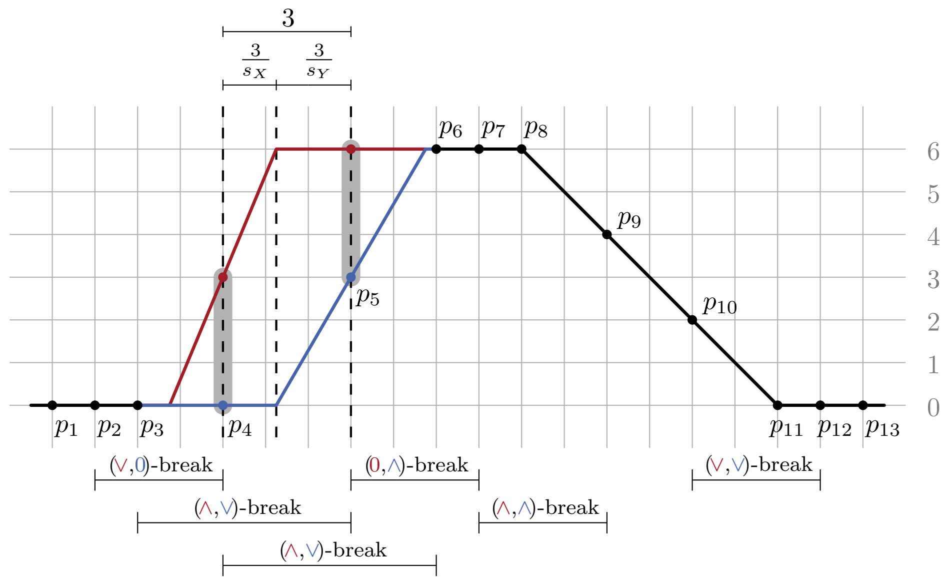


Figure 12: Cross-section of the inversion gadget. Data points p_4 and p_5 have different labels in the first (red) and second (blue) output dimension.

- *b_4 is on ℓ_8 and b_5 is on ℓ_{11}.*
- *The inversion gadget has two* slopes *s_X and s_Y.*
  - *In dimension 1, slope s_X is the norm of the gradient between b_1 and b_2.*
  - *In dimension 2, slope s_Y is the norm of the gradient between b_2 and b_3.*

  We have $s_X s_Y = s_X + s_Y$.

Before we prove Lemma 8.9, let us describe the functionality of an inversion gadget: As mentioned above, an inversion gadget is the superposition of two variable gadgets. Only four of the five breaklines are "visible" in each output dimension, the fifth being erased. It has a slope in each dimension: s_X in the first, s_Y in the second. Therefore, it encodes two values, $X = s_X - 1$ and $Y = s_Y - 1$. We have

$$XY = (s_X - 1)(s_Y - 1) = s_X s_Y - s_X - s_Y + 1 \overset{(*)}{=} s_X + s_Y - s_X - s_Y + 1 = 1,$$

where the equality labeled $(*)$ follows from the $s_X s_Y = s_X + s_Y$ condition provided by Lemma 8.9. We conclude that the non-linear relation between the two slopes exactly models the inversion constraint $XY = 1$ of an ETR-INV instance.

Proof of Lemma 8.9. As in the proof of Lemma 8.6, we use non-collinear triples to argue that each orthogonal cross-section requires exactly five breakpoints. Again, because all data lines are parallel to each other, all orthogonal cross-sections look the same, and all breakpoints correspond to a breakline that is parallel to the data lines.

All the following observations rely on Observation 8.5.

- The non-collinear triple (p_2, p_3, p_4) enforces a breakline of type $(\vee, 0)$ strictly between ℓ_2 and ℓ_4. We call this breakline b_1. As (p_1, p_2, p_3) is collinear in both dimensions, b_1 must be on or to the right of ℓ_3 and f must be $(0, 0)$ everywhere to the left of b_1.
- The non-collinear triple (p_5, p_6, p_7) enforces a breakline of type $(0, \wedge)$ strictly between ℓ_5 and ℓ_7. We call this breakline b_3. As (p_6, p_7, p_8) is collinear in both dimensions, b_3 must be on or to the left of ℓ_6 and f must be $(6, 6)$ everywhere to the right of b_3.
- The non-collinear triple (p_7, p_8, p_9) enforces a breakline of type $(\wedge, \wedge)$ strictly between ℓ_7 and ℓ_9. We call this breakline b_4. As (p_6, p_7, p_8) and (p_8, p_9, p_{10}) are collinear in both dimensions, b_4 must lie on ℓ_8.
- The non-collinear triple (p_{10}, p_{11}, p_{12}) enforces a breakline of type $(\vee, \vee)$ strictly between ℓ_{10} and ℓ_{12}. We call this breakline b_5. The triples (p_9, p_{10}, p_{11}) and (p_{11}, p_{12}, p_{13}) are collinear in both dimensions, so b_5 must lie on ℓ_{11}.

The triples enforcing the breaklines b_1, b_3, b_4 and b_5 are pairwise disjoint, so all of these breaklines are indeed necessary.

An orthogonal cross-section of an inversion gadget contains two more non-collinear triples: (p_3, p_4, p_5) and (p_4, p_5, p_6). Both enforce a breakline of type $(\wedge, \vee)$. Note, that all other non-collinear triples intersecting them have a different type. Therefore, none of the previous four breaklines is compatible, and at least one more breakline is indeed necessary. Assuming that $\ell_1, \ldots, \ell_{13}$ must be fit with just five breaklines, we call this breakline b_2, and conclude that it must be on or between ℓ_4 and ℓ_5. Further, f must be 0 everywhere in dimension 2 to the left of b_2.

It remains to prove that $s_X s_Y = s_X + s_Y$. To this end, we derive the exact position of b_2 between ℓ_4 and ℓ_5. In dimension 1, we have $f^1(\ell_4, \Theta) = 3$ and $f^1(b_2, \Theta) = 6$. The distance between ℓ_4 and b_2 is $\frac{6-3}{s_X}$. In dimension 2, we have $f^2(\ell_5, \Theta) = 3$ and $f^2(b_2, \Theta) = 0$. The distance between ℓ_5 and b_2 is $\frac{3-0}{s_Y}$. We conclude that $3 = \frac{3}{s_X} + \frac{3}{s_Y}$, which is equivalent to $s_X + s_Y = s_X s_Y$ for all $s_X, s_Y \neq 0$. □

8.3.5 Applying the inversion gadget

An inversion gadget has two pairs of measuring lines, one in each dimension. The lower and upper measuring lines in dimension 1 have distance 1 to ℓ_4. Similar they have distance 1 to ℓ_5 in dimension 2.

To encode an $XY = 1$ constraint of an ETR-INV instance, we first identify two normal variable gadgets carrying the variables X and Y. Then the inversion gadget is placed such that its measuring lines intersect the measuring lines of the variable gadgets. We copy X to the first dimension of the inversion gadget at the intersection with the variable gadget for X. Similarly, we copy Y to the second dimension of the inversion gadget at the intersection with the variable gadget for Y. This copying can be achieved using weak data points that are only active in the respective dimension (having a "≥ 0" label in the other dimension). See Figure 13 for a top-down view. Technically, the inversion gadget enforces the inversion constraint only in one dimension of each variables gadget involved. However, this is sufficient because variable gadgets always carry the same value in both output dimensions.

