# When Deep Learning Meets Polyhedral Theory: A Survey

Joey Huchette
Google Research, USA

Gonzalo Muñoz
Universidad de O'Higgins, Chile

Thiago Serra
Bucknell University, USA

Calvin Tsay
Imperial College London, UK

September 2023

**Abstract**

In the past decade, deep learning became the prevalent methodology for predictive modeling thanks to the remarkable accuracy of deep neural networks in tasks such as computer vision and natural language processing. Meanwhile, the structure of neural networks converged back to simpler representations based on piecewise constant and piecewise linear functions such as the Rectified Linear Unit (ReLU), which became the most commonly used type of activation function in neural networks. That made certain types of network structure —such as the typical fully-connected feedforward neural network— amenable to analysis through polyhedral theory and to the application of methodologies such as Linear Programming (LP) and Mixed-Integer Linear Programming (MILP) for a variety of purposes. In this paper, we survey the main topics emerging from this fast-paced area of work, which brings a fresh perspective to understanding neural networks in more detail as well as to applying linear optimization techniques to train, verify, and reduce the size of such networks.

## 1 Introduction

Deep learning has continuously achieved new landmarks in varied areas of artificial intelligence for the past decade. Examples of those areas include predictive tasks in computer vision (Krizhevsky et al., 2012, Ciresan et al., 2012, Szegedy et al., 2015, He et al., 2016, Xie et al., 2020b), natural language processing (Sutskever et al., 2014, Peters et al., 2018, Radford et al., 2018, Devlin et al., 2019), and speech recognition (Hinton et al., 2012, Graves and Jaitly, 2014, Park et al., 2019). The artificial neural networks behind such feats are being used in many applications, and there is a growing interest for analytical insights to help design such networks and then to leverage the model that they have learned. For the most commonly used types of neural networks, some of those results and

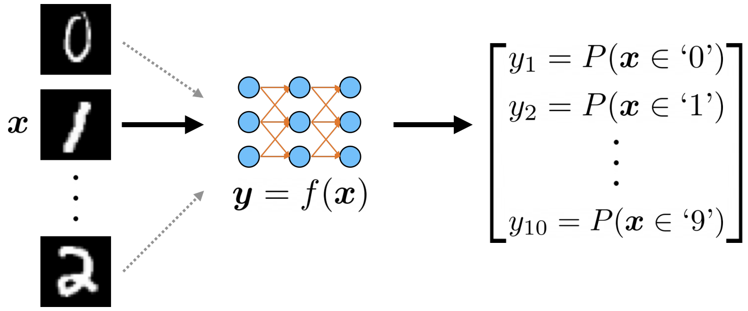
$\boldsymbol{x}$
$\boldsymbol{y} = f(\boldsymbol{x})$
$y_1 = P(\boldsymbol{x} \in \text{‘0’})$
$y_2 = P(\boldsymbol{x} \in \text{‘1’})$
$y_{10} = P(\boldsymbol{x} \in \text{‘9’})$

Figure 1: Example classification task on the MNIST database of handwritten digits, in which the image of a handwritten digit is given as input and the probability of that digit being from each possible class is provided as output.

methods are coming from operations research tools such as polyhedral theory and associated optimization techniques such as Linear Programming (LP) and Mixed-Integer Linear Programming (MILP). Among other things, these connections with mathematical optimization may help us understand what neural networks can represent, how to train them, and how to make them more compact. For example, consider the popular task of classifying images (Figure 1); polyhedral theory and associated optimization techniques may help us answer questions such as the following. How should we train the classifier model? How large should it be? How robust to perturbations is it?

## 1.1 What neural networks can model

We can essentially think of artificial neural networks as functions mapping an input $\boldsymbol{x}$ from a given domain to an output $\boldsymbol{y}$ for a given application. For the classification task in Figure 1, inputs $\boldsymbol{x}$ correspond to images from the dataset, and $\boldsymbol{y}$ to the associated predicted labels, or probabilities for labels describing the content of those images. The basic units of neural networks mimic biological neurons in that they receive inputs from adjacent units, transform those inputs, and may produce an output to subsequent units of the network. In other words, every unit is also a function, and in fact the output of most units is defined by the composition of a nonlinear function with a linear function. The nonlinear function is often denoted as the *activation function* in analogy to how a biological neuron is triggered to send a signal to adjacent neurons when the stimulus caused by the input exceeds a certain activation threshold. Such non-linearity is behind the remarkable expressiveness of neural networks.

This model was pioneered by McCulloch and Pitts (1943), who considered a thresholding function for activation that is now often denoted as the Linear Threshold Unit (LTU). That activation is also the basis of the classic *perceptron*

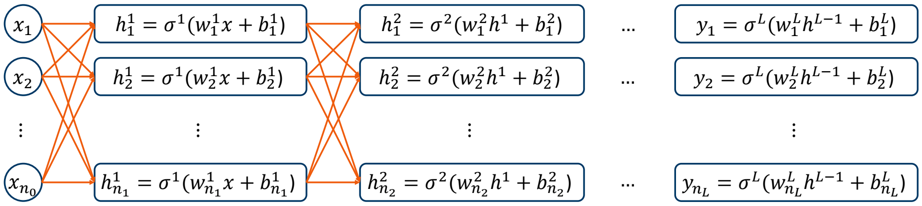
$x_1$
$h_1^1 = \sigma^1(w_1^1 x + b_1^1)$
$h_1^2 = \sigma^2(w_1^2 h^1 + b_1^2)$
...
$y_1 = \sigma^L(w_1^L h^{L-1} + b_1^L)$
$x_2$
$h_2^1 = \sigma^1(w_2^1 x + b_2^1)$
$h_2^2 = \sigma^2(w_2^2 h^1 + b_2^2)$
...
$y_2 = \sigma^L(w_2^L h^{L-1} + b_2^L)$
⋮
$x_{n_0}$
$h_{n_1}^1 = \sigma^1(w_{n_1}^1 x + b_{n_1}^1)$
$h_{n_2}^2 = \sigma^2(w_{n_2}^2 h^1 + b_{n_2}^2)$
...
$y_{n_L} = \sigma^L(w_{n_L}^L h^{L-1} + b_{n_L}^L)$

Figure 2: Mapping from $\boldsymbol{x} \in \mathbb{R}^{n_0}$ to $\boldsymbol{y} \in \mathbb{R}^{n_L}$ through a feedforward neural network with $L$ layers, layer widths $\{n_l\}_{l \in \mathbb{L}}$, and activation functions $\{\sigma_l\}_{l \in \mathbb{L}}$.

algorithm by Rosenblatt (1957), which yields a binary classifier of the form

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0; \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

for an input $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and with parameters $\boldsymbol{w} \in \mathbb{R}^{n_0}$ and $b \in \mathbb{R}$. Those parameters are chosen by optimizing the predictions for a given task, as discussed below and in Section 5. The term *single-layer perceptron* is used for a neural network consisting of a set of such units processing the same input in parallel. The term *multi-layer perceptron* is used for a generalization of this concept, by which the output of a *layer* —a set of units with same input— is the input for a subsequent layer. This perceptron terminology has also been loosely applied to neural networks with other activation functions.

More generally, neural networks that successively transform inputs through an ordered sequence of layers are also denoted *feedforward networks*. The layers that do not produce the final output of the neural network are denoted *hidden layers*. For a network with $L$ layers, we denote $n_l$ as the number of units —or *width*— of layer $l \in \mathbb{L} := \{1, 2, \ldots, L\}$ and $h_i^l$ as the output of the $i$-th unit in layer $l$, where $i \in \{1, 2, \ldots, n_l\}$. The output of a unit is given by

$$h_i^l = \sigma^l \left( \boldsymbol{w}_i^l \cdot \boldsymbol{h}^{l-1} + b_i^l \right), \tag{2}$$

where the *weights* $\boldsymbol{w}_i^l \in \mathbb{R}^{n_{l-1}}$ and the *bias* $b_i^l \in \mathbb{R}$ are parameters of the unit. Those parameters can be aggregated across the layer as the matrix $\boldsymbol{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$ and the vector $\boldsymbol{b}^l \in \mathbb{R}^{n_l}$. The vector $\boldsymbol{h}^{l-1} \in \mathbb{R}^{n_{l-1}}$ represents the aggregated outputs from layer $(l-1)$. The activation function $\sigma^l : \mathbb{R} \to \mathbb{R}$ is applied by the units in layer $l$. These definitions implicitly assume that $n_0$ is the size of the network input $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and that $\boldsymbol{h}^0$ and $\boldsymbol{x}$ are the same. Figure 2 illustrates the operation of a feedforward network as described above.

## 1.2 How neural networks are obtained and evaluated

In resemblance to how other models for *supervised learning* in machine learning are obtained, we can *train* a neural network for a given task by adjusting its behavior with respect to the examples of a *training set* and then evaluate the final trained network on a *test set*. Both of these sets consist of inputs for which the correct output $\hat{y}$ is known. We can define an objective function to model a measure of distance between the output $y$ and the correct output $\hat{y}$, which is typically denoted as the *loss function*, and then iteratively update parameters such as $\{\boldsymbol{W}^l\}_{l \in \mathbb{L}}$ and $\{\boldsymbol{b}^l\}_{l \in \mathbb{L}}$ to minimize that loss function over the training set. A common objective function is the square error $\|y - \hat{y}\|^2$ summed over the points in the training set. The test set contains a separate collection of inputs and their outputs, which is used to evaluate the trained neural network with examples that were not seen during training. A good performance on the test set may indicate that the trained neural network is able to *generalize* beyond the seen examples, whereas a bad performance may suggest that it *overfits* for the training set. Neural networks also have *hyperparameters* that are often chosen manually and do not change during training, such as the *depth* $L$, the widths of the layers $\{n_l\}_{l \in \mathbb{L}}$, and the activation functions used in each layer $\{\sigma^l\}_{l \in \mathbb{L}}$. Different models can be produced by varying the hyperparameters. In such a case, a *validation set* disjoint from the training and test sets can be used to compare models with different hyperparameters. Whereas the validation set may serve as a benchmark to different trained models corresponding to different choices of hyperparameters, the test set can only be used to evaluate a single neural network chosen among those evaluated with the validation set. The emergent field of *neural architecture search* —recently surveyed by Elsken et al. (2019)— concerns with automatically choosing such hyperparameters.

One of the key factors for the success of deep learning is that first-order methods for continuous optimization can be effectively applied to train deep networks. The interest in neural networks first vanished due to negative results in the Perceptrons book by Minsky and Papert (1969), which showed that single-layer perceptrons cannot represent functions such as the Boolean XOR. However, moving to multi-layer perceptrons capable of expressing the Boolean XOR as well as other more expressive models would require a clever training strategy. Hence, the interest was regained with papers that popularized the use of *backpropagation*, such as Rumelhart et al. (1986) and LeCun et al. (1989). Note that backpropagation was first discussed much earlier in the context of networks by Linnainmaa (1970) and of neural networks explicitly by Werbos (1974). The backpropagation algorithm calculates the derivative of the loss function with respect to each neural network parameter by applying the chain rule through the units of the neural network, which is considerably more efficient than explicitly evaluating the derivative of each network parameter. Consequently, neural networks are generally trained with gradient descent methods in which the parameters are updated sequentially from the output to the input layer in each step. In fact, most algorithms for training neural networks are based on Stochastic Gradient Descent (SGD), which is a form of the stochastic approximation through sampling pioneered by Robbins and Monro (1951). SGD approximates the partial derivatives of the loss function at each step by using only a subset of the data in order to make the training process more efficient. Examples of popular SGD algorithms include momentum (Polyak, 1964), Adam (Kingma and Ba, 2014), and Nesterov Adaptive Gradient

(Sutskever et al., 2013) —the later inspired by Nesterov (1983). Interestingly, however, we generally cannot guarantee convergence to a global optimum with gradient descent due to the nonconvexity of the loss function. Nevertheless, neural networks trained with adequately parameterized SGD algorithms tend to generalize well.

## 1.3 Why nonlinearity is important in artificial neurons

The nonlinearity of the activation function leads to such nonconvexity of the loss function. However, as we will see in Section 3, that same nonlinearity enables the neural network to represent more complex functions as a whole. In fact, removing such nonlinearities by using an identity activation function $\sigma^l(u) = u \; \forall l \in \mathbb{L}$ would reduce the entire neural network to an affine transformation of the form $f(x) = \boldsymbol{W}^L(\boldsymbol{W}^{L-1}\left(\ldots\left(\boldsymbol{W}^2\left(\boldsymbol{W}^1 x + \boldsymbol{b}^1\right) + \boldsymbol{b}^2\right) + \ldots\right) + \boldsymbol{b}^{L-1}) + \boldsymbol{b}^L$. Hence, a feedforward network without nonlinear activation functions is equivalent to a linear regression model. However, in that case we can easily obtain such a model without resorting to neural networks and backpropagation: the loss function is convex and the optimal solution is given by a closed formula, such as in least squares regression. In contrast, neural networks with a single hidden layer of arbitrary width have been long known to be universal function approximators for a broad variety of activation functions (Cybenko, 1989, Funahashi, 1989, Hornik et al., 1989), as well as for ReLU more recently (Yarotsky, 2017). These results have also been extended to the converse case of limited width but arbitrarily large depth (Lu et al., 2017, Hanin and Sellke, 2017, Park et al., 2021a).

Although nonlinear activation functions are important for obtaining more complex models, these functions do not need to be overly complex to produce good results. In the past, it was common practice to use sigmoid functions for activation (LeCun et al., 1998). Those are monotonically increasing functions that approach finite values for arbitrarily large positive and negative inputs, such as the standard logistic function $\sigma(u) = \frac{1}{1+e^{-u}}$ and the hyperbolic tangent $\sigma(u) = \tanh(u)$. In the present, the most commonly used activation function is the Rectified Linear Unit (ReLU) $\sigma(u) = \max\{0, u\}$ (LeCun et al., 2015, Ramachandran et al., 2018), which was proposed by Hahnloser et al. (2000) and first applied to neural networks by Nair and Hinton (2010). The popularity of ReLU is in part due to experiments by Nair and Hinton (2010) and Glorot et al. (2011) showing that this simpler form of activation yields competitive results. Thinking back in terms of the analogy with biological neurons, we say that a ReLU is *active* when the output is positive and *inactive* when the output is zero. ReLUs have a linear output behavior on the inputs associated with the same ReLUs being active and inactive; this property also holds for other piecewise linear and piecewise constant functions that are used as activation functions in neural networks. Table 1 lists some of the most commonly used activation functions of that kind. For more comprehensive lists of activation functions, including several other variations based on ReLU, we refer to Dubey et al. (2021) and Tao et al. (2022).

Table 1: Main piecewise constant and piecewise linear activation functions.

| **Name** | **Function** | **Reference** |
|---|---|---|
| LTU | $\sigma(u) = \begin{cases} 1 & \text{if } u > 0 \\ 0 & \text{if } u \leq 0 \end{cases}$ | McCulloch and Pitts (1943) |
| ReLU | $\sigma(u) = \max\{0, u\}$ | Hahnloser et al. (2000), Nair and Hinton (2010) |
| leaky ReLU | $\sigma(u) = \begin{cases} u & \text{if } u > 0 \\ \varepsilon u & \text{if } u \leq 0 \end{cases}$<br>($\varepsilon$ is small and fixed) | Maas et al. (2013) |
| parametric ReLU | $\sigma(u) = \begin{cases} u & \text{if } u > 0 \\ au & \text{if } u \leq 0 \end{cases}$<br>($a$ is a trainable parameter) | He et al. (2015) |
| hard tanh | $\sigma(u) = \begin{cases} 1 & \text{if } u > 1 \\ u & \text{if } -1 \leq u \leq 1 \\ -1 & \text{if } u < -1 \end{cases}$ | Collobert (2004) |
| hard sigmoid | $\sigma(u) = \begin{cases} 1 & \text{if } u > \frac{1}{2} \\ u + \frac{1}{2} & \text{if } -\frac{1}{2} \leq u \leq \frac{1}{2} \\ 0 & \text{if } u < -\frac{1}{2} \end{cases}$ | Courbariaux et al. (2015) |
| max pooling | $\sigma(u_1, \ldots, u_k) = \max\{0, u_1, \ldots, u_k\}$<br>(each $u_i$ is the output of another neuron) | Weng et al. (1992) |
| maxout | $\sigma(u_1, \ldots, u_k) = \max\{u_1, \ldots, u_k\}$<br>(each $u_i$ is an affine function) | Goodfellow et al. (2013) |

## 1.4 When deep learning meets polyhedral theory

It is commonly accepted in machine learning that a simpler model is preferred if it trains as well as a more complex one, since a simpler model is less likely to overfit. Conveniently, the successful return of neural networks to relatively simpler activation functions prepared the ground for deep learning to meet polyhedral theory. In other words, we are now able to analyze and leverage neural networks through the same lenses and tools that have been successfully used for linear and discrete optimization in operations research for many decades. We explain this connection in more detail and some lines of research that it has opened up in Section 2.

## 1.5 Scope of this survey and related work

The interplay between mathematical optimization and machine learning has also been discussed by other recent surveys. Bengio et al. (2021) review the use of machine learning in mathematical optimization, whereas Gambella et al. (2021) formulate mathematical optimization problems with the main focus of obtaining machine learning models, such as by training neural networks. A similar scope has been previously surveyed by Curtis and Scheinberg (2017) and Bottou et al. (2018). Our survey complements those by focusing exclusively on neural networks while outlining how linear optimization can be used more broadly in that context, from network training and verification to model embedding and compression, as well as refined through formulation strengthening. In addition, we illustrate how polyhedral theory can ground the use of such linear formulations and also provide a more nuanced understanding of the discriminative ability of neural networks.

The presentation in this survey is centered on *feedforward rectifier networks*. These are very commonly used networks with only ReLU activations and for which most polyhedral results and applications of linear optimization are known. The focus on a single type of neural network is intended to help the reader capture the intuition behind different developments and understand the nuances involved. Despite our focus on *fully-connected* models, which are those in which every unit is connected to all units in the subsequent layer, there are many variants of interest with fewer or different types of connection that can be interpreted as a special case of fully-connected models. For example, the units of Convolutional Neural Networks (CNNs or ConvNets) (Fukushima, 1980) have local connectivity: only a subset of adjacent units defines the output of each unit in the next layer, and the same parameters are used to define the output of different units. In fact, multiple *filters* of parameters can be applied to a set of adjacent units through the output of different units in the next layer. CNNs are often applied to identify and aggregate the same local features in different parts of a picture, and we can interpret them as a special case of feedforward networks. Another common variant, the Residual Network (ResNet) (He et al., 2016), includes *skip connections* that directly connect units in nonadjacent layers. Those connections can be emulated by adding units passing their outputs through the intermediary layers. Hence, many of the results and applications discussed along the survey are relevant to other variants (e.g., LTU and maxout activations, or those other connectivity patterns), and we also provide references to more specific results and applications involving them.

We also discuss the extent to which other variants remain relevant or can be analyzed through the same lenses. For example, *feedback connections* in *recurrent networks* (Little, 1974, Hopfield, 1982) allow the output of a unit to be used as an input of units in previous layers. Recurrent networks such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) produce outputs that depend on their internal state, and they may consequently process sequential inputs with arbitrary length. While feedback connections may not be emulated with a feeforward network, we discuss in the following paragraph how recurrent networks have been replaced with great success by attention mechanisms, which are implemented with feedforward networks. In the realm of variants that remain relevant, it is very common to apply a different type of activation to the output layer of the network, such as the layer-wise softmax function $\sigma : \mathbb{R}^{n_L} \to \mathbb{R}^{n_L}$ in which $\sigma(u)_i = e^{u_i} / \sum_{j=1}^{n_L} e^{u_j} \ \forall i \in \{1, \ldots, n_L\}$ (Bridle, 1990), which is used to normalize a multidimensional output as a probability distribution. While softmax is not piecewise linear, we describe how its output can also be analyzed from a polyhedral perspective.

**Other uses of deep learning** Deep learning is also being used in machine learning beyond the realm of supervised learning. In *unsupervised learning*, the focus is on drawing inferences from unlabeled datasets. For example, Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) have been used to generate realistic images using a pair of neural networks. One of these networks is a *discriminator* trained to identify elements from a dataset and the other is a *generator* aiming to mislead the discriminator with synthetic inputs that could be classified as belonging to the dataset.

In *reinforcement learning*, the focus is on modeling agents that can interact with their environment through actions and associated rewards. Examples of such applications include neural networks designed for the navigation of self-driving vehicles (Gao et al., 2020) and for playing Atari games (Mnih et al., 2015), more contemporary electronic games such as Dota 2 (OpenAI et al., 2019) and StarCraft II (Vinyals et al., 2017), and the game of Go (Silver et al., 2017) at levels that are either better or at least comparable to human players.

A more recent and popular example are generative transformers (Radford et al., 2018), such as DALL·E 2 (Ramesh et al., 2022) producing realistic images from text prompts in mid-2022 and ChatGPT (OpenAI, 2022) producing realistic dialogues with users in early 2023, the latter belonging to the fast growing family of large language models. They are based on replacing architectures based on feedback connections, such as LSTM, with the attention mechanisms aimed at scoring the relevance of past states (Bahdanau et al., 2015), which is the foundation of the transformer architecture (Vaswani et al., 2017).

**Further reading** For a historical perspective on neural networks, we recommend Schmidhuber (2015). For a recent and broad introduction to the fundamentals of deep learning, we recommend Zhang et al. (2023). For other forms of measuring model complexity in neural networks, we refer to Hu et al. (2021).

# 2 The Polyhedral Perspective

A feedforward rectifier network models a piecewise linear function (Arora et al., 2018) in which every such piece is a polyhedron (Raghu et al., 2017), and represents a special case among neural networks modeling piecewise polynomials (Balestriero and Baraniuk, 2018). Therefore, training a rectifier network is equivalent to performing a piecewise linear regression, and we can potentially interpret such neural networks in terms of what happens in each piece of the function that they model. However, we are only beginning to answer some of the questions entailed by such a remark. In this survey, we discuss how insights on this subject may help us answer the following questions.

1. Which piecewise linear functions can or cannot be obtained from training a neural network given its architecture?

2. Which neural networks are more susceptible to adversarial exploitation?

3. Can we integrate the model learned by a neural network into a broader decision-making problem for which we want to find an optimal solution?

4. Is it possible to obtain a smaller neural network that models exactly the same function as another trained neural network?

5. Can we exploit the polyhedral geometry present in neural networks in the training phase?

6. Can we efficiently incorporate extra structure when training neural network, such as linear constraints over the weights?

The first question complements the universal approximation results for neural networks. Namely, there is a limit to what functions can be well approximated when limited computational resources are translated into constraints on the depth and width of the layers of neural networks that can be used in practice. The functions that can be modeled depend on the particular choice of hyperparameters subject to the computational resources available, and in the long run that may also lead to a more principled approach for the choice of hyperparameters than the current approaches of neural architecture search. In Section 3, we analyze how a rectifier network partitions the input space into pieces in which it behaves linearly, which we denote as *linear regions*. We discuss the geometry of linear regions, the effect of parameters and hyperparameters on the number of linear regions of a neural network, and the extent to which such number of linear regions relates to the accuracy of the network.

The second question relies on formal verification methods to evaluate the robustness of neural networks, which can be approached with mathematical optimization formulations that are also relevant for the third and fourth questions. Such formulations are convenient since a direct inspection of every piece of the function modeled by a neural network is prohibitive given how quickly their number scale with the size of the network. The linear behavior of the network for every choice of active and inactive units implies that we can use linear formulations with binary variables corresponding to the activation of units to model trained neural networks using MILP. Therefore, we are able to solve a variety of optimization problems over a trained neural network, such as the neural network verification problem, identifying the range of outputs for each ReLU of the network, and modeling a trained neural network as part of a larger decision-making problem. In Section 4, we discuss how to formulate optimization problems over a trained neural network, the applications of such formulations, and the progress toward obtaining stronger formulations that scale more easily with the network size.

The fifth and sixth questions involve the training procedure of a DNN, where linear programming tools have been applied to partially answer them. In Section 5, we overview these developments. In terms of the fifth question —exploiting polyhedrality in training neural networks— we describe algorithms that use the polyhedral geometry induced by activation sets to solve training problems. We also cover a recently proposed polyhedral construction that can approximately encode multiple training problems at once, showing a strong relationship across training problems that arise from different datasets, for a fixed architecture. Additionally, we review some recent uses of mixed-integer linear programming in the training phase as an alternative to SGD when the weights are required to be integer. Regarding the sixth question —the incorporation of extra structure when training— we review multiple approaches that have included techniques related to linear programming within SGD to impose a desirable structure when training, or to find better step-lengths in the execution of SGD.