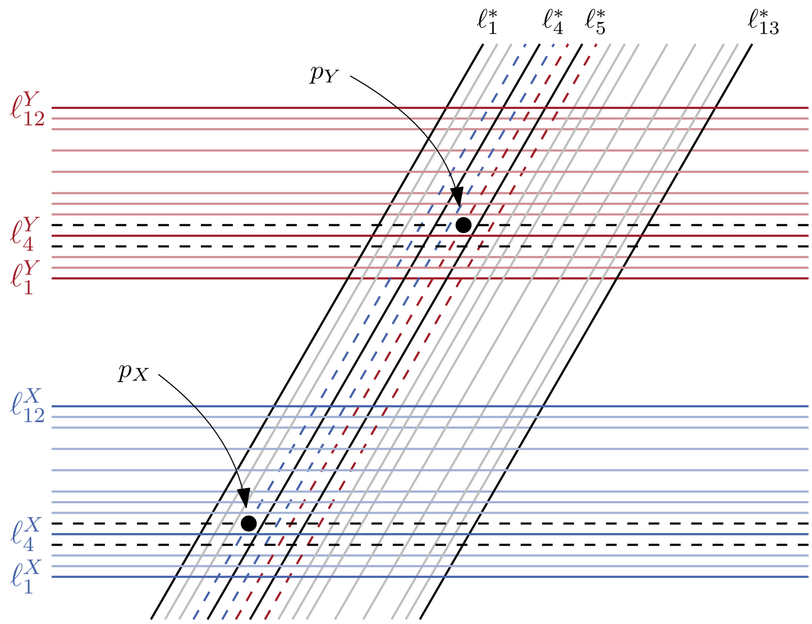


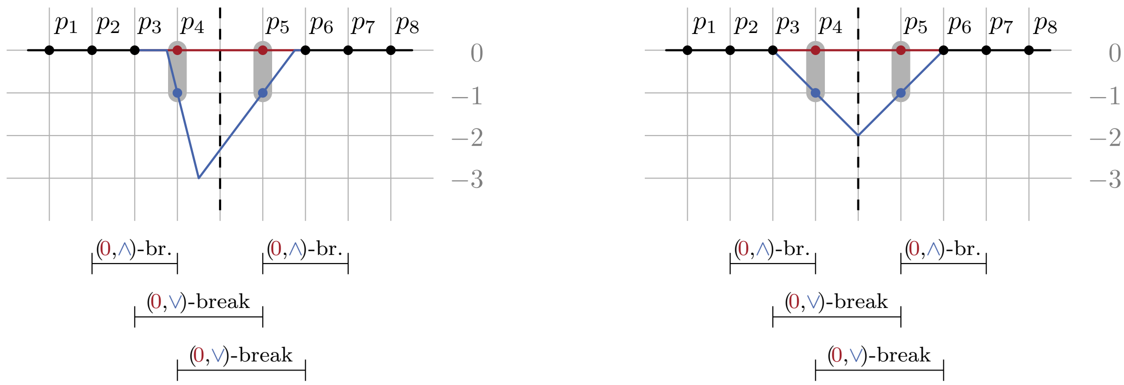
Figure 13: Top-down view on two variable gadgets (horizontal) for variables X (blue) and Y (red) (the data lines are solid, the measuring lines are dashed). The sloped gadget is an inversion gadget. Two weak data points p_X and p_Y copy X and Y to the first and second dimension of the inversion gadget, respectively.

8.3.6 Realizing weak data points: lower bound gadgets

So far, we used weak data points, i. e., data points whose label is just a lower bound on $f(\cdot, \Theta)$. Weak data points are just a concept meant to simplify the description of the gadgets; a TRAIN-F2NN instance cannot contain weak data points. For this reason, we introduce a *lower bound gadget* that simulates a weak data point using only ordinary data points, i. e., data points with constant labels.

Recall that a weak data point can have a lower bound label in either one or in both dimensions. For this reason, a lower bound gadget can be either *active* or *inactive* in each dimension. If the lower bound gadget is active in some dimension, its breaklines form a $\vee$-shape of (almost) arbitrary depth in that dimension. On the other hand, if the lower bound gadget is inactive in some output dimension, then it is the constant 0 in this dimension.

Formally, a lower bound gadget consists of eight data lines $\ell_1, \ldots, \ell_8$, numbered from one side to the other, in the figures from left to right. The following table lists their relative distance to ℓ_1 and their labels.



(a) The lower bound gadget can be asymmetric.

(b) A lower bound gadget has a maximum contribution to the weak data point of −2.

Figure 14: Cross-sections of a lower bound gadget which is inactive in the first (red) dimension and active in the second (blue) dimension. It is used to simulate a weak data point in the active dimension (blue) that lies on the dashed vertical line. It does not contribute to $f(\cdot,\Theta)$ in the inactive dimension (red), i. e., all breaklines are erased (type 0).

	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6	ℓ_7	ℓ_8
distance to ℓ_1	0	1	2	3	5	6	7	8
label in active dimension(s)	0	0	0	−1	−1	0	0	0
label in inactive dimension	0	0	0	0	0	0	0	0

See Figure 14 for orthogonal cross-sections through a lower bound gadget. We always assume that a lower bound gadget has at least one active dimension, as it is otherwise unnecessary.

Lemma 8.10. *A continuous piecewise linear function f that fits $\ell_1,\dots,\ell_8$ exactly must have at least three breaklines. If it has exactly three breaklines, then they must all be parallel to the data lines. In this case, let b_1, b_2 and b_3 be the breaklines. In an inactive dimension, f is the constant 0. In an active dimension, the following hold.*

- *f is 0 everywhere to the left of b_1 and to the right of b_3.*
- *$f(p,\Theta) \le -2$ for all points p with equal distance to ℓ_4 and ℓ_5.*

Proof. As in the proof of Lemma 8.6, we use non-collinear triples to argue that at least three breaklines are necessary for a lower bound gadget with at least one active dimension. Again, because all data lines are parallel to each other, all orthogonal cross-sections look the same, and all breakpoints correspond to a breakline that is parallel to the data lines.

The following observations rely on Observation 8.5. In an inactive dimension, all triples are collinear, so f must be 0 everywhere. From now on, we focus purely on the active dimension(s).

- The non-collinear triple (p_2, p_3, p_4) enforces a $\wedge$-type breakline strictly between ℓ_2 and ℓ_4. We call this breakline b_1. By the collinear triple (p_1, p_2, p_3), b_1 must be on or to the right of ℓ_3. Further, f must be 0 everywhere to the left of b_1.

- The non-collinear triple (p_5, p_6, p_7) enforces a $\wedge$-type breakline strictly between ℓ_5 and ℓ_7. We call this breakline b_3. By the collinear triple (p_6, p_7, p_8), b_3 must be on or to the left of ℓ_6. Further, f must be 0 everywhere to the right of b_3.

Breaklines b_1 and b_3 are both necessary, as the non-collinear triples enforcing them are disjoint. The lower bound gadget contains two more non-collinear triples: (p_3, p_4, p_5) and (p_4, p_5, p_6), both of type $\vee$. Assuming that all data lines are fit with just three breaklines, a third breakline called b_2 must lie on or between p_4 and p_5. The value of $f(b_2, \Theta)$ can be arbitrarily small. However, in the extreme case, we have $b_1 = \ell_4$ and $b_3 = \ell_5$, in which case $f(b_2, \Theta) = -2$. No larger values are possible. □

For a lower bound gadget with two active dimensions, we can make the following observation. It holds because the breaklines must be at the exact same positions in both output dimensions.

Observation 8.11. *A lower bound gadget that is active in both dimensions contributes the same amount in both dimensions.*

We need one lower bound gadget per weak data point. It gets placed such that the weak data point is equidistant to ℓ_4 and ℓ_5. The weak data point with label $\geq y$ is converted into an ordinary data point with label $y - 2$.

By Lemma 8.10, the lower bound gadget can contribute any value $c \in (-\infty, -2]$ to the new data point. Thus, the data point can be fit perfectly if and only if the other gadgets contribute at least a value of y to the data point, that is, the intended lower bound constraint is met.

8.3.7 Realizing data lines using data points

We previously assumed that our gadgets are defined by data *lines*, while in reality, we are only allowed to use data *points*. In this section, we argue that a set of data lines can be simulated by replacing each data line by three data points. This allows us to define the gadgets described throughout previous sections solely using data points.

This section is devoted to showing the following lemma, which captures this transformation formally. Note that our replacement of data lines by data points does not work in full generality, but we show it for all the gadgets that we constructed.

Lemma 8.12. *Assume we are given a set of variable, inversion and lower bound gadgets that in total requires at least m breaklines (four, five, and three per variable, inversion and lower bound gadget, respectively). Further, let the gadgets be placed in $\mathbb{R}^2$ such that no two parallel gadgets overlap. Then each data line can be replaced by three data points, such that a continuous piecewise linear function with at most m breaklines fits the data points if and only if it fits the data lines.*

For the proof, consider the line arrangement induced by the data lines. We introduce three vertical lines v_1, v_2, v_3 to the right of all intersections. The vertical lines are placed at unit distance to one another. In our construction in Section 8.4, we make sure that no data line is vertical. Thus, each data line intersects each of the vertical lines exactly once. We place one data point on each intersection between a vertical line and a data line. The new data point inherits the label of the underlying data line. Furthermore, on each vertical line, we ensure that the minimum distance α between any two data points belonging to different gadgets is larger than the maximum distance w between data points belonging to the same gadget. This can be achieved by placing the v_1, v_2 and v_3 far enough to the right and by ensuring a minimum distance between parallel gadgets. See Figure 15 for an illustration.