# 3 The Linear Regions of a Neural Network

Every piece of the piecewise linear function modeled by a neural network is a linear region, and —without loss of generality— we can think of it as a polyhedron. In this section, we define a linear region, exemplify how they can be so numerous, and what may affect their count in a neural network. We also discuss the practical implications of such insights, as well as other related forms of analyzing the ability of a neural network to represent expressive models.

**Definition 1** *A linear region corresponds to the set of points from the input space that activates the same units along the neural network, and hence can be characterized by the set $\mathbb{S}^l$ of units that are active in each layer $l \in \mathbb{L}$.*

Since a neural network behaves uniformly over a linear region, the latter is the smallest finite scale in which we can analyze its behavior. If we restrict the domain of a neural network to a linear region $\mathbb{I} \subseteq \mathbb{R}^{n_0}$, then the neural network behaves as an affine transformation $\boldsymbol{y}_{\mathbb{I}} : \mathbb{I} \to \mathbb{R}^{n_L}$ of the form $\boldsymbol{y}_{\mathbb{I}}(\boldsymbol{x}) = \boldsymbol{T}\boldsymbol{x} + \boldsymbol{t}$ with a matrix $\boldsymbol{T} \in \mathbb{R}^{n_L \times n_0}$ and a vector $\boldsymbol{t} \in \mathbb{R}^{n_L}$ that are directly defined by the network parameters and the set of neurons that are activated by any input $\boldsymbol{x} \in \mathbb{I}$. For a small perturbation $\varepsilon$ to some input $\overline{\boldsymbol{x}} \in \mathbb{I}$ such that $\overline{\boldsymbol{x}} + \varepsilon \in \mathbb{I}$, the network output for $\bar{x} + \varepsilon$ is given by $\boldsymbol{y}_{\mathbb{I}}(\overline{\boldsymbol{x}} + \varepsilon)$. While it is possible that two adjacent regions defined in such way correspond to the same affine transformation, thinking of each linear region as having a distinct signature of active units makes it easier to analyze them.

The number of linear regions defined by a neural network is one form with which we can measure the complexity of the models that it can represent (Bengio, 2009). Hence, if a more complex model is desired, we may want to design a neural network that can potentially define more linear regions. On the one hand, the number of linear regions may grow exponentially on the depth of a neural network. On the other hand, such a number depends on the interplay between network parameters and hyperparameters. As we consider how the inputs from adjacent linear regions are evaluated, the change to the affine transformation can be characterized in algebraic and geometric terms. Understanding such changes may help us grasp how a neural network is capable of telling its inputs apart, including what are the sources of the complexity of the model.

For neural networks in which the activation function is not piecewise linear, Bianchini and Scarselli (2014) have used more elaborate topological measures to compare the expressiveness of shallow and deep neural networks. Hu et al. (2020b) followed a closer approach by producing a linear approximation neural network in which the number of linear regions can be counted.

## 3.1 The combinatorial aspect of linear regions

One of the most striking aspects about analyzing a neural network in terms of its linear regions is how quickly such number grows. Early work on this topic by Pascanu et al. (2014) and Montúfar et al. (2014) have drawn two important observations. First, that it is possible to construct simple deep neural networks with a number of linear regions that grows exponentially in the depth. Second, that the number of linear regions can be exponential in the number of neurons alone.

The first observation comes from analyzing the role of ReLUs in a very simple setting. Namely, that of a neural network in which we regard every layer as having a single input in the $[0, 1]$ domain, which is produced by combining the outputs of the units from the preceding layer, as illustrated by Example 1.

**Example 1** *Consider a neural network with input $x$ from the domain $[0, 1]$ and layers having 4 neurons with ReLU activation. For the first layer, assume that the output of the neurons are given by the following functions: $f_1(x) = \max\{4x, 0\}$, $f_2(x) = \max\{8x - 2, 0\}$, $f_3(x) = \max\{6.5x - 3.25, 0\}$, and $f_4(x) = \max\{12.5x - 11.25, 0\}$. In other words, $\boldsymbol{h}_i^1 = f_i(x)$ $\forall i \in \{1, 2, 3, 4\}$. For the* *subsequent layers, assume that the outputs coming from the previous layer are combined through the function $F(x) = f_1(x) - f_2(x) + f_3(x) - f_4(x)$, which substitutes $x$ as the input to the next layer; upon which the same set of functions $\{f_i(x)\}_{i=1}^4$ defines the output of the next layer. In other words, $\boldsymbol{h}_i^l = f_i(F(\boldsymbol{h}^{l-1})) = f_i(h_1^{l-1} - h_2^{l-1} + h_3^{l-1} - h_4^{l-1})\ \forall i \in \{1,2,3,4\}, l \in \mathbb{L} \setminus \{1\}$.*

*When the output of the units in the first layer is combined as $F(x)$, we obtain a zigzagging function with 4 slopes in the $[0,1]$ domain, each of which defining a bijection between segments of the input —namely, $[0, 0.25]$, $[0.25, 0.5]$, $[0.5, 0.9]$, and $[0.9, 1.0]$— and the image $[0,1]$. The effect of repeating such structure in the second layer is that of composing $F(x)$ with itself, with 4 slopes being produced within each of those 4 initial segments. Hence, the number of slopes —and therefore of linear regions— in the output of such a neural network with L layers of activation functions is $4^L$, which implies an exponential growth on depth.*

*The network structure and the parameters of the neurons in the first two layers are illustrated in Figure 3; the set of functions $\{f_i(x)\}_{i=1}^4$ and the combined outputs of the first two layers —$F(x)$ and $F(F(x))$— are illustrated in Figure 4.*

In Example 1, every neuron changes the slope of the resulting function once it becomes active, in which we purposely alternate between positive and negative slopes once the function reaches either 0 or 1, respectively. By selecting the network parameters accordingly, Montúfar et al. (2014) were the first to show that a layer with $n$ ReLUs can be used to create a zigzagging function with $n$ slopes on the $[0,1]$ domain, with the image along every slope also corresponding to the interval $[0,1]$. Consequently, stacking $L$ of such layers results in a neural network with $n^L$ linear regions.

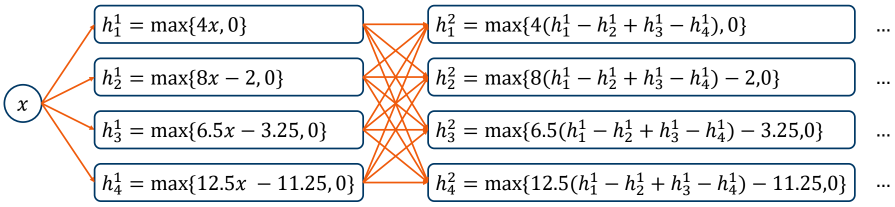
$x$
$h_1^1 = \max\{4x, 0\}$
$h_2^1 = \max\{8x - 2, 0\}$
$h_3^1 = \max\{6.5x - 3.25, 0\}$
$h_4^1 = \max\{12.5x - 11.25, 0\}$
$h_1^2 = \max\{4(h_1^1 - h_2^1 + h_3^1 - h_4^1), 0\}$
$h_2^2 = \max\{8(h_1^1 - h_2^1 + h_3^1 - h_4^1) - 2, 0\}$
$h_3^2 = \max\{6.5(h_1^1 - h_2^1 + h_3^1 - h_4^1) - 3.25, 0\}$
$h_4^2 = \max\{12.5(h_1^1 - h_2^1 + h_3^1 - h_4^1) - 11.25, 0\}$
...
...
...
...

Figure 3: Mapping from the input $x \in [0,1]$ to the intermediary output $\boldsymbol{h}^2 \in [0,1]^4$ through the first two layers of a neural network in which the number of linear regions growths exponentially on the depth, as described in Example 1. The parameters of subsequent layers are the same as those in the second layer.

The second observation —that the number of linear regions can grow exponentially in the number of neurons alone— comes from the interplay between the parts of the input space in which each the units are active, especially in higher-dimensional spaces. This is based on some geometric observations that we discuss in Section 3.3. Even for a *shallow* network —i.e., the number of layers being $L = 1$— such a number of linear regions may approach $2^n$, which corresponds to every possible activation set $\mathbb{S} \subseteq \{1, \ldots, n\}$ defining a nonempty

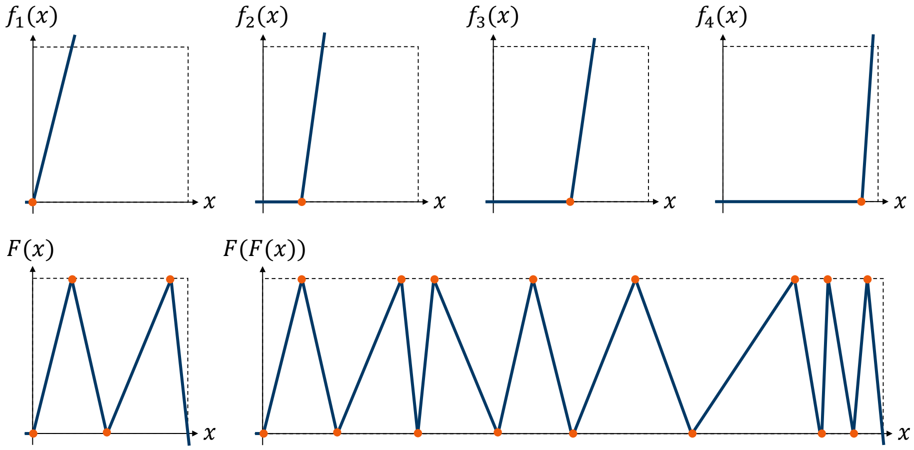

Figure 4: Set of activation functions $\{f_i(x)\}_{i=1}^4$ of the units in the first layer and combined outputs of the first two layers —$F(x) = f_1(x) - f_2(x) + f_3(x) - f_4(x)$ for the first and $F(F(x))$ for the second— of a neural network in which the number of linear regions grows exponentially on the depth, as described in Example 1.

linear region. However, as we discuss later, that is not always the case due to architectural choices such as the number of layers and their width.

### 3.2 The algebra of linear regions

Given the activation sets $\{\mathbb{S}^l\}_{l \in \mathbb{L}}$ denoting which neurons are active for each layer of the neural network, we can explicitly describe the affine transformation $\boldsymbol{y}_{\mathbb{I}}(\boldsymbol{x}) = \boldsymbol{T}\boldsymbol{x} + \boldsymbol{t}$ associated with the corresponding linear region $\mathbb{I}$. For every activation set $\mathbb{S}^l$, layer $l$ defines an affine transformation of the form $\Omega^{\mathbb{S}^l}(\boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l)$, where $\Omega^{\mathbb{S}^l}$ is a diagonal $n_l \times n_l$ matrix in which $\Omega^{\mathbb{S}^l}_{ii} = 1$ if $i \in \mathbb{S}^l$ and $\Omega^{\mathbb{S}^l}_{ii} = 0$ otherwise. Hence, the matrix $\boldsymbol{T}$ and vector $\boldsymbol{t}$ are as follows:

$$\boldsymbol{T} = \prod_{l=1}^{L} \Omega^{\mathbb{S}^l} \boldsymbol{W}^l,$$

$$\boldsymbol{t} = \sum_{l_1=1}^{L} \left( \prod_{l_2=l_1+1}^{L} \Omega^{\mathbb{S}^{l_2}} \boldsymbol{W}^{l_2} \right) \Omega^{\mathbb{S}^{l_1}} \boldsymbol{b}^{l_1}.$$

On a side note, Takai et al. (2021) proposed a related metric for networks modeling piecewise linear functions by counting the number of distinct functions among linear regions upon equivalence through isometric affine transformation.