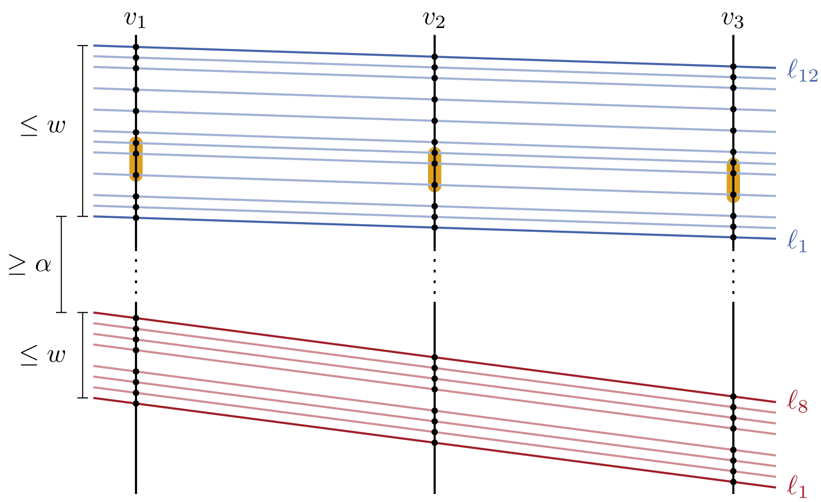


Figure 15: Data lines defining a variable gadget (blue) and a lower bound gadget (red), and their intersections with the vertical lines v_1, v_2, v_3. We add a data point at each intersection. The values α and w describe the minimal distance between data lines of different gadgets, and the maximal distance between data lines of the same gadget, respectively. In orange, we highlighted three matching breakpoint intervals (forcing a $\wedge$-breakpoint between ℓ_4 and ℓ_6 of the variable gadget).

Along each of the three vertical lines, the data points form cross-sections of all the gadgets, similar to the cross-sections shown in Figures 8, 12 and 14 (but here, the cross-sections are not orthogonal). We have previously analyzed cross-sections of individual gadgets in the proofs of Lemmas 8.6, 8.9 and 8.10. There, we identified certain intervals between some of the data lines that need to contain a *breakpoint* (the intersection of a breakline and the cross section). We refer to these intervals as *breakpoint intervals* along the vertical lines. Note that a breakpoint interval may degenerate to just one point. By our placement of the vertical lines, the cross-sections (and thus also the breakpoint intervals) of different gadgets do not overlap.

Any two data lines bounding a breakpoint interval on v_1 also bound a breakpoint interval on v_2 and v_3. We call the three breakpoint intervals on v_1, v_2 and v_3 which are bounded by the same data lines *matching* breakpoint intervals.

In total, there are $3m$ breakpoint intervals. We show that the only way to *stab* each of them exactly once using m breaklines is if each breakline stabs exactly three matching breakpoint intervals. The first observation towards this is that each breakline can only stab a single breakline interval per vertical line because all breakline intervals are pairwise disjoint. Thus, having m breakpoint intervals on each vertical line, each of the m breaklines has to stab exactly three intervals, one per vertical line. In a first step, we show that each breakline has to stab three breakpoint intervals belonging to the same gadget.

Claim 8.13. *Each breakline has to stab three breakpoint intervals of the same gadget.*

Proof of Claim. The proof is by induction on the number of gadgets. For a single gadget, the claim holds trivially. For the inductive step, we consider the lowest gadget g (on v_1, v_2 and v_3) and assume for the sake of contradiction that there is a breakline b stabbing a breakpoint interval of g on v_2 and a breakpoint interval of a different gadget g' above g on v_1. By construction, the minimum distance α between different gadgets is larger than the maximum width w of any gadget on all three vertical lines. Thus, the distance of any breakpoint interval of g' to any breakpoint interval of g on v_1 is larger than the width of g on v_3. Therefore, we know that the breakline b intersects v_3 below any breakpoint intervals of g, which is the lowest gadget on v_3. Thus, it stabs at most two breakpoint intervals in total, and therefore not all intervals can be stabbed. The same reasoning holds if the roles of v_1 and v_3 are flipped. All breaklines stabbing breakpoint intervals of g on v_2 must therefore also stab breakpoint intervals of g on v_1 and v_3. Applying the induction hypothesis on the remaining gadgets, it follows that each breakline only stabs breakpoint intervals of the same gadget. ◁

We can therefore analyze the situation for each gadget in isolation. The main idea is to distinguish by the type of the required breakline. Each breakline must stab three breakpoint intervals of the same type. Let us summarize the findings about required breakline locations and types from the proofs of Lemmas 8.6, 8.9 and 8.10 in Table 1.

Table 1: Location and type of the breaklines in variable gadgets, inversion gadgets, and lower bound gadgets.

	Location	Type
b_1	$[\ell_3, \ell_4)$	$(\vee, \vee)$
b_2	$(\ell_4, \ell_5]$	$(\wedge, \wedge)$
b_3	on ℓ_7	$(\wedge, \wedge)$
b_4	on ℓ_{10}	$(\vee, \vee)$

(a) Variable gadget.

	Location	Type
b_1	$[\ell_3, \ell_4)$	$(\vee, 0)$
b_2	(ℓ_4, ℓ_5)	$(\wedge, \vee)$
b_3	$(\ell_5, \ell_6]$	$(0, \wedge)$
b_4	on ℓ_8	$(\wedge, \wedge)$
b_5	on ℓ_{11}	$(\vee, \vee)$

(b) Inversion gadget.

	Location	Type
b_1	$[\ell_3, \ell_4)$	$(0, \wedge)$
b_2	(ℓ_4, ℓ_5)	$(0, \vee)$
b_3	$(\ell_5, \ell_6]$	$(0, \wedge)$

(c) Lower bound gadget.

Claim 8.14. *To stab all breakpoint intervals of a variable gadget with only four breaklines, each of them has to stab three matching breakpoint intervals.*

Proof of Claim. See Table 1a. On the three vertical lines, there are six breakpoint intervals for breaklines of type $(\vee, \vee)$ in total. If only two breaklines should stab these six breakpoint intervals, one breakline needs to stab at least two of the single-point intervals. If a breakline goes through two of the single points, it also goes through the third point, and can thus not go through the proper intervals. Therefore, one breakline must stab the single-point intervals, and the other one stabs the proper breakpoint intervals.

The same argument can be made for the breakpoint intervals of type $(\wedge, \wedge)$, and thus each breakline stabs three matching breakpoint intervals. ◁

Claim 8.15. *To stab all breakpoint intervals of an inversion gadget with only five breaklines, each of them has to stab three matching breakpoint intervals.*

Proof of Claim. See Table 1b. All five sets of three matching breakpoint intervals have a different type of required breakline, thus each breakline stabs three matching breakpoint intervals. ◁

Claim 8.16. *To stab all breakpoint intervals of a lower bound gadget with only three breaklines, each of them has to stab three matching breakpoint intervals.*

Proof of Claim. See Table 1c. There is only one set of three breakpoint intervals for a breakline of type $(0, \vee)$, so it is trivially matched correctly.

We can see that the breakpoint intervals for a breakline of type $(0, \wedge)$ have distance 2 from each other, each having width 1. If the two breaklines of this type would not stab three matching breakpoint intervals, one of them would need to stab two matching intervals and one non-matching interval. As the distance between the vertical lines is equal, and the breakpoint intervals are further apart from each other than their width, there is no way for a breakline to lie in this way. We conclude that all breaklines stab three matching breakpoint intervals. ◁

It also follows from Claims 8.14 to 8.16 that no two breaklines can cross each other between the vertical lines. Neither can a breakline cross a data line. Together with Claim 8.13, we can finally prove Lemma 8.12.

Proof of Lemma 8.12. By Claim 8.13, every breakline must stab three breakpoint intervals of the same gadget. By Claims 8.14 to 8.16, each breakline must stab three matching breakpoint intervals, and therefore the breaklines do not cross any data lines between the three vertical lines.

It remains to show that the data points already ensure that each breakline b is parallel to the two parallel data lines d and d' enclosing it. To this end, consider the parallelogram defined by d, d', v_1, v_3 (see Figure 16) and let j be an output dimension in which b is not erased. Since no other breakline intersects this parallelogram, we obtain that f^j has exactly two linear pieces within the parallelogram, which are separated by b. Moreover, since b stabs matching breakpoint intervals, the three data points on d must belong to one of the pieces. Since these points have the same label, it follows that the gradient of this piece in output dimension j must be orthogonal to d (and, thus, to d' as well). Applying the same argument on the data points on d', we obtain that the gradient of the other piece must be orthogonal to d and d' as well.