Each linear region is associated with a polyhedron, and we can describe the union of polyhedra $\mathcal{D}$ on the space $(\boldsymbol{x}, \boldsymbol{h}^1, \ldots, \boldsymbol{h}^L)$ that covers the entire input space $x$ of the neural network as follows:

$$\mathcal{D} = \bigvee_{(\mathbb{S}^1,\ldots,\mathbb{S}^L)\subseteq\{1,\ldots,n_1\}\times\ldots\times\{1,\ldots,n_L\}} \left( \begin{array}{cc} \boldsymbol{w}_i^l \cdot \boldsymbol{h}^{l-1} + b_i^l \geq 0 & \forall l \in \mathbb{L}, i \in \mathbb{S}^l \\ h_i^l = \boldsymbol{w}_i^l \cdot \boldsymbol{h}^{l-1} + b_i^l & \forall l \in \mathbb{L}, i \in \mathbb{S}^l \\ \boldsymbol{w}_i^l \cdot h^{l-1} + b_i^l \leq 0 & \forall l \in \mathbb{L}, i \notin \mathbb{S}^l \\ h_i^l = 0 & \forall l \in \mathbb{L}, i \notin \mathbb{S}^l \end{array} \right).$$

Such partitioning entails an overlap between adjacent linear regions when $\boldsymbol{w}_i^l\boldsymbol{h}^{l-1}+b_i^l = 0$, i.e., at the boundary in which unit $i$ in layer $l$ is active in one region and inactive in another. Nevertheless, for any input $\overline{\boldsymbol{x}}$ associated with a point at such a boundary between two linear regions $\mathbb{I}_1$ and $\mathbb{I}_2$, it holds that $\boldsymbol{y}_{\mathbb{I}_1}(\overline{\boldsymbol{x}}) = \boldsymbol{y}_{\mathbb{I}_2}(\overline{\boldsymbol{x}})$ even if those affine transformations are not entirely identical since the output of the neural network is continuous. More importantly, such overlap implies that each term of $\mathcal{D}$ is defined using only equalities and nonstrict inequalities, and therefore that each linear region corresponds to polyhedra in the extended space $(\boldsymbol{x}, \boldsymbol{h}^1, \ldots, \boldsymbol{h}^L)$. Consequently, those linear regions also define polyhedra if projected to the input space $\boldsymbol{x}$, since by using Fourier-Motzkin elimination (Fourier, 1826, Motzkin, 1936) we can obtain a polyhedral description of the linear region in $\boldsymbol{x}$. Moreover, the interior of those polyhedra are disjoint. If one of those polyhedra does not have an interior, which means that it is not full-dimensional, then that linear region lies entirely on the boundary of other linear regions. In such a case, we do not regard it as a proper linear region. By looking at the geometry of those linear regions from a different perspective in Section 3.3 and understanding its impact on the number of linear regions in Section 3.4, we will see that many terms of $\mathcal{D}$ may actually be empty.

The optimization over the union of polyhedra is the subject of disjunctive programming, which has contributed to the development of stronger formulations and better algorithms to solve discrete optimization problems. These are formulated as MILPs as well as more general types of problems in recent years (Balas, 2018), including generalized disjunctive programming for Mixed-Integer Non-Linear Programming (MINLP) (Raman and Grossmann, 1994, Grossmann and Ruiz, 2012). One of such contributions is the generation of valid inequalities to strengthen MILP formulations, which are also denoted as cutting planes, through the lift-and-project technique (Balas et al., 1993, 1996). In fact, we can develop stronger formulations for optimization problems involving neural networks through the lenses of disjunctive programming, as we discuss later in Section 4.2.

## 3.3 The geometry of linear regions

Another form of looking at the geometry of linear regions is through their transformation along the layers of the neural network. Namely, we can think of the input space as initially being partitioned by the units of the first layer, and then each resulting linear region being further partitioned by the subsequent layers. In that sense, we can think of every layer as a particular form of "slicing" the input. In fact, a layer may slice each linear region that is defined by the preced-

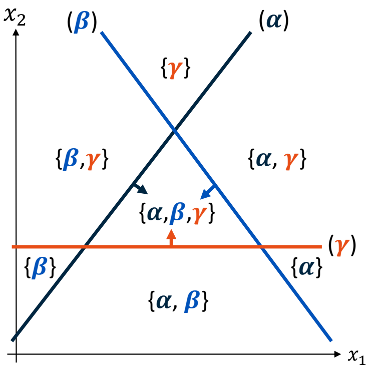

Figure 5: Linear regions defined by the shallow neural network described in Example 2. Every line corresponds to the activation hyperplane of a different neuron, which is given by $\alpha$, $\beta$, and $\gamma$ in parentheses. The arrow next to each line points to the half space in which the inputs activate that neuron. Every linear region has a subset of $\{\alpha, \beta, \gamma\}$ as its corresponding activation set.

ing layer in a different way due to which neurons are active or not in previous layers.

Let us begin by illustrating how a given layer $l \in \mathbb{L}$ partitions its input space $\boldsymbol{h}^{l-1}$. Every neuron $i$ in layer $l$ is associated with an *activation hyperplane* of the form $\boldsymbol{w}_i^l \cdot \boldsymbol{h}^{l-1} + b_i^l = 0$, which divides the possible inputs of its layer into an open half-space in which the unit is active ($\boldsymbol{w}_i^l \cdot \boldsymbol{h}^{l-1} + b_i^l > 0$) and a closed half-space in which the unit is inactive ($\boldsymbol{w}_i^l \cdot \boldsymbol{h}^{l-1} + b_i^l \leq 0$). These hyperplanes define the boundary between adjacent linear regions, and the arrangement of such hyperplanes for a given layer $l \in \mathbb{L}$ determines how that layer partitions the $\boldsymbol{h}^{l-1}$ space. In other words, every input in $\boldsymbol{h}^{l-1}$ can be located with respect to each of those hyperplanes, which corresponds to the activation set of the linear region to which it belongs. However, not every activation set out of the $2^{n_l}$ possible ones maps to a nonempty region of the input space. In the case of Example 2, there is no linear region in which the activation set is empty.

**Example 2** *Consider a neural network with domain $\boldsymbol{x} \in \mathbb{R}^2$ and a single layer having 3 neurons $\alpha$, $\beta$, and $\gamma$ with outputs given as follows: $h_\alpha^1 = \max\{2.3x_1 - 1.9x_2 + 0.6, 0\}$, $h_\beta^1 = \max\{-0.9x_1 - 0.7x_2 + 4.8, 0\}$, and $h_\gamma^1 = \max\{0x_1 + 3x_2 - 5, 0\}$. These neurons define the activation hyperplanes ($\alpha$) $2.3x_1 - 1.9x_2 + 0.6 = 0$, ($\beta$) $-0.9x_1 - 0.7x_2 + 4.8 = 0$, and ($\gamma$) $0x_1 + 3x_2 - 5 = 0$ in the space $\boldsymbol{x}$, which are illustrated along with the activation sets of the linear regions in Figure 5.*

*Instead of $2^3$ linear regions corresponding to each possible activation set defined by a subset of the neurons in $\{\alpha, \beta, \gamma\}$, the arrangement of such hyperplanes produces 7 linear regions, which is in fact the maximum number of 2-dimensional regions that can be defined by drawing 3 lines on a plane.*

The maximum number of full-dimensional regions resulting from a partitioning defined by $n$ hyperplanes depends on the dimension $d$ of the space in which those hyperplanes are defined (Zaslavsky, 1975). That number never exceeds $\sum\limits_{i=1}^{\min\{d,n\}} \binom{n}{i}$. Such bound only coincides with $2^n$ if $d \geq n$; otherwise, as illustrated in Example 2, that number can be smaller. As observed by Hanin and Rolnick (2019b), that bound is $O\left(\frac{n^d}{d!}\right)$ when $n \gg d$.

In fact, the above bound is all that we need to determine the maximum number of linear regions in shallow networks. While not every shallow network may define as many linear regions, it is always possible to put the hyperplanes in what is called a *general position* in order to reach that bound. Thus, the maximum number of linear regions defined by a shallow network is $\sum\limits_{i=0}^{\min\{n_0,n_1\}} \binom{n_1}{n_0}$.

For the polyhedron associated with each linear region, being in general position implies that each vertex lies on exactly $d$ activation hyperplanes. For context, the converse situation in linear programming —having more hyperplanes active on a vertex than the space dimension— characterizes degeneracy.

In the case of deep networks, the partitioning of each linear region by the subsequent layers is based on the output of that linear region. This affects the shape and the number of the linear regions defined by the following layers, which may vary between adjacent linear regions due to which units are active or inactive from one linear region to another, as illustrated in Example 3.

**Example 3** *Consider a neural network with domain $\boldsymbol{x} \in \mathbb{R}^2$ and 2 layers having 2 neurons each —say neurons $\alpha$ and $\beta$ in layer 1, and neurons $\gamma$ and $\delta$ in layer 2— with outputs given as follows: $h^1_\alpha = \max\{2.3x_1 - 1.9x_2 + 1.5, 0\}$, $h^1_\beta = \max\{-0.9x_1 - 0.7x_2 + 5, 0\}$, $h^2_\gamma = \max\{0.4h^1_1 - 3.1h^1_2 + 4, 0\}$, $h^2_\delta = \max\{-0.6h^1_1 - 1.6h^1_2 + 5, 0\}$. These neurons define the activation hyperplanes ($\alpha$) $2.3x_1 - 1.9x_2 + 1.5 = 0$ and ($\beta$) $-0.9x_1 - 0.7x_2 + 5 = 0$ in the $\boldsymbol{x}$ space and the activation hyperplanes ($\gamma$) $0.4h^1_1 - 3.1h^1_2 + 4 = 0$ and ($\delta$) $-0.6h^1_1 - 1.6h^1_2 + 5 = 0$ in the $\boldsymbol{h}^1$ space, which are illustrated along with the activation sets of the linear regions in the first two plots of Figure 6. The third plot illustrates the linear regions jointly defined by the two layers in terms of the input space $\boldsymbol{x}$.*

*The third plot is repeated in Figure 7, in which the shape of each linear region $\mathbb{I}$ is filled in accordance to the dimension of the image of $\bar{\boldsymbol{y}}_{\mathbb{I}}(\boldsymbol{x})$ —the output of the neural network for each linear region $\mathbb{I}$.*

Example 3 highlights two important aspects about the structure of linear regions in deep neural networks. First, the linear regions defined by a neural network with multiple layers are different because activation hyperplanes after the first layer may look “bent” from the input space $x$, such as with the inflections of hyperplanes ($\gamma$) and ($\delta$) in the third plot of Figure 6 from one linear region defined by the first layer to another. This partitioning of the input space would not be possible with a single layer.

By comparing side by side the first and the third plots of Figure 6, we can see how every linear region of a given layer of a neural network may be partitioned

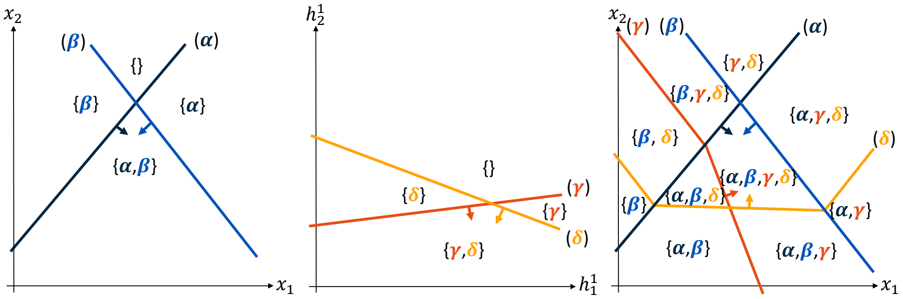

Figure 6: Linear regions defined by the 2 layers of the neural network described in Example 3, following the same notation as in Figure 5. The first and second plots show the linear regions and corresponding activation sets defined by the first and the second layers in terms of their input spaces ($\boldsymbol{x}$ and $\boldsymbol{h}^1$). The third plot shows the linear regions defined by the combination of the 2 layers and the union of their activation sets in terms of the input space of the first layer ($\boldsymbol{x}$).

differently by the following layer. When defined in terms of the input space $\boldsymbol{x}$, the hyperplanes associated with the second layer differ across the linear regions defined by the first layer because each of those linear regions is associated with a different affine transformation from $\boldsymbol{x}$ to $\boldsymbol{h}^1$. Hence, the activation hyperplanes of layer $l$ may break each linear region from layer $l-1$ differently. To every linear region defined by the hyperplane arrangement in the $\boldsymbol{h}^{l-1}$ space there is a linear transformation $\boldsymbol{h}^{l-1} = \Omega^{\mathbb{S}^{l-1}}(\boldsymbol{W}^{l-1}\boldsymbol{h}^{l-2} + \boldsymbol{b}^{l-1})$ to the points of that linear region based on the set of active neurons $\mathbb{S}^{l-1}$. Consequently, inputs in the $\boldsymbol{h}^{l-1}$ space that are associated with different linear regions are transformed differently to the $\boldsymbol{h}^l$ space, and therefore the form in which those linear regions are further partitioned by layer $l$ is not the same when seen from the $\boldsymbol{h}^{l-1}$ space.