This implies that also the difference of the gradients of the two pieces is orthogonal to d and d'. Finally, since b must be orthogonal to this difference of gradients, we obtain that it is parallel to d and d'. □

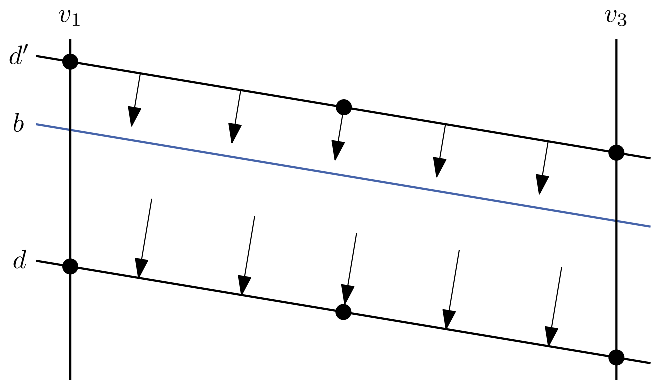


Figure 16: The parallelogram enclosed by the two data lines d, d' and the vertical lines v_1, v_3. The three data points (black) on each data line enforce the gradient in both cells to be orthogonal to the data lines. As a consequence, the breakline b (blue) separating the cells has to be parallel to the data lines.

8.4 Global construction

As a last step in our ∃ℝ-hardness proof of Train-F2NN, we describe the global arrangement of the different gadgets. To this end, fix an arbitrary ETR-Inv instance. See Figure 17 for a visualization.

Variables For each variable X, we build a horizontal variable gadget carrying the value of this variable. We say that this is the *canonical* variable gadget for X. Lemma 8.6 already ensures that $X \in \left[\frac{1}{2}, 2\right]$.

In order to realize the weak data points of the variable gadgets, we add one lower bound gadget each. They are placed parallel to each other, but not parallel to the variable gadgets. Further, their stripes must not intersect any other data points.

Addition The following is done for each addition constraint $X + Y = Z$, next to each other: For each variables involved, we copy the value from its canonical variable gadget to a new variable gadget. To this end, a data point with label 6 is placed on the intersection of the upper measuring line of the canonical variable gadget and the lower measuring line of the new variable gadget (Corollary 8.8).

Further, the three new variable gadgets are positioned such that the correct measuring lines intersect in a common intersection above all horizontal variable gadgets (upper for X, Y and lower for Z). A data point with label 10 at the intersection enforces the addition constraint (Corollary 8.8).

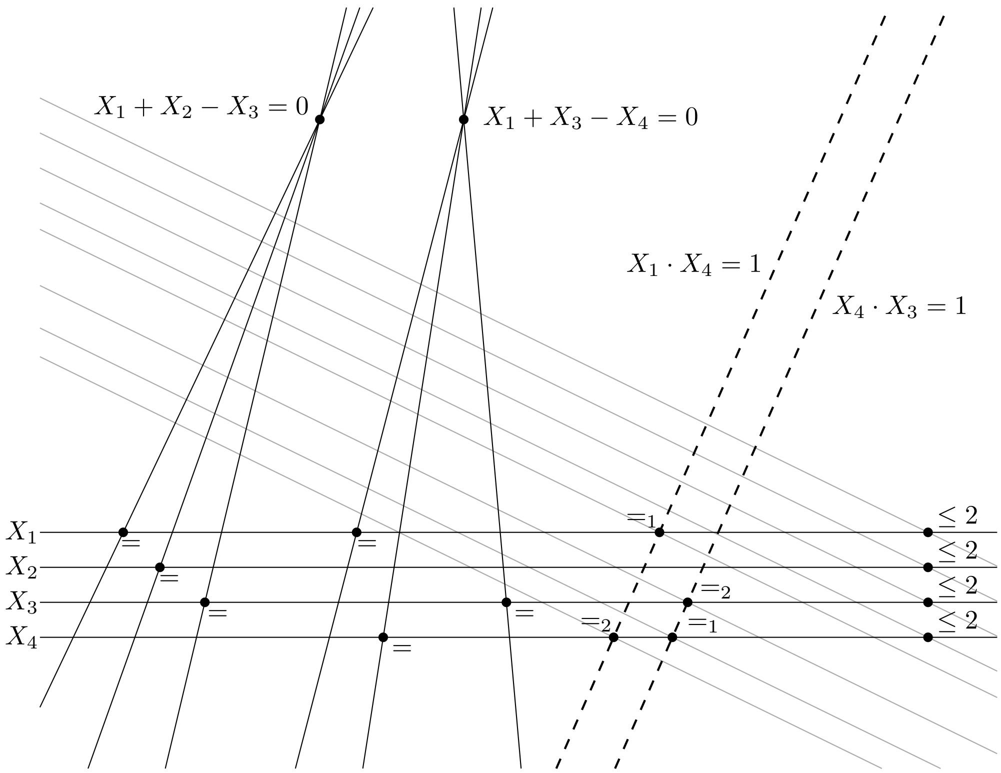


Figure 17: The layout of all gadgets and additional data points for the complete reduction. Each gadget is simplified to a single line for clarity. Solid: Variable gadgets. Dashed: Inversion gadgets. Gray: Lower bound Gadgets. A point with label $=_i$ indicates a copy that is only active in output dimension i.

Inversion The following is done for each inversion constraint $XY = 1$, next to each other: We add an inversion gadget that intersects all canonical variable gadgets. Using weak data points with label 6 in the respective dimensions, we copy X to the first dimension of the inversion gadget, and Y to its second dimension (Corollary 8.8). The inversion constraint itself is then enforced by the non-linear relation of the two slopes of the inversion gadget (Lemma 8.9).

All inversion gadgets are placed parallel to each other.

The weak data points involved are realized by two lower bound gadgets per inversion gadget. We place them parallel to the lower bound gadgets for the variables, again such that their stripes do not intersect any other data points.

Based on the global arrangement of the gadgets, we can finally prove our main theorem, i. e., the ∃ℝ-completeness of TRAIN-F2NN.

Proof of Theorem 3.1. For ∃ℝ-membership we refer to [3] and to Section 7.

For $\exists\mathbb{R}$-hardness, we reduce the $\exists\mathbb{R}$-complete problem ETR-INV to TRAIN-F2NN. Given an instance of ETR-INV, we construct an instance of TRAIN-F2NN as described in the previous paragraphs. We set the target error to $\gamma = 0$.

Let m be the minimum number of breaklines needed to realize all gadgets of the above construction: We need four breaklines per variable gadget (Lemma 8.6), five breaklines per inversion gadget (Lemma 8.9) and three breaklines per lower bound gadget (Lemma 8.10). By Lemma 8.12, no breakline can contribute to different gadgets, so we need exactly that many.

In the remainder of the proof, we show equivalence of the following statements.

(1) The ETR-INV instance is a yes-instance, i. e., there exists a satisfying assignment of the variables.

(2) There exists a continuous piecewise linear function with m breaklines that fits all data points of the constructed TRAIN-F2NN instance. Further, it fulfills the conditions of Lemma 8.2.

(3) The TRAIN-F2NN instance is a yes-instance, i. e., there exists a fully connected two-layer neural network with m hidden ReLU neurons exactly fitting all the data points.

To see that (1) implies (2), assume that there is a satisfying assignment of the ETR-INV instance with all variables in $\left[\frac{1}{2}, 2\right]$. For each variable X, we use $s_X = X + 1$ as the slope of all corresponding variable gadgets and inversion gadgets. The superposition of all these gadgets yields the desired continuous piecewise linear function. It satisfies Lemma 8.2 because, first, the gadgets are built in such a way that functions fitting all data points are zero everywhere except for within the gadgets, and second, the gradient condition is satisfied for each gadget separately and, hence, also for the whole function.

For the other direction, i. e., that (2) implies (1), assume that such a continuous piecewise linear function exists. By Lemmas 8.10 and 8.12, the data points enforce exactly the same continuous piecewise linear function as the conceptual data lines and weak data points would. Then, by Lemmas 8.6 and 8.9, this continuous piecewise linear function has the shape of the gadgets. The fact that all data points are fit implies that the slopes of the variable and inversion gadgets indeed correspond to a satisfying assignment of ETR-INV.

Lemma 8.2 yields that (2) implies (3).