Second, some combinations of activation sets of multiple layers do not correspond to linear regions even if the activation hyperplanes are in general position with respect to each layer. For each layer, the first two plots of Figure 6 show that every activation set corresponds to a nonempty region of the layer input. However, not every pair of such activation sets would define a nonempty linear region for the neural network. For example, the linear region of the first layer associated with the activation set $\mathbb{S}^1 = \{\}$ defines a linear region in $\boldsymbol{x}$ which is always mapped to $\boldsymbol{h}^1 = 0$, and thus only corresponds to activation set $\mathbb{S}^2 = \{\gamma, \delta\}$ in the second layer because both units are active for such input. Thus, no linear region in $\boldsymbol{x}$ is associated with only the units in sets $\{\}$, $\{\gamma\}$, and $\{\delta\}$ being active —i.e., there is no linear region such that $\mathbb{S}^1 \cup \mathbb{S}^2 = \{\}, \{\gamma\}, \text{or} \{\delta\}$.

More generally, the number of units that is active on each linear region defined by the first layer also imposes a geometric limit to how that linear region can be further partitioned. If only one unit is active at a layer, that means that the output of the layer within that linear region has dimension 1, and, consequently, the subsequent hyperplane arrangements within that linear region are limited to a 1-dimensional space. For the network in the example,

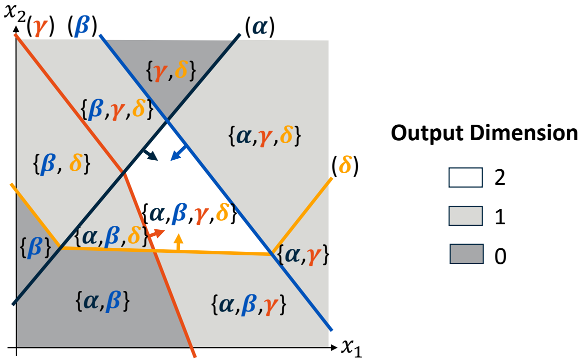

Figure 7: Dimension of the image of the affine function $\boldsymbol{y}_{\mathbb{I}}(\boldsymbol{x})$ associated with each linear region $\mathbb{I}$ defined by the neural network described in Example 3. The linear regions are the same illustrated in the third plot of Figure 6.

we thus expect no more than $\sum_{i=0}^{1} \binom{2}{i} = 3$ linear regions being defined instead of $2^2 = 4$ when only one unit is active. In fact, that is precisely the number of subdivisions by the second layer of the linear region defined by activation set $\mathbb{S}^1 = \{\beta\}$ from the first layer.

For every linear region defined by layer $l$ with an activation set $\mathbb{S}^l$, the dimension of the output of the corresponding transformation $\Omega_{\mathbb{S}^l}(\boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l)$ is at most $|\mathbb{S}^l|$ since $\text{rank}(\Omega_{\mathbb{S}^l}) = |\mathbb{S}^l|$. Hence, the dimension of the output of every linear region defined by a rectifier network is upper bounded by its smallest activation set across all layers. This phenomenon was first identified by Serra et al. (2018) as the *bottleneck effect*. In neural networks with uniform width, this phenomenon leads to a surprising consequence: the number of linear regions with full-dimensional output is at most one. There are also consequences to the maximum number of linear regions that can be defined, as we discuss later.

### 3.3.1 The geometry of decision regions

It is also common to study what inputs are associated with each class by a neural network. The set of inputs associated with the same class define a *decision region*. Difficulties in modeling functions such as the Boolean XOR in shallow networks are related to limitations on the form of the decision regions, which may be limited by the depth of the neural network. For example, Makhoul et al. (1989) showed that two layers suffice to obtain disconnected decision regions.

The softmax layer is typically used for the output of neural networks trained on classification problems, in which the largest output corresponds to the class to which the input is associated. In rectifier networks coupled with a softmax layer, the decision regions can also be defined by polyhedra. Although the output of the softmax layer is not piecewise linear, its largest output corresponds to its largest input. Hence, every linear region $\mathbb{I}$ defined by layers 1 to $L-1$ is partitioned by the softmax layer into decision regions where $\boldsymbol{h}_i^{L-1} \geq \boldsymbol{h}_j^{L-1} \; \forall j \neq i$

for each class $i$ associated with the input $\boldsymbol{h}_i^{L-1}$ to the softmax layer. Therefore, each decision region of a rectifier networks consist of a union of polyhedra.

In fact, we may say further in the typical setting where no hidden layer is wider than the input —i.e., $n_0 \geq n_l \; \forall l \in \mathbb{L}$: Nguyen et al. (2018) showed that at least one layer $l \in \mathbb{L}$ must be such that $n_l > n_0$ for the network to present disconnected decision regions; and Grigsby and Lindsey (2022) showed that, for an input size $n_0 \geq 2$, the decision regions are either empty or unbounded.

## 3.4 The number of linear regions

We have seen conditions that affect the number of linear regions both positively and negatively. We discuss these and other analytical results in Section 3.4.1, and then discuss work on counting linear regions in practice in Section 3.4.2.

### 3.4.1 Analytical results

At least three lines of work on analytical results have brought important insights. The first line is based on constructing networks with a large number of linear regions, which leads to lower bounds on the maximum number of linear regions. The second line is based on showing how the network architecture —in particular its hyperparameters— may impact the hyperplane arrangements defined by the layers, which leads to upper bounds on the maximum number of linear regions. The third line is based on characterizing the parameters of neural networks according to how they are initialized and updated along training, which leads to results on the expected number of linear regions for such networks.

**Lower bounds** The lower bounds on the maximum number of linear regions are obtained through a careful choice of network parameters aimed at increasing the number of linear regions. In some cases, they also depend on particular choices of hyperparameters. We present them by order of refinement in Table 2.

The first lower bound was introduced by Pascanu et al. (2014) and then improved by those authors with a new construction technique in Montúfar et al. (2014). In fact, Example 1 shows the case in which $n_0 = 1$ for the technique in Montúfar et al. (2014). While a different construction is proposed by Telgarsky (2015), subsequent developments in the literature have been based on Montúfar et al. (2014).

The lower bound by Arora et al. (2018) is based on a different technique to construct a first wide layer based on zonotopes, which is then followed by the same layers as in Montúfar et al. (2014). The first lower bound by Serra et al. (2018) reflects a slight change to the technique used by Montúfar et al. (2014), which in terms of Example 1 corresponds to using $n$ neurons to define $n+1$ instead of $n$ slopes on $[0, 1]$. The second lower bound by Serra et al. (2018) extends that of Arora et al. (2018) by changing in the same way the construction of the subsequent layers of the network.

Table 2: Lower bounds on the maximum number of linear regions defined by a neural network.

| Reference | Bound and conditions |
|---|---|
| Pascanu et al. (2014) | $\left( \prod_{l=1}^{L-1} \left\lfloor \frac{n_l}{n_0} \right\rfloor \right) \sum_{i=0}^{n_0} \binom{n_L}{i}$ |
| Montúfar et al. (2014) | $\left( \prod_{l=1}^{L-1} \left\lfloor \frac{n_l}{n_0} \right\rfloor^{n_0} \right) \sum_{i=0}^{n_0} \binom{n_L}{i}$, where $n_l \geq n_0 \ \forall l \in \mathbb{L}$ |
| Telgarsky (2015) | $2^{\frac{L-3}{2}}$, where $n_i = 1$ for $i$ odd, $n_i = 2$ for $i$ even, and $L - 3$ divides by 2 |
| Arora et al. (2018) | $2 \sum_{j=0}^{n_0-1} \binom{m-1}{j} w^{L-1}$, where $2m = n_1$ and $w = n_l \ \forall l \in \mathbb{L} \setminus \{1\}$ |
| Serra et al. (2018) | $\left( \prod_{l=1}^{L-1} \left( \left\lfloor \frac{n_l}{n_0} \right\rfloor + 1 \right)^{n_0} \right) \sum_{i=0}^{n_0} \binom{n_L}{i}$, where $n_l \geq 3n_0 \ \forall l \in \mathbb{L}$ |
| Serra et al. (2018) | $2 \sum_{j=0}^{n_0-1} \binom{m-1}{j} (w+1)^{L-1}$, where $2m = n_1$ and $w = n_l \geq 3n_0 \ \forall l \in \mathbb{L} \setminus \{1\}$ |

**Upper bounds** The upper bounds on the maximum number of linear regions are obtained by primarily considering changes to the geometry of the linear regions from one layer to another, as previously outlined and revisited below. We present those with a close form by order of refinement in Table 3.

Pascanu et al. (2014) established the connection between linear regions and hyperplane arrangements, which lead to the tight bound for shallow networks based on Zaslavsky (1975) for activation hyperplanes in general position. Montúfar et al. (2014) defined the first bound for deep networks based on enumerating all activation sets. The subsequent upper bounds extended the result by Pascanu et al. (2014) to deep networks by considering its successive application through the sequence of layers of the network.

In the case of *deep* networks, where $L > 1$, we need to consider how the linear regions defined up to a given layer of the network can be further partitioned by the next layers. We start by assuming that every linear region defined by the first $l - 1$ layers is then subdivided into the maximum possible number of linear regions defined by the activation hyperplanes of layer $l$. That leads to the bound in Raghu et al. (2017), which is implicit in their proof of an asymptotic bound of $O(n^{n_0 L})$, where $n$ is used as the width of every layer. However, there are many ways in which this bound can be refined upon careful examination. First, the dimension of the input of layer $l$ —i.e., the output of layer $l - 1$— within each linear region is never larger than the smallest dimension among layers 1 to $l$, since for every linear region we have an affine transformation between inputs and outputs of each layer (Montúfar, 2017). Second, the dimension of the input coming through each linear region is in fact bounded by the smallest number of active units in each of the previous layers (Serra et al., 2018). This leads to a tight upper bound for $n_0 = 1$, since it matches the lower bound in Serra et al. (2018). Finally, the activation hyperplane of some units may not partition the linear regions because all possible inputs to the unit are in the same half-space, and in some of those cases the unit may never produce a positive output. For the number $k$ of active units in a given layer $l$, we can use the network parameters to calculate the maximum number of units that can be active in the next layer, $\mathcal{A}_l(k)$, as well as the number of units that can be active or inactive for different inputs, $\mathcal{I}_l(k)$ (Serra and Ramalingam, 2020).

Hinz and van de Geer (2019) observed that the upper bound by Serra et al. (2018) can be tightened by explicitly computing a recursive histogram of linear regions on the layers of the neural network according to the dimension of their image subspace. However, the resulting bound is not explicitly defined in terms of the network hyperparameters, and hence cannot be included on the table. This work is further extended in Hinz (2021) by also allowing a composition of bounds on subnetworks instead of only on the sequence of layers. Another extension of the framework from Hinz and van de Geer (2019) by Xie et al. (2020c) highlights that residual connections prevent the bottleneck effect in ResNets, by which reason such networks tend to have more linear regions.

Cai et al. (2023) proposed a separate recursive bound based on Serra et al. (2018) to account for the sparsity of the weight matrices, which illustrates how pruning connections may affect the maximum number of linear regions.

The results above have also been extended to other architectures. In some cases, results on other types of activations are also part of the papers previously mentioned: Montúfar et al. (2014) and Serra et al. (2018) present upper bounds for *maxout* networks; Raghu et al. (2017) present an upper bound for networks using *hard tanh* activation. In other cases, the ideas discussed above have been adapted for sparser networks with parameter sharing: Xiong et al. (2020) present upper and lower bounds for convolutional networks, which are shown to asymptotically define more linear regions per parameter than rectifier networks with the same input size and number of layers. Chen et al. (2022a) present upper and lower bounds for graph convolutional networks. Matoba et al. (2022) discuss the expresiveness of the maxpooling layers typically used in convolutional neural networks through their equivalence to a sequence of rectifier layers. Moreover, Goujon et al. (2022) present results for recently proposed activation functions, such as DeepSpline (Agostinelli et al., 2015, Unser, 2019, Bohra et al., 2020) and GroupSort (Anil et al., 2019).

Some of the results above were also revisited through the lenses of tropical algebra, in which every linear region corresponds to a tropical hypersurface (Zhang et al., 2018b, Charisopoulos and Maragos, 2018, Maragos et al., 2021). Notably, Montúfar et al. (2022) presented considerably tighter upper bounds for the number of linear regions in maxout networks with rank $k = 3$ or greater.

Recently, a converse line of work started exploring the minimum dimensions of a neural network capable of representing a given piecewise linear function, starting with considerations about the minimum depth necessary (Arora et al., 2018) and further refinements of bounds on the network dimensions (He et al., 2020, Hertrich et al., 2021, Chen et al., 2022b), with Chen et al. (2022b) proposing an algorithm that can construct such a neural network. On a related note, Karg and Lucia (2020) show that linear time-invariant systems in model predictive control can be exactly expressed by rectifier networks and provide bounds on the width and number of layers necessary for a given system, whereas Ferlez and Shoukry (2020) describe an algorithm for producing architectures that can be parameterized as an optimal model predictive control strategy.

Table 3: Upper bounds on the maximum number of linear regions defined by a neural network.

| Reference | Bound and conditions |
|---|---|
| Pascanu et al. (2014) | $\sum_{i=0}^{n_0} \binom{n_1}{n_0}$ for shallow networks, $n_1 \geq n_0$ |
| Montúfar et al. (2014) | $2^{\sum_{l=1}^{L} n_l}$ |
| Raghu et al. (2017) | $\prod_{l=1}^{L} \sum_{j=0}^{n_{l-1}} \binom{n_l}{j}$ |
| Montúfar (2017) | $\prod_{l=1}^{L} \sum_{j=0}^{d_l} \binom{n_l}{j}$, $d_l = \min\{n_0, n_1, \ldots, n_{l-1}\}$ |
| Serra et al. (2018) | $\sum_{(j_1,\ldots,j_L) \in J} \prod_{l=1}^{L} \binom{n_l}{j_l}, J = \{(j_1, \ldots, j_L) \in \mathbb{Z}^L : 0 \leq j_l \leq d_l \ \forall l \in \mathbb{L}\},$<br>$d_l = \min\{n_0, n_1 - j_1, \ldots, n_{l-1} - j_{l-1}, n_l\}$ |
| Serra and Ramalingam (2020) | $\sum_{(j_1,\ldots,j_L) \in J} \prod_{l=1}^{L} \binom{\mathcal{I}_l(k_{l-1})}{j_l}, J = \{(j_1, \ldots, j_L) \in \mathbb{Z}^L : 0 \leq j_l \leq d_l,$<br>$d_l = \min\{n_0, k_1, \ldots, k_{l-1}, \mathcal{I}_l(k_{l-1})\}, k_0 = n_0, k_l = \mathcal{A}_l(k_{l-1}) - j_{l-1} \ \forall l \in \mathbb{L}\}$ |

**Expected number** The third analytical approach has been the evaluation of the expected number of linear regions. In a pair of papers, Hanin and Rolnick studied the number of linear regions based on how the network parameters are typically initialized. In the first paper (Hanin and Rolnick, 2019a), they show that the average number of linear regions along 1-dimensional subspaces of the input grows linearly with respect to the number of neurons, irrespective of the network depth. In the second paper (Hanin and Rolnick, 2019b), they show that the average number of linear regions in higher-dimensional subspaces of the input also grows similarly in deep and shallow networks. For $N = \sum_{i=1}^{L} n_i$ as the total number of linear regions, the expected number of linear regions is $O(2^N)$ if $N \leq n_0$ and $O\left(\frac{(TN)^{n_0}}{n_0!}\right)$ otherwise, where $T > 0$ is a constant based on the network parameters. Moreover, some of their experiments suggest that the number of linear regions in shallow networks is slightly greater. According to the authors, these bounds reflect the fact that the family of functions that can be represented by neural networks in the way that they are typically initialized is considerably smaller. They further argue that training as currently performed is unlikely to expand the family of functions much further, as illustrated by their experiments. Similar results on the expected number of linear regions for maxout networks are presented by Tseran and Montúfar (2021), and an application of the results above results to data manifolds is explored by Tiwari and Konidaris (2022). Additional results for specific architectures of rectifier networks are conjectured by Wang (2022), although without proof.

### 3.4.2 Counting linear regions

Counting the actual number of linear regions of a given network has been a more challenging topic to explore. Serra et al. (2018) have shown that the linear regions of a trained network can be enumerated as the solutions of an MILP formulation, which has been slightly corrected in Cai et al. (2023)[1]. However, MILP solutions are generally counted one by one (Danna et al., 2007), with exception of special cases (Serra and Hooker, 2020) and small subproblems (Serra, 2020), which makes this approach impractical for large neural networks. Serra and Ramalingam (2020) have shown that approximate model counting methods, which are commonly used to count the number of feasible assignments in propositional satisfiability, can be easily adapted to solution counting in MILP, which leads to an order-of-magnitude speedup in comparison with exact counting. This type of approach is particularly suitable for obtaining probabilistic lower bounds, which can complement the analytical upper bounds for the maximum number of linear regions. In Craighero et al. (2020a) and Craighero et al. (2020b), a directed acyclic graph is used to model the sets of active neurons on each layer and how they connected with those in subsequent layers. Yang et al. (2020) describe a method for decomposing the input space of rectifier networks into their linear regions by representing each linear region in terms of its face lattice, upon which the splitting operations corresponding to the transforma-

[1] The MILP formulation of neural networks is discussed in Section 4.

tions performed by each layer can be implemented. As the number of linear regions grow, these splitting operations can be processed in parallel. Yang et al. (2021) extend that method to convolutional neural networks. Moreover, Wang (2022) describes an algorithm for enumerating linear regions that counts adjacent linear regions with same corresponding affine function as a single linear region.

Another approach is to enumerate the linear regions in subspaces, which limits their number and reduces the complexity of the task. This idea was first explored by Novak et al. (2018) for measuring the complexity of a neural network in terms of the number of transitions along a single line. Hanin and Rolnick (2019a,b) also use this method with a bounded line segment or rectangle as a single set representing the input and then sequentially partitioning it. If this first set is intersected by the activation hyperplane of a neuron in the first layer, then we replace this set by two sets corresponding to the parts of the input space in which that neuron is active and not. Once those sets are further subdivided by all activation hyperplanes associated with the neurons in the first layer, the process can be continued with the neurons in the following layers. This method is used to count the number of linear regions along subspaces of the input with dimension 1 in Hanin and Rolnick (2019a) and dimension 2 in Hanin and Rolnick (2019b). A generalized version for counting the number of linear regions in affine subspaces spanned by a set of samples using an MILP formulation is presented in Cai et al. (2023). An approximate approach for counting the number of linear regions along a line by computing the closest activation hyperplane in each layer is presented in Gamba et al. (2022).

Other approaches have obtained lower bounds on the number of linear regions of a trained network by limiting the enumeration or considering exclusively the inputs from the dataset. In Xiong et al. (2020), the number of linear regions is estimated by sampling points from the input space and enumerating all activation patterns identified through this process. In Cohan et al. (2022), the counting is restricted to the linear regions found between consecutive states of a neural network modeling a reinforcement learning policy.

## 3.5 Applications and insights

Thinking about neural networks in terms of linear regions led to a variety of applications. In turn, that inspired further studies on the structure and properties of linear regions under different settings. We organize the literature about applications and insights around some central themes in the subsections below.

### 3.5.1 The number of linear regions

From our discussion, the number of linear regions emerges as a potential proxy for the complexity of neural networks, which has been studied by some authors and exploited empirically by others. Novak et al. (2018) observed that the number of transitions between linear regions in 1-dimensional subspaces correlates with generalization. Hu et al. (2020a) used bounds on the number of linear regions as proxy to model the capacity of a neural network used for learning through distillation, in which a smaller network is trained based on the outputs of another network. Chen et al. (2021a) and Chen et al. (2021b) present one of the first approaches to training-free neural architectural search through the analysis of network properties. One of the two metrics that they have shown to be effective for that purpose is the number of linear regions associated with a sample of inputs from the training set on randomly initialized networks. Biau et al. (2021) observed that obtaining a discriminator network for Wasserstein GANs (Arjovsky et al., 2017) that correctly approximates the Wasserstein distance entails that such a discriminator network has a growing number of linear regions as the complexity of the data distribution increases. Park et al. (2021b) maximized the number of linear regions in unsupervised learning in order to produce more expressive encodings for downstream tasks using simpler classifiers. In neural networks modeling reinforcement learning policies, Cohan et al. (2022) observed that the number of transitions between linear regions in inputs corresponding to consecutive states increases by 50% with training while the number of repeated linear regions decreases. Cai et al. (2023) proposed a method for pruning different proportions of parameters from each layer by maximizes the bound on the number of linear regions, which lead to better accuracy than uniform pruning across layers. On a related note, Liang and Xu (2021) proposed a new variant of the ReLU activation function for dividing the input space into a greater number of linear regions.

The number of linear regions also inspired further theoretical work. Amrami and Goldberg (2021) presented an argument for the benefit of depth in neural networks based on the number of linear regions for correctly separating samples associated with different classes. Liu and Liang (2021) studied upper and lower bounds on the optimal approximation error of a convex univariate function based on the number of linear regions of a rectifier network. Henriksen et al. (2022) used the maximum number of linear regions as a metric for capacity that may limit repairing incorrect classifications in a neural network.

### 3.5.2 The shape of linear regions

Some studies aimed at understanding what affects the shape of linear regions in practice, including how to train neural networks in such a way to induce certain shapes in the linear regions. Zhang and Wu (2020) observed that multiple training techniques may lead to similar accuracy, but very different shape for the linear regions. For example, batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) lead to more linear regions. While batch normalization breaks the space in regions with uniform size, more orthogonal norms, and more gradient variability across adjacent regions; dropout produces more linear regions around decision boundaries, norms are more parallel, and data points less likely to be in the region containing the decision boundary. Croce et al. (2019) and Lee et al. (2019) applied regularization to the loss function to push the boundary of each linear region away from points in the training set that it contains, as long as those points are correctly classified. They show that this form of regularization improves the robustness of the neural network while making the linear regions larger. In fact, Zhu et al. (2020) observed that the boundaries of the linear regions move away from the training data; and He et al. (2021) conjectured that the linear regions near training samples becomes smaller through training, or that conversely the activation patterns are denser around the training samples. Gamba et al. (2020) presented an empirical study on the angles between activation hyperplanes defined by convolutional layers, and observed that their cosines tend to be similar and more negative with depth after training.

The geometry of linear regions also led to other theoretical and algorithmic advances. Theoretically, Phuong and Lampert (2020) proved that architectures with nonincreasing layer widths have unique parameters —upon permutation and scaling— for representing certain functions. In other words, some pairs of neural networks are only equivalent if their parameters only differ by permutation and multiplication. Grigsby et al. (2023) showed that equivalences other than by permutation are less likely to occur with greater input size and width, but more likely with greater depth. Algorithmically, Rolnick and Kording (2020) proposed a procedure to reconstruct a neural network by evaluating several inputs in order to determine regions of the input space for which the output of the neural network can be defined by an affine function —and thus consist of a single linear region. Depending on how the shape changes between adjacent linear regions, the boundaries of the linear regions are replicated with neurons in the first hidden layer or in subsequent layers of the reconstructed neural network. Masden (2022) provided theoretical results and an algorithm for characterizing the face lattice of the polyhedron associated with each linear region.