It remains to prove that (3) implies (2). To this end, first note that the function realized by a fully connected two-layer neural network with m hidden ReLU neurons is always a continuous piecewise linear function with at most m breaklines that additionally satisfies the gradient condition by Lemma 8.2. The existence of a $(0, 0)$-cell follows from the fact, that all gadgets are 0 everywhere outside their stripes.

The TRAIN-F2NN instance can be constructed in polynomial time, as the gadgets can be arranged in such a way that all data points (residing on intersections of lines) have coordinates which can be encoded in polynomial length.

To ensure the latter, we make sure that all angles between data lines are such that their sin and cos value are rational numbers with small encoding size. Angles with rational sin and cos values correspond to rational points on the unit circle, which form a group with respect to rotation (i. e., addition of angles). Furthermore, these points are a dense subset of the unit circle.

The number of hidden neurons m is linear in the number of variables and the number of constraints of the ETR-INV instance. The number of data points can be bounded by $10m$, thus the number of hidden neurons is linear in the number of data points.

As can be gathered from Lemmas 8.6, 8.9 and 8.10 and Corollary 8.8, the set of labels used has cardinality 13 as claimed. ◻

Remark 8.17. Note that if the ETR-INV instance is satisfiable, then each variable gadget and inversion gadget in a corresponding solution Θ to the constructed TRAIN-F2NN instance has a maximum slope of 3 in each dimension. Furthermore, no lower bound gadget needs to contribute less than -12 to satisfy its corresponding weak data point. Thus, there must also be a solution Θ', where each lower bound gadget is symmetric, and thus the function $f(\cdot, \Theta')$ is Lipschitz continuous with a low Lipschitz constant L, which in particular does not depend on the given ETR-INV instance. Checking all the different ways how our gadgets intersect, one can verify that $L = 25$ is sufficient.

Remark 8.18. The number of data points and neurons used in the reduction is linear in the number of variables and constraints of the ETR-INV instance. To see this, observe that each variable or constraint only introduces a constant number of gadgets, each gadget consists of a bounded number of data lines and a bounded number of additional data points, and each data line is realized with three data points.

9 Algebraic universality

It remains to prove the algebraic universality of TRAIN-F2NN. Intuitively, it suffices to show that a solution of an ETR-INV instance can be transformed into a solution of the corresponding TRAIN-F2NN instance (and vice versa) using only basic field arithmetic, that is, addition, subtraction, multiplication, and division.

For the following lemma, let Φ be an instance of ETR-INV with k variables, and let N be an instance of TRAIN-F2NN with a total of ℓ weights and biases. Further, N was constructed from Φ via our reduction. We denote by $V(\Phi) \subseteq \mathbb{R}^k$ the set of all satisfying variable assignments. Similarly, let $V(N)$ denote the set of all weight–bias assignments that fit all data points.

Lemma 9.1. *The following holds for any field extension $\mathbb{F}$ of $\mathbb{Q}$.*

$$V(\Phi) \cap \mathbb{F}^k \neq \emptyset \iff V(N) \cap \mathbb{F}^\ell \neq \emptyset.$$

Proof. **"⟹:"** Let $X_1, \ldots, X_n \in V(\Phi) \cap \mathbb{F}^k$ be a satisfying variable assignment for Φ. In our reduction, we place our data points on rational coordinates, and thus all implied data lines can be described by equations with rational coefficients. There exists a unique continuous piecewise linear function f which fits these data points, corresponds to the solution $X_1, \ldots, X_n$, and in which all lower bound gadgets are symmetric. This function can be realized by a fully connected two-layer neural network by Lemma 8.2. The gradients of all cells in this function can be obtained through elementary operations from the values $X_1, \ldots, X_n$ and rational numbers. Furthermore, all breaklines can be described by equations with coefficients derivable from these same numbers. Thus, there exists an assignment $\Theta \in \mathbb{F}^\ell$ of weights and biases for the neural network which realize function f, showing that $\Theta \in V(N) \cap \mathbb{F}^\ell \neq \emptyset$.

"$\Longleftarrow$:" Let $\Theta \in V(N) \cap \mathbb{F}^\ell$ be an assignment of weights and biases fitting all data points of the Train-F2NN instance N. For each variable X of Φ, there is a canonical variable gadget corresponding to X whose slope s_X satisfies $X = s_X - 1$. There is a unique hidden neuron v_i contributing the first breakline of that variable gadget. Using the notation from Section 8.1, the slope of this variable gadget is $a_{2,i} \cdot c_{i,1}$, because the variable gadget is horizontal (implying that $a_{1,i} = 0$) and its output is equal in both output dimensions (implying $c_{i,1} = c_{i,2}$). Thus, $X = a_{2,i} \cdot c_{i,1} - 1$, which is clearly in $\mathbb{F}$. The same holds for all other variables, thus $V(\Phi) \cap \mathbb{F}^k \neq \emptyset$. ◻

Now Theorem 3.2, i. e., the algebraic universality of Train-F2NN, follows directly from the algebraic universality of ETR-Inv (Theorem 8.4) combined with Lemma 9.1.

References

[1] Mikkel Abrahamsen: Covering polygons is even harder. In *Proc. 62nd FOCS*, pp. 375–386. IEEE Comp. Soc., 2021. [doi:10.1109/FOCS52979.2021.00045] 11

[2] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow: The art gallery problem is ∃ℝ-complete. *J. ACM*, 69(1):4:1–70, 2022. [doi:10.1145/3486220] 11, 18

[3] Mikkel Abrahamsen, Linda Kleist, and Tillmann Miltzow: Training neural networks is ∃ℝ-complete. In *Proc. 34th Adv. Neural Info. Proc. Sys. (NeurIPS'21)*, pp. 18293–18306. Curran Assoc., 2021. Available at NeurIPS. 2, 3, 10, 15, 18, 35

[4] Mikkel Abrahamsen and Tillmann Miltzow: Dynamic toolbox for ERTINV, 2019. [arXiv:1912.08674] 5, 18

[5] Reyan Ahmed, Felice de Luca, Sabin Devkota, Stephen Kobourov, and Mingwei Li: Multicriteria scalable graph drawing via stochastic gradient descent, $(SGD)^2$. *IEEE Trans. Visualiz. & Computer Graphics*, 28(6):2388–2399, 2022. Open-access link at the NSF PAR. [doi:10.1109/TVCG.2022.3155564] 7

[6] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee: Understanding deep neural networks with rectified linear units. In *Int. Conf. Learning Representations (ICLR'18)*. ICLR, 2018. Available at OpenReview. 2, 3, 8, 9, 10, 11, 13, 16

[7] Egor Bakaev, Florestan Brunck, Christoph Hertrich, Jack Stade, and Amir Yehudayoff: Better neural network expressivity: Subdividing the simplex. In *Proc. 58th STOC*, pp. 500–507. ACM Press, 2026. [doi:10.1145/3798129.3800768] 11

[8] AINESH BAKSHI, RAJESH JAYARAM, AND DAVID P. WOODRUFF: Learning two layer rectified neural networks in polynomial time. In *Proc. 32nd Ann. Conf. on Learning Theory (COLT'19)*, pp. 195–268. MLR Press, 2019. Available at PMLR. 10

[9] MARIE LOUISA TØLBØLL BERTHELSEN AND KRISTOFFER ARNSFELT HANSEN: On the computational complexity of decision problems about multi-player Nash equilibria. *Theory Computing Sys.*, 66(3):519–545, 2022. [doi:10.1007/s00224-022-10080-1] 11

[10] DANIEL BERTSCHINGER, CHRISTOPH HERTRICH, PAUL JUNGEBLUT, TILLMANN MILTZOW, AND SIMON WEBER: Training fully connected neural networks is ∃ℝ-complete. In *Proc. 36th Adv. Neural Info. Proc. Sys. (NeurIPS'23)*, pp. 36222–36237. Curran Assoc., 2023. Available at NeurIPS. 38

[11] DANIEL BIENSTOCK, GONZALO MUÑOZ, AND SEBASTIAN POKUTTA: Principled deep neural network training through linear programming. *Discr. Optimization*, 49(100795):1–37, 2023. [doi:10.1016/j.disopt.2023.100795] 7, 10

[12] VITTORIO BILÒ AND MARIOS MAVRONICOLAS: ∃ℝ-complete decision problems about (symmetric) Nash equilibria in (symmetric) multi-player games. *ACM Trans. Econ. Comput.*, 9(3):14:1–25, 2021. [doi:10.1145/3456758] 11