### 3.5.3 Activation patterns and the discrimination of inputs

Another common theme is understanding how inputs from the training and test sets are distributed among the linear regions, and what can be inferred the encoding of the activation patterns associated with the linear regions. Gopinath et al. (2019) noted that many properties of neural networks, including the classes of different inputs, are associated with activation patterns —and thus with their linear regions. Several works (He et al., 2021, Sattelberg et al., 2020, Trimmel et al., 2021) observed that each training sample is typically located in a different linear region when the neural network is sufficiently expressive; whereas He et al. (2021) noted that simple machine learning algorithms can be applied using the activation patterns as features, and Sattelberg et al. (2020) noted that there is some similarity between activation patterns of different neural networks under affine mapping, meaning that the training of these neural networks lead to similar models. Chaudhry et al. (2020) exploited the idea of continual learning with different tasks being encoded in disjoint subspaces, which thus corresponds to a disjoint set of activation sets on each layer being associated with classifications for each of those tasks. Based on their approach for enumerating linear regions, Craighero et al. (2020a) and Craighero et al. (2020b) have found that inputs from larger linear regions are often correctly classified by the neural network, that inputs from smaller linear regions are often incorrectly classified, and that the number of distinct activations sets reduces along the layers of the neural network. Gamba et al. (2022) also discussed the issue of some linear regions being smaller and thus less likely to occur in practice. Moreover, they propose a measurement for the similarity of the affine functions associated with linear regions along a line and observed that the linear regions tend to be less similar to one another when the network is trained with incorrectly classified labels.

### 3.5.4 Function approximation

Because of the linear behavior of the output within each linear region, we can approximate the output of the neural network based on the output of its linear regions. Chu et al. (2018) and Sudjianto et al. (2020) produced linear models based on this local behavior; whereas Glass et al. (2021) observed that we can interpret neural networks as equivalent to local linear model trees (Nelles et al., 2000), in which a distinct linear model is used at each leaf of a decision tree, and provided a method to produce such models from neural networks. Trimmel et al. (2021) described how to extract the linear regions associated with the inputs from the training set as means to approximate the output of the inputs from the test set. Robinson et al. (2019) presented another approach for explicitly representing the function modeled by a neural network through the enumeration of its linear regions. On a related note, Chaudhry et al. (2020) used the assumption of training samples remaining within the same linear region during gradient descent to simplify the analysis of backpropagation.

This topic also relates to the broad literature on neural networks as universal function approximators, to which the concept of linear regions helps articulating ideal conditions. As observed by Mhaskar and Poggio (2020), the optimal number of linear regions in a neural network —or, correspondingly, of pieces of the piecewise linear function modeled by it— depends on the function being approximated. In addition, linear regions were also used explicitly to build function approximations. Kumar et al. (2019) have shown that rectifier networks can we approximated to arbitrary precision with two hidden layers, the largest of which having a neuron corresponding to each different activation pattern of the original network; an exact counterpart of this result was later presented by Villani and Schoots (2023). Fan et al. (2020) described the transformation between sufficiently wide and deep networks while arguing that the fundamental measure of complexity should be counting simplices within linear regions. In subsequent work, Fan et al. (2023) empirically observed that linear regions tend to have a small number of higher dimensional faces, or facets.

More recent studies aimed at understanding the expressiveness and approximability of neural networks in terms of their number of parameters, in particular when the number of linear regions is greater than the number of parameters (Malach and Shalev-Shwartz, 2019, Dym et al., 2020, Daubechies et al., 2022). They all discuss how the composition the modeled functions tend to present the self-similarity property of fractal distributions, which is one reason why they have so many linear regions. Keup and Helias (2022) interpreted the connection between linear regions in different parts of the input space in terms of how paper origamis are constructed: by "folding" the data for separability.

Another related topic is computing the Lipschitz constant $\rho$ of the function $f(x)$ modeled by the neural network, the smallest $\rho$ for which $\|f(x') - f(x)\| \leq \rho\|x' - x\|$ for any two inputs $x$ and $x'$. Note that the first derivative of the output of a linear region is constant, which is leveraged by Hwang and Heinecke (2020) to evaluate the stability of the network by computing the constant across linear regions by changing the activation pattern. Interestingly, Zhou and Schoellig (2019) showed that the constant grows similarly to the number of linear regions: polynomial in width and exponential in depth. A smaller constant limits the susceptibility of the network to adversarial examples (Huster et al., 2018), which are discussed in Section 4, and also lead to smaller bias variance (Loukas et al., 2021). While calculating the exact Lipschitz constant is NP-hard and encourages approximations (Virmaux and Scaman, 2018, Patrick L. Combettes, 2019), the exact constant can be computed using MILP (Jordan and Dimakis, 2020). Notably, many studies have focused on relaxations such as linear programming (Zou et al., 2019), semidefinite programming (Fazlyab et al., 2019, Chen et al., 2020), and polynomial optimization (Latorre et al., 2020). An alternative approach is to use more sophisticated activation functions for limiting the value of the constant (Anil et al., 2019, Aziznejad et al., 2020).

### 3.5.5 Optimizing over linear regions

As an alternative to optimizing over neural networks as described next in Section 4, a number of approaches have resorted to techniques that are equivalent to systematically enumerating or traversing linear regions and optimizing over them (Croce and Hein, 2018, Croce et al., 2020, Khedr et al., 2020, Vincent and Schwager, 2021, Xu et al., 2022). Notably, Vincent and Schwager (2021) and Xu et al. (2022) are mindful of the facet-defining inequalities associated with a linear region, which are the ones to change when moving toward an adjacent linear region. On a related note, Seck et al. (2021) alternates between gradient steps and solving a linear programming model within the current linear region.

# 4 Optimizing Over a Trained Neural Network

In Section 5 we will see how polyhedral-based methods can be used to *train* a neural network. In this section, we will focus on how polyhedral-based methods can be used to do something with a neural network *after it has been trained.* Specifically, after the network architecture and all parameters have been fixed, a neural network $f$ is merely a function. If each activation function $\sigma$ used to describe the network is piecewise linear (as is the case with those presented in Table 1), $f$ is also a piecewise linear function. Therefore, any optimization problem containing $f$ in some way will be a piecewise linear optimization problem. For example, in the simple case where the output of $f$ is univariate, the optimization problem

$$\min_{x \in \mathcal{X}} f(x)$$

is a piecewise linear optimization problem. As discussed in Section 3, this problem can have an enormous number of "pieces" (linear regions) when $f$ is a neural network; solving this problem thus heavily depends on the size and structure of the neural network $f$. For example, the training procedure by which $f$ is obtained can greatly influence the performance of optimization strategies (Tjeng et al., 2019, Xiao et al., 2019).

In this section, we first explore situations in which you might want to optimize over a trained neural network in this manner. We will then survey available methods for solving this method (either exactly or on the dual side) using polyhedral-based methods. We conclude with a brief view of future directions.

## 4.1 Applications of optimization over trained networks

Applications where you might want to optimize over a trained neural network $f$ broadly fall into two categories: those where $f$ is the "true" object of interest, and those where $f$ is a convenient proxy for some unknown, underlying behavior.

### 4.1.1 Neural network verification

Neural network verification is a burgeoning field of study in deep learning. Starting in the early 2000s, researchers began to recognize the importance of rigorously verifying the behavior of neural networks, mainly in aviation-related applications (Schumann et al., 2003, Zakrzewski, 2001). More recently, the seminal works of Szegedy et al. (2014) and Goodfellow et al. (2015) observed that neural networks are unusually susceptible to *adversarial attacks*. These are small, targeted perturbations that can drastically affect the output of the network; as shown in Figure 8, even powerful models such as MobileNetV2 (Sandler et al., 2018) are susceptible. The existence and prevalence of adversarial attacks in deep neural networks has raised justifiable concerns about the deployment of these models in mission-critical systems such as autonomous vehicles (Deng et al., 2020), aviation (Kouvaros et al., 2021), or medical systems (Finlayson et al., 2019). One fascinating empirical work by Eykholt et al. (2018) showed the susceptibility of standard image classification networks that might be used in self-driving vehicles to a very analogue form of attacks: black/white stickers, placed in a careful way, could confuse these models enough that they would mis-classify road signs (e.g., mistaking stop signs for "speed limit 80" signs).

Neural network verification seeks to prove (or disprove) a given input-output relationship, i.e., $x \in \mathcal{X} \Rightarrow y \in \mathcal{Y}$, that gives some indication of model robustness. Methods for verifying this relationship are classified as being sound and/or complete. A method that is *sound* will only certify the relationship if it is indeed true (no false positives), while a method that is *complete* will (i) always return an answer and (ii) only disprove the relationship if it is false (no false

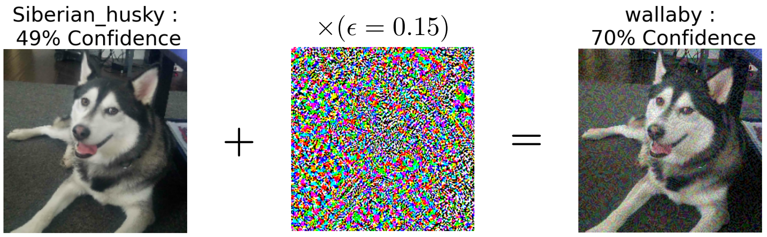

Figure 8: Example of adversarial attack on MobileNetV2 (Sandler et al., 2018). The original image taken by one of the survey authors is classified as 'siberian_husky,' but is re-classified as 'wallaby' with a small (in an $\ell_\infty$-norm sense) targeted attack.

negatives). An early set of papers (Fischetti and Jo, 2018, Lomuscio and Maganti, 2017, Tjeng et al., 2019) recognized that MILP provides an avenue for verification that is both sound and complete, given that $\mathcal{X}$ and $f(x)$ are both linear, or piecewise linear. We refer the readers to recent reviews (Huang et al., 2020, Leofante et al., 2018, Li et al., 2022, Liu et al., 2021) for a more comprehensive treatment of the landscape of verification methods, including MILP- and LP-based technologies.

**Example 4** *Consider a classification network $f : [0,1]^{n_0} \to \mathbb{R}^d$ where the $j$-th output, $f(x)_j$, corresponds to the probability that input $x$ is of class $j$.*[2] *Then consider a labeled image $\hat{x}$ known to be of class $i$, and a "target" adversarial class $k \neq i$. Then verifying local robustness of the prediction corresponds to checking $x \in \{x : ||x - \hat{x}|| \leq \epsilon\} \Rightarrow y = f(x) \in \{y : y_i \geq y_k\}$, where $\epsilon > 0$ is a constant which prescribes the radius around which $\hat{x}$ we will search for an adversarial example.*

*This verification task can be formulated as an optimization problem of the form:*

$$\begin{aligned} \max_{x \in [0,1]^{n_0}} \quad & f(x)_k - f(x)_i \\ s.t.||x - \hat{x}|| \leq \epsilon. & \end{aligned} \tag{3}$$

*Any feasible solution $x$ to this problem with positive cost is an adversarial example: it is very "close" to $\hat{x}$ which has true label $i$, yet the network believes it is more likely to be of class $k$.*[3] *If, on the other hand, it is proven that the optimal objective value is negative, this proves that $f$ is robust (at least in the*

[2] In actuality, we will instead typically work with the outputs corresponding to "logits", or unnormalized probabilities. These are typically fed into a softmax layer that then normalize these values to correspond to a probability distribution over the classes. However, this nonlinear softmax transformation is not piecewise linear. Thankfully, it can be omitted in the context of the verification task without loss of generality.

[3] Alternative objectives are sometimes used which would allow us to strengthen this statement to say that the network *will* classify $x$ to be of class $k$. However, this will require a more complex reformulation to model this problem via MILP, so we omit it for simplicity.

*neighborhood around $\tilde{x}$). Note that the verification problem can terminate once the sign of the optimal objective value is determined, but solving the problem returns an optimal adversarial example.*

The objective function of (3) models the desired input-output relationship, $x \in \mathcal{X} \Rightarrow y \in \mathcal{Y}$, while the constraints model the domain $\mathcal{X}$. The domain $\mathcal{X}$ is typically a box or hyperrectangular domain. Extensions to this are described in Section 4.4.1. Some explanation-focused verification applications define the input-output relationship in a derivative sense, e.g., $x \in \mathcal{X} \Rightarrow \partial y / \partial x \in \mathcal{Y}'$ (Wicker et al., 2022). As the derivative of the ReLU function is also piecewise linear, this class of problems can also be modeled in MILP. For example, in the context of fairness and explainability, Liu et al. (2020) and Jordan and Dimakis (2020) used MILP to certify monotonicity and to compute local Lipschitz constants, respectively.