[13] MANON BLANC AND KRISTOFFER ARNSFELT HANSEN: Computational complexity of multi-player evolutionarily stable strategies. In *Proc. 16th Comp. Sci. Symp. in Russia (CSR'21)*, pp. 1–17. Springer, 2021. [doi:10.1007/978-3-030-79416-3_1] 11

[14] LENORE BLUM, MIKE SHUB, AND STEVE SMALE: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. AMS*, 21(1):1–46, 1989. [doi:10.1090/S0273-0979-1989-15750-9] 5, 10

[15] DIGVIJAY BOOB, SANTANU S. DEY, AND GUANGHUI LAN: Complexity of training ReLU neural network. *Discr. Optimization*, 44(1):1–16, 2022. [doi:10.1016/j.disopt.2020.100620] 10

[16] CORNELIUS BRAND, ROBERT GANIAN, AND MATHIS ROCTON: New complexity-theoretic frontiers of tractability for neural network training. In *Proc. 36th Adv. Neural Info. Proc. Sys. (NeurIPS'23)*, pp. 56456–56468. Curran Assoc., 2023. Available at NeurIPS. 10

[17] ALON BRUTZKUS AND AMIR GLOBERSON: Globally optimal gradient descent for a ConvNet with Gaussian inputs. In *Proc. 34th Internat. Conf. Machine Learning (ICML'17)*, pp. 605–614. MLR Press, 2017. Available at PMLR. 6

[18] SÉBASTIEN BUBECK AND MARK SELLKE: A universal law of robustness via isoperimetry. *J. ACM*, 70(2):10:1–18, 2023. [doi:10.1145/3578580] 10

[19] PETER BÜRGISSER AND FELIPE CUCKER: Exotic quantifiers, complexity classes, and complete problems. *Found. Computational Math.*, 9(2):135–170, 2009. [doi:10.1007/s10208-007-9006-9] 11

[20] John F. Canny: Some algebraic and geometric computations in PSPACE. In *Proc. 20th STOC*, pp. 460–467. ACM Press, 1988. [doi:10.1145/62212.62257] 5, 6

[21] Jean Cardinal, Stefan Felsner, Tillmann Miltzow, Casey Tompkins, and Birgit Vogtenhuber: Intersection graphs of rays and grounded segments. *J. Graph Algor. & Appl.*, 22(2):273–295, 2018. [doi:10.7155/jgaa.00470] 11

[22] Sitan Chen, Aravind Gollakota, Adam R. Klivans, and Raghu Meka: Hardness of noise-free learning for two-hidden-layer neural networks. In *Proc. 35th Adv. Neural Info. Proc. Sys. (NeurIPS'22)*, pp. 10709–10724. Curran Assoc., 2022. Available at NeurIPS. 10

[23] Sitan Chen, Adam R. Klivans, and Raghu Meka: Learning deep ReLU networks is fixed-parameter tractable. In *Proc. 62nd FOCS*, pp. 696–707. IEEE Comp. Soc., 2021. [doi:10.1109/FOCS52979.2021.00073] 10

[24] Dmitry Chistikov, Stefan Kiefer, Ines Marusic, Mahsa Shirmohammadi, and James Worrell: On restricted nonnegative matrix factorization. In *Proc. 43rd Internat. Colloq. on Automata, Languages, and Programming (ICALP'16)*, pp. 103:1–14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. [doi:10.4230/LIPIcs.ICALP.2016.103] 11

[25] George Cybenko: Approximation by superpositions of a sigmoidal function. *Math. of Control, Signals & Systems*, 2(4):303–314, 1989. [doi:10.1007/BF02551274] 11

[26] Julian D'Costa, Engel Lefaucheux, Eike Neumann, Joël Ouaknine, and James Worrel: On the complexity of the escape problem for linear dynamical systems over compact semialgebraic sets. In *Proc. Internat. Symp. Math. Foundations of Comp. Sci. (MFCS'21)*, pp. 33:1–21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2021. [doi:10.4230/LIPIcs.MFCS.2021.33] 11

[27] Pedro J. de Rezende, Cid C. de Souza, Stephan Friedrichs, Michael Hemmer, Alexander Kröller, and Davi C. Tozoni: Engineering art galleries. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of *LNCS*, pp. 379–417. Springer, 2016. [doi:10.1007/978-3-319-49487-6_12, arXiv:1410.8720] 6

[28] Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis: Approximating the existential theory of the reals. *J. Comput. System Sci.*, 125:106–128, 2022. [doi:10.1016/j.jcss.2021.11.002] 7

[29] Steffen Dereich and Sebastian Kassing: On minimal representations of shallow ReLU networks. *Neural Networks*, 148:121–128, 2022. [doi:10.1016/j.neunet.2022.01.006] 11, 16, 17, 18

[30] Santanu S. Dey, Guany Wang, and Yao Xie: Approximation algorithms for training one-node ReLU neural networks. *IEEE Trans. Signal Processing*, 68:6696–6706, 2020. [doi:10.1109/TSP.2020.3039360] 10

[31] ILIAS DIAKONIKOLAS, SURBHI GOEL, SUSHRUT KARMALKAR, ADAM R. KLIVANS, AND MAHDI SOLTANOLKOTABI: Approximation schemes for ReLU regression. In *Proc. 33rd Ann. Conf. on Learning Theory (COLT'20)*, pp. 1452–1485. MLR Press, 2020. Available at PMLR. 10

[32] MICHAEL G. DOBBINS, LINDA KLEIST, TILLMANN MILTZOW, AND PAWEŁ RZĄŻEWSKI: Completeness for the complexity class ∀∃ℝ and area-universality. *Discr. Comput. Geom.*, 70(1):154–188, 2023. [doi:10.1007/s00454-022-00381-0] 11, 18

[33] RONEN ELDAN AND OHAD SHAMIR: The power of depth for feedforward neural networks. In *Proc. 29th Ann. Conf. on Learning Theory (COLT'16)*, pp. 907–940. Springer, 2016. Available at PMLR. 11

[34] JEFF ERICKSON, IVOR VAN DER HOOG, AND TILLMANN MILTZOW: Smoothing the gap between NP and ∃ℝ. *SIAM J. Comput.*, 53(6):102–138, 2024. [doi:10.1137/20M1385287] 5, 7, 9, 15

[35] VINCENT FROESE AND CHRISTOPH HERTRICH: Training neural networks is NP-hard in fixed dimension. In *Proc. 36th Adv. Neural Info. Proc. Sys. (NeurIPS'23)*, pp. 44039–44049. Curran Assoc., 2023. Available at NeurIPS. 8, 10

[36] VINCENT FROESE, CHRISTOPH HERTRICH, AND ROLF NIEDERMEIER: The computational complexity of ReLU network training parameterized by data dimensionality. *J. Artif. Intell. Res.*, 74:1775–1790, 2022. [doi:10.1613/jair.1.13547] 10

[37] JUGAL GARG, RUTA MEHTA, VIJAY V. VAZIRANI, AND SADRA YAZDANBOD: ∃ℝ-completeness for decision versions of multi-player (symmetric) Nash equilibria. *ACM Trans. Econ. Comput.*, 6(1):1:1–23, 2018. [doi:10.1145/3175494] 11

[38] XAVIER GLOROT, ANTOINE BORDES, AND YOSHUA BENGIO: Deep sparse rectifier neural networks. In *Proc. 14th Internat. Conf. Artificial Intelligence and Statistics (AISTATS'11)*, pp. 315–323. MLR Press, 2011. Available at PMLR. 3, 9

[39] SURBHI GOEL, VARUN KANADE, ADAM R. KLIVANS, AND JUSTIN THALER: Reliably learning the ReLU in polynomial time. In *Proc. 30th Ann. Conf. on Learning Theory (COLT'17)*, pp. 1004–1042. MLR Press, 2017. Available at PMLR. 10

[40] SURBHI GOEL AND ADAM R. KLIVANS: Learning neural networks with two nonlinear layers in polynomial time. In *Proc. 32nd Ann. Conf. on Learning Theory (COLT'19)*, pp. 1470–1499. MLR Press, 2019. Available at PMLR. 10

[41] SURBHI GOEL, ADAM R. KLIVANS, PASIN MANURANGSI, AND DANIEL REICHMAN: Tight hardness results for training depth-2 ReLU networks. In *Proc. 12th Innovations in Theoret. Comp. Sci. Conf. (ITCS'21)*, pp. 22:1–14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2021 (virtual conference). [doi:10.4230/LIPIcs.ITCS.2021.22] 10

[42] SURBHI GOEL, ADAM R. KLIVANS, AND RAGHU MEKA: Learning one convolutional layer with overlapping patches. In *Proc. 35th Internat. Conf. Machine Learning (ICML'18)*, pp. 1783–1791. MLR Press, 2018. Available at PMLR. 10

[43] IAN GOODFELLOW, YOSHUA BENGIO, AND AARON COURVILLE: *Deep Learning*. MIT Press, 2016. Available at MIT. 2, 3, 9

[44] HENRY GOUK, EIBE FRANK, BERNHARD PFAHRINGER, AND MICHAEL J. CREE: Regularisation of neural networks by enforcing Lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021. [doi:10.1007/s10994-020-05929-w] 9

[45] CHRISTIAN HAASE, CHRISTOPH HERTRICH, AND GEORG LOHO: Lower bounds on the depth of integral ReLU neural networks via lattice polytopes. In *Int. Conf. Learning Representations (ICLR'23)*. ICLR, 2023. Available at OpenReview. 11

[46] BORIS HANIN: Universal function approximation by deep neural nets with bounded width and ReLU activations. *Mathematics*, 7(10):992:1–9, 2019. [doi:10.3390/math7100992] 11

[47] BORIS HANIN AND DAVID ROLNICK: Complexity of linear regions in deep networks. In *Proc. 36th Internat. Conf. Machine Learning (ICML'19)*, pp. 2596–2604. MLR Press, 2019. Available at PMLR. 11

[48] BORIS HANIN AND MARK SELLKE: Approximating continuous functions by ReLU nets of minimal width, 2018. [arXiv:1710.11278] 11

[49] SIMON B. HENGEVELD AND TILLMANN MILTZOW: A practical algorithm with performance guarantees for the Art Gallery problem. *Discr. Math. & Theor. Comput. Sci.*, 25(2):1–59, 2023. Preliminary version in SoCG'21. [doi:10.46298/DMTCS.9225] 6

[50] CHRISTOPH HERTRICH, AMITABH BASU, MARCO DI SUMMA, AND MARTIN SKUTELLA: Towards lower bounds on the depth of ReLU neural networks. *SIAM J. Discr. Math.*, 37(2):997–1029, 2023. [doi:10.1137/22M1489332] 11, 16

[51] CHRISTOPH HERTRICH AND LEON SERING: ReLU neural networks of polynomial size for exact maximum flow computation. *Math. Programming*, 210(1):377–406, 2025. [doi:10.1007/s10107-024-02096-x] 11

[52] KURT HORNIK: Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. [doi:10.1016/0893-6080(91)90009-T] 11

[53] JOEY HUCHETTE, GONZALO MUÑOZ, TIAGO SERRA, AND CALVIN TSAY: When deep learning meets polyhedral theory: A survey, 2023. [arXiv:2305.00241] 11

[54] PAUL JUNGEBLUT: On the complexity of Lombardi graph drawing. In *Graph Drawing & Network Visualization (GD'23)*, pp. 180–194. Springer, 2023. [doi:10.1007/978-3-031-49272-3_13] 11

[55] PAUL JUNGEBLUT, LINDA KLEIST, AND TILLMANN MILTZOW: The complexity of the Hausdorff distance. *Discr. Comput. Geom.*, 71(1):177–213, 2024. [doi:10.1007/s00454-023-00562-5] 11

[56] ROSS KANG AND TOBIAS MÜLLER: Sphere and dot product representations of graphs. *Discr. Comput. Geom.*, 47(3):548–569, 2012. [doi:10.1007/s00454-012-9394-8] 11

[57] SAMMY KHALIFE, HONGYU CHENG, AND AMITABH BASU: Neural networks with linear threshold activations: Structure and algorithms. *Math. Programming*, 206:333–356, 2024. [doi:10.1007/s10107-023-02016-5] 9

[58] JAN KRATOCHVÍL AND JIŘÍ MATOUŠEK: Intersection graphs of segments. *J. Combin. Theory–B*, 62(2):289–315, 1994. [doi:10.1006/jctb.1994.1071] 11

[59] SHIYU LIANG AND RAYADURGAM SRIKANT: Why deep neural networks for function approximation? In *Int. Conf. Learning Representations (ICLR'17)*. ICLR, 2017. Available at OpenReview. 11

[60] ANNA LUBIW, TILLMANN MILTZOW, AND DEBAJYOTI MONDAL: The complexity of drawing a graph in a polygonal region. *J. Graph Algor. & Appl.*, 26(4):421–446, 2022. [doi:10.7155/jgaa.00602] 11, 18

[61] JIŘÍ MATOUŠEK: Intersection graphs of segments and ∃ℝ, 2014. [arXiv:1406.2636] 11

[62] COLIN MCDIARMID AND TOBIAS MÜLLER: Integer realizations of disk and segment graphs. *J. Combin. Theory–B*, 103(1):114–143, 2013. [doi:10.1016/j.jctb.2012.09.004] 11

[63] TILLMANN MILTZOW AND REINIER F. SCHMIERMANN: On classifying continuous constraint satisfaction problems. *TheoretiCS*, 3(10):1–54, 2024. Preliminary version in FOCS'22. [doi:10.46298/THEORETICS.24.10] 11

[64] NIKOLAI E. MNËV: The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In OLEG Y. VIRO AND ANATOLY M. VERSHIK, editors, *Topology and Geometry — Rohlin Seminar*, volume 1346 of *Lecture Notes in Mathematics*, pp. 527–543. Springer, 1988. [doi:10.1007/BFb0082792] 11

[65] GUIDO MONTÚFAR, YUE REN, AND LEON ZHANG: Sharp bounds for the number of regions of maxout networks and vertices of Minkowski sums. *SIAM J. Appl. Algebra Geom.*, 6(4):618–649, 2022. [doi:10.1137/21M1413699] 11

[66] GUIDO F. MONTÚFAR, RAZVAN PASCANU, KYUNGHYUN CHO, AND YOSHUA BENGIO: On the number of linear regions of deep neural networks. In *Proc. 27th Adv. Neural Info. Proc. Sys. (NIPS'14)*, pp. 2924–2932. Curran Assoc., 2014. Available at NeurIPS. 11

[67] ANIRBIT MUKHERJEE AND AMITABH BASU: Lower bounds over Boolean inputs for deep neural networks with ReLU gates, 2017. [arXiv:1711.03073] 11, 16

[68] QUYNH NGUYEN, MAHESH CHANDRA MUKKAMALA, AND MATTHIAS HEIN: Neural networks should be wide enough to learn disconnected decision regions. In *Proc. 35th Internat. Conf. Machine Learning (ICML'18)*, pp. 3740–3749. MLR Press, 2018. Available at PMLR. 11

[69] RAZVAN PASCANU, GUIDO MONTÚFAR, AND YOSHUA BENGIO: On the number of inference regions of deep feed forward networks with piece-wise linear activations. In *Int. Conf. Learning Representations (ICLR'14)*. ICLR, 2014. Available at OpenReview. 11

[70] Grant O. Passmore and Paul B. Jackson: Combined decision techniques for the existential theory of the reals. In *Internat. Conf. Intell. Computer Math. (CICM'09)*, pp. 122–137. Springer, 2009. [doi:10.1007/978-3-642-02614-0_14] 6

[71] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein: On the expressive power of deep neural networks. In *Proc. 34th Internat. Conf. Machine Learning (ICML'17)*, pp. 2847–2854. MLR Press, 2017. Available at PMLR. 11

[72] Daniel Richardson: Some undecidable problems involving elementary functions of a real variable. *J. Symbolic Logic*, 33(4):514–520, 1969. [doi:10.2307/2271358] 9

[73] Jürgen Richter-Gebert and Günter M. Ziegler: Realization spaces of 4-polytopes are universal. *Bull. AMS*, 32(4):403–412, 1995. [doi:10.1090/S0273-0979-1995-00604-X] 11

[74] Itay Safran and Ohad Shamir: Depth-width tradeoffs in approximating natural functions with neural networks. In *Proc. 34th Internat. Conf. Machine Learning (ICML'17)*, pp. 2979–2987. MLR Press, 2017. Available at PMLR. 11

[75] Marcus Schaefer: Complexity of some geometric and topological problems. In *Graph Drawing (GD'09)*, pp. 334–344. Springer, 2009. [doi:10.1007/978-3-642-11805-0_32] 6, 11

[76] Marcus Schaefer: Realizability of graphs and linkages. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, pp. 461–482. Springer, 2013. [doi:10.1007/978-1-4614-0110-0_24] 11

[77] Marcus Schaefer: Complexity of geometric k-planarity for fixed k. *J. Graph Algor. & Appl.*, 25(1):29–41, 2021. [doi:10.7155/jgaa.00548] 11

[78] Marcus Schaefer: RAC-drawability is $\exists\mathbb{R}$-complete and related results. *J. Graph Algor. & Appl.*, 27(9):803–841, 2023. [doi:10.7155/jgaa.00646] 11

[79] Marcus Schaefer, Jean Cardinal, and Tillmann Miltzow: The existential theory of the reals as a complexity class: A compendium. In János Pach and Géza Tóth, editors, *Courses in Discrete and Computational Geometry*, volume 31 of *Bolyai Society Mathematical Studies*, pp. 167–313. Springer, Cham, 2026. [doi:10.1007/978-3-032-10503-5_5, arXiv:2407.18006] 11

[80] Marcus Schaefer and Daniel Štefankovič: Fixed points, Nash equilibria, and the existential theory of the reals. *Theory Computing Sys.*, 60(2):172–193, 2017. [doi:10.1007/s00224-015-9662-0] 11

[81] Marcus Schaefer and Daniel Štefankovič: Beyond the existential theory of the reals. *Theory Computing Sys.*, 68(2):195–226, 2024. [doi:10.1007/s00224-023-10151-x] 11

[82] Marcus Schaefer and Daniel Štefankovič: The complexity of tensor rank. *Theory Computing Sys.*, 62(5):1161–1174, 2018. [doi:10.1007/s00224-017-9800-y] 11

[83] THIAGO SERRA, CHRISTIAN TJANDRAATMADJA, AND SRIKUMAR RAMALINGAM: Bounding and counting linear regions of deep neural networks. In *Proc. 35th Internat. Conf. Machine Learning (ICML'18)*, pp. 4558–4566. MLR Press, 2018. Available at PMLR. 11

[84] SHAI SHALEV-SHWARTZ AND SHAI BEN-DAVID: *Understanding Machine Learning: From Theory to Algorithms*. Cambridge Univ. Press, 2014. [doi:10.1017/CBO9781107298019] 7, 10

[85] YAROSLAV SHITOV: The complexity of positive semidefinite matrix factorization. *SIAM J. Optim.*, 27(3):1898–1909, 2017. [doi:10.1137/16M1080616] 11

[86] PETER W. SHOR: Stretchability of pseudolines is NP-hard. In *Appl. Geom. & Discr. Math.*, pp. 531–554. Amer. Math. Soc., 1991. [doi:10.1090/dimacs/004/41] 5, 6, 11

[87] JACK STADE: The point-boundary art gallery problem is ∃ℝ-hard. In *Proc. 41st Internat. Symp. Comput. Geom. (SoCG'25)*, pp. 74:1–23. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2025. [doi:10.4230/LIPIcs.SoCG.2025.74, arXiv:2210.12817] 11

[88] MORITZ STARGALLA, CHRISTOPH HERTRICH, AND DANIEL REICHMAN: The computational complexity of counting linear regions in ReLU neural networks. In *Proc. 38th Adv. Neural Info. Proc. Sys. (NeurIPS'25)*, pp. 125970–125988. Curran Assoc., 2025. Available at NeurIPS. 11

[89] MATUS TELGARSKY: Benefits of depth in neural networks. In *Proc. 29th Ann. Conf. on Learning Theory (COLT'16)*, pp. 1517–1539. Springer, 2016. Available at PMLR. 11

[90] LESLIE G. VALIANT: A theory of the learnable. *Comm. ACM*, 27(11):1134–1142, 1984. [doi:10.1145/1968.1972] 7

[91] GAL VARDI, GILAD YEHUDAI, AND OHAD SHAMIR: On the optimal memorization power of ReLU neural networks. In *Int. Conf. Learning Representations (ICLR'22)*. ICLR, 2022. Available at OpenReview. 11

[92] DMITRY YAROTSKY: Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017. [doi:10.1016/j.neunet.2017.07.002] 11

[93] CHULHEE YUN, SUVRIT SRA, AND ALI JADBABAIE: Small ReLU networks are powerful memorizers: A tight analysis of memorization capacity. In *Proc. 32nd Adv. Neural Info. Proc. Sys. (NeurIPS'19)*. Curran Assoc., 2019. Available at NeurIPS. 11

[94] CHIYUAN ZHANG, SAMY BENGIO, MORITZ HARDT, BENJAMIN RECHT, AND ORIOL VINYALS: Understanding deep learning (still) requires rethinking generalization. *Comm. ACM*, 64(3):107–115, 2021. [doi:10.1145/3446776] 8, 11

[95] LIWEN ZHANG, GREGORY NAITZAT, AND LEK-HENG LIM: Tropical geometry of deep neural networks. In *Proc. 35th Internat. Conf. Machine Learning (ICML'18)*, pp. 5824–5832. MLR Press, 2018. Available at PMLR. 11, 16

[96] XIAO-DONG ZHANG: Complexity of neural network learning in the real number model. In *Workshop on Physics and Computation*, pp. 146–150. IEEE Comp. Soc., 1992. [doi:10.1109/PHYCMP.1992.615511] 10

AUTHORS

Daniel Bertschinger
At time of submission:
Institute of Theoretical Computer Science
Department of Computer Science
ETH Zurich
Zurich, Switzerland

Christoph Hertrich
Tenure-track professor
Applied Discrete Mathematics
Department of Liberal Arts and Social Sciences
University of Technology Nuremberg
Nuremberg, Germany
christoph.hertrich@utn.de.de
https://christophhertrich.gitlab.io/
Parts of the work were completed while Christoph Hertrich was affiliated with the London School of Economics, UK, the Goethe-Universität Frankfurt, Germany, and the Université libre de Bruxelles, Belgium.

Paul Jungeblut
Institute of Theoretical Informatics
Department of Computer Science
Karlsruhe Insitute of Technology
Karlsruhe, Germany
paul.jungeblut@partner.kit.edu
https://i11www.iti.kit.edu/en/members/paul_jungeblut/index

Tillmann Miltzow
Assistant professor
Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
t.miltzow@uu.nl
https://sites.google.com/view/miltzow/home

Simon Weber
At the time of submission:
Ph. D. student
Institute of Theoretical Computer Science
Department of Computer Science
ETH Zurich
Zurich, Switzerland
simon.lukas.weber@alumni.ethz.ch
https://people.inf.ethz.ch/siweber/

ABOUT THE AUTHORS

DANIEL BERTSCHINGER was a researcher at ETH Zürich at the time of this project, advised by Emo Welzl. He left the academic world and is pursuing new professional directions.

CHRISTOPH HERTRICH is a tenure-track professor for Applied Discrete Mathematics at the University of Technology Nuremberg. He is particularly enthusiastic about applying methods from polyhedral geometry, combinatorial optimization, and computational complexity to the theory of neural networks. A survey talk with some favorite problems can be found here. Previously, in 2023/24, he was a postdoc at the Université libre de Bruxelles advised by Samuel Fiorini and partially funded through an individual Marie Skłodowska-Curie fellowship. The time in Brussels was paused when he acted as a substitute professor for discrete mathematics at the Goethe-Universität Frankfurt in the winter semester 2023/24. Furthermore, in 2022/23, he was a postdoc with László Végh at LSE in London. He earned his Ph. D. at TU Berlin under the supervision of Martin Skutella (2018-22), and his BSc and MSc at TU Kaiserslautern (now RPTU) under the supervision of Sven O. Krumke (2013-18). His brother Johannes works in mathematical image processing.

Paul Jungeblut graduated from the Karlsruhe Institute of Technology in 2024 under the supervision of Torsten Ueckerdt. His thesis was about the computational complexity of geometric problems in relation to the first-order theory of the reals. Since 2024, he has been working as a software engineer.

Tillmann Miltzow is an Assistant Professor at Utrecht University. He received his Ph. D. in Computer Science from Free University Berlin, where he was advised by Günter Rote. His research lies in theoretical computer science, with a focus on computational geometry and complexity theory. Much of his work studies the existential theory of the reals as a unifying framework for understanding the complexity of geometric and combinatorial problems. A recurring theme in his research is the interplay between discrete and continuous models of computation.

Simon Weber obtained his Ph. D. at ETH Zurich under the supervision of Bernd Gärtner (2021–25). His main research interests lay in combinatorial frameworks for optimization problems, in particular Unique Sink Orientations, but also in computational geometry, computational complexity, and topology. Since finishing his thesis, he has been working in the cyber security industry.