Although in this survey we focus on optimization over trained neural networks, it is important to note that polyhedral theory underlies numerous strategies for neural network verification. For example, SAT and SMT (Satisfiability Modulo Theories) solvers designed for Boolean satisfiability problems (and more general problems for the case of SMT) can also be used to search through activation patterns for a neural network (Pulina and Tacchella, 2010), resulting in tools that are sound and complete, such as Planet (Ehlers, 2017) and Reluplex (Katz et al., 2017). Bunel et al. (2018) presented a unified view to compare MILP and SMT formulations, as well as the relaxations that result from these formulations (we will revisit this in Section 4.3). On the other hand, strategies such as ExactReach (Xiang et al., 2017) exploit polyhedral theory to compute reachable sets: given an input set to a ReLU function defined as a union of polytopes, the output reachable set is also a union of polytopes. Other methods over-approximate the reachable set to improve scalability, e.g., for vision models (Yang et al., 2021), often resulting in methods that are sound, but not complete.

### 4.1.2 Neural network as proxy

Another situation in which you may want to solve an optimization problem containing trained neural networks is when you would like to optimize some other, unknown function for which you have historical input/output data. A similar situation arises when you want to solve an optimization problem where (some of) the constraints are overly complicated, but you can query samples from the constraints on which to train a simpler *surrogate* model. In these cases, you might imagine training a neural network in a standard supervised learning setting to approximate this underlying, unknown or complicated function. Then, since the neural network is known, you are left with a deterministic piecewise linear optimization problem. Note that we focus here on using a neural network as a surrogate; neural networks can additionally learn other components of an optimization problem, e.g., uncertainty sets for robust optimization (Goerigk and Kurtz, 2023).

Several software tools have been developed for this class of problems. For the case of constraint learning, JANOS (Bergman et al., 2022) and OptiCL (Maragno et al., 2021) both provide functionality for learning a ReLU neural network to approximate a constraint based on data and embedding the learned neural network in MILP. The reluMIP package (Lueg et al., 2021) has also been introduced to handle the latter embedding step. More generally, OMLT (Ceccon et al., 2022) translates neural networks to pyomo optimization blocks, including various MILP formulations and activation functions. Finally, recent developments in gurobipy[4] enable directly parsing in trained neural networks.

Applications of this paradigm can be envisioned in a number of domain areas. This approach is common in deep reinforcement learning, where neural networks are used to approximate an unknown "$Q$-function" which models the long-term cost of taking a particular action in a particular state of the world. In $Q$-learning, this $Q$-function is optimized iteratively to produce new candidate policies, which are then evaluated (typically via simulation) to produce new training data for future iterations. Optimization over the learned $Q$-function must be relatively fast in control applications, and several practical methods have been proposed. When the action space is discrete, the $Q$-function neural network is trained with one output value for each possible action, simplifying optimization to evaluating the model and selecting the largest output. Continuous action spaces require the $Q$ network be optimized over (Burtea and Tsay, 2023, Delarue et al., 2020, Ryu et al., 2020), or an "actor" network can be trained to learn the optimal actions (Lillicrap et al., 2015). In a related vein, ReLU neural networks can be used as a process model for optimal scheduling or control (Wu et al., 2020).

Chemical engineering also presents applications where surrogate models have proven beneficial for optimization, as is the subject of recent reviews (Bhosekar and Ierapetritou, 2018, McBride and Sundmacher, 2019, Tsay and Baldea, 2019). In particular, ReLU neural networks can be seamlessly embedded in larger MILP problems such as flow networks and reservoir control where the other constraints are also mixed-integer linear (Grimstad and Andersson, 2019, Say et al., 2017, Yang et al., 2022). Focusing on control applications where the neural network is embedded in a MILP that must be solved repeatedly, Katz et al. (2020) showed how multiparametric programming can be used to learn the solution map of the resulting MILP itself, which is also piecewise affine. An emerging area of research uses verification tools to reason about neural networks used as controllers, e.g., see Johnson et al. (2020). These applications involve optimization formulations combining the neural network with constraints defining the controlled system. For example, verification can be used to bound the reachable set (Sidrane et al., 2022) (alongside piecewise linear bounds on the dynamical system) or the maximum error against a baseline controller (Schwan et al., 2022).

Finally, applications for optimization over neural networks arise in machine learning applications. MILP formulations can be used to compress neural networks (Serra et al., 2020, 2021, ElAraby et al., 2020), which consequently result in more tractable surrogate models (Kody et al., 2022). The main idea is to

[4] https://github.com/Gurobi/gurobi-machinelearning

use MILP to identify *stable* nodes, i.e., nodes that are always on or off over an input domain, which can then be algebraically eliminated. Optimization has also been employed in techniques for feature selection, based on identifying strongest input nodes (Sildir and Aydin, 2022, Zhao et al., 2023). In the context of Bayesian optimization, Volpp et al. (2020) use reinforcement learning to meta-learn acquisition functions parameterized as neural networks; selecting ensuing query points then requires optimization over the trained acquisition function. Later work modeled both the acquisition function and feasible region in black-box optimization as neural networks (Papalexopoulos et al., 2022). In that work, exploration and exploitation are balanced via Thompson sampling and training multiple neural networks from a random parameter initialization.

**A word of caution** Standard supervised learning algorithms aim to learn a function which fits the underlying function according to some distribution under which the data is generated. However, optimizing a function corresponds to evaluating it at a single point. This means that you may end up with a model that well-approximates the underlying function in distribution, but for which the pointwise minimizer is a poor approximation of the true function. This phenomena is referred to as the "Optimizer's curse" (Smith and Winkler, 2006).

### 4.1.3 Single neuron relaxations

For the following subsections, consider the $i$-th neuron in the $l$-th layer of a neural network, endowed with a ReLU activation function, whose behavior is governed by (2). Presume that a input domain of interest $\mathcal{D}^{l-1} \subset \mathbb{R}^{n_l}$ is a bounded region. Further, since $\mathcal{D}^{l-1}$ is bounded, presume that finite bounds are known on each input component, i.e. that vectors $L^{l-1}, U^{l-1} \in \mathbb{R}^{n_l}$ are known such that $\mathcal{D}^{l-1} \subseteq [L^{l-1}, U^{l-1}] \subset \mathbb{R}^{n_l}$. We can then write the *graph* of the neuron, which couples together the input and the output of the nonlinear ReLU activation function:

$$\begin{aligned} \mathtt{gr} = & \left\{ \, (\boldsymbol{h}^{l-1}, h_i^l) \in \mathcal{D}^{l-1} \times \mathbb{R} \;\middle|\; h_i^l = 0 \geq \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l \, \right\} \\ & \cup \left\{ \, (\boldsymbol{h}^{l-1}, h_i^l) \in \mathcal{D}^{l-1} \times \mathbb{R} \;\middle|\; h_i^l = \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l \geq 0 \, \right\}. \end{aligned}$$

This is a disjunctive representation for $\mathtt{gr}$ in terms of two polyhedral alternatives. We assume that every included neuron exhibits this disjunction, i.e., every neuron can be on or off depending on the model input. This assumption of *strict activity* implies that $L^{l-1} < 0$ and $U^{l-1} > 0$, noting that neurons not satisfying this property can be exactly pruned from the model (Serra et al., 2020).

We observe that, given this (or any) formulation for each individual unit, it is straightforward to construct a formulation for the entire network. For example, if we take $X_i^l = \left\{ \, (\boldsymbol{h}^{l-1}, h_i^l, z_i^l) \;\middle|\; (4) \, \right\}$ for each layer $l$ and each unit $i$, we can construct a MILP formulation for the graph of the entire network, $\left\{ \, (x, f(x)) : x \in \mathcal{D}^0 \, \right\}$ as

$$(\boldsymbol{h}^{l-1}, h_i^l, z_i^l) \in X_i^l \quad \forall l \in \mathbb{L}, i \in [\![ n_l ]\!].$$

This also generalizes in a straightforward manner to more complex feedforward network architectures (e.g. convolutions, or sparse or skip connections), though we omit the explicit description for notational simplicity.

### 4.1.4 Beyond the scope of this survey

The effectiveness of the single-neuron formulations described above is bounded by the tightness of the optimal univariate formulation; this property is known as the "single-neuron barrier" (Salman et al., 2019). This has motivated research in convex relaxations that jointly account for multiple neurons within a layer (Singh et al., 2019a). Nevertheless, the analysis of polyhedral formulations for multiple neurons simultaneously quickly becomes intractable, and is beyond the scope of this survey. Instead, we point the interested reader to the recent survey by Roth (2021), and highlight a few approaches taken in the literature. Multi-neuron analysis has been used to: improve bounds tightening schemes (Rössig and Petkovic, 2021), prune linearizable neurons (Botoeva et al., 2020), design dual decompositions (Ferrari et al., 2022), and generate strengthening inequalities (Serra and Ramalingam, 2020). Similarly, we do not review formulations for ensembles of ReLU networks, though MILP formulations have been proposed (Wang et al., 2021, 2023).

Additionally, recent works have exploited polyhedral structure to develop sampling based strategies, which can be used to warm-start MILP or accelerate local search in verification (Perakis and Tsiourvas, 2022, Wu et al., 2022). Lombardi et al. (2017) computationally compare MILP against local search and constraint programming approaches. In a related vein, Cheon (2022) examines local solutions and proposes an outer approximation method to improve gradient-based optimization. Finally, following Raghunathan et al. (2018), a large body of work has presented optimization-based methods for verification that use semidefinite programming concepts (Dathathri et al., 2020, Fazlyab et al., 2020, Newton and Papachristodoulou, 2021). Notably, Batten et al. (2021) showed how combining semidefinite and MILP formulations can produce a new formulation that is tighter than both. This was later extended with reformulation-linearization technique, or RLT, cuts (Lan et al., 2022). While related to linear programming and other methods based on convex relaxations, this stream of work is beyond the scope of this survey. We refer the reader to Zhang (2020) for a discussion of the tightness of these formulations.

## 4.2 Exact models using mixed-integer programming

Mixed-integer programming offers a powerful algorithmic framework for *exactly* modeling nonconvex piecewise linear functions. The Operations Research community has studied has a long and storied history of developing MILP-based methods for piecewise linear optimization, with research spanning decades (Croxton et al., 2003, Dantzig, 1960, Geißler et al., 2012, Huchette and Vielma, 2022, Lee and Wilson, 2001, Misener and Floudas, 2012, Padberg, 2000, Vielma et al., 2010). However, many of these techniques are specialized for low-dimensional or separable piecewise linear functions. While a reasonable assumption in many OR problems, this is not the case when modeling neurons in a neural network. Therefore, the standard approach in the literature is to apply general-purpose MILP formulation techniques to model neural networks.

**Connection to Boolean satisfiability** Some SMT-based methods such as Reluplex (Katz et al., 2017) and Planet (Ehlers, 2017) effectively construct branching technologies similar to MILP solvers. Indeed, `Marabou` (Katz et al., 2019) builds on Reluplex, and a recent extension `MarabouOpt` can optimize over trained neural networks (Strong et al., 2021). The authors also outline general procedures to extend verification solvers to optimization. Our focus in this review is on more general MILP formulations, or those that can be incorporated into off-the-shelf MILP solvers with relative ease. Bunel et al. (2020b, 2018) provide a more comprehensive discussion of similarities and differences to SMT.

### 4.2.1 The big-$M$ formulation

The big-$M$ method is a standard technique used to formulate logic and disjunctive constraints using mixed-integer programming (Bonami et al., 2015, Vielma, 2015). Big-$M$ formulations are typically very simple to reason about and implement, and are quite compact, though their convex relaxations can often be quite poor, leading to weak dual bounds and (often) slow convergence when passed to a mixed-integer programming solver. Since `gr` is a disjunctive set, the big-$M$ technique can be applied to produce the following formulation:

$$\begin{aligned} h_i^l &\geq \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l && \text{(4a)} \\ h_i^l &\leq \left(\boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l\right) - M_{i,-}^l (1-z) && \text{(4b)} \\ h_i^l &\leq M_{i,+}^l z && \text{(4c)} \\ (\boldsymbol{h}^{l-1}, h_i^l) &\in [L^{l-1}, U^{l-1}] \times \mathbb{R}_{\geq 0} && \text{(4d)} \\ z_i^l &\in \{0,1\}. && \text{(4e)} \end{aligned}$$

Here, $M_{i,-}^l$ and $M_{i,+}^l$ are data which must satisfy the inequalities

$$\begin{aligned} M_{i,-}^l &\leq \min_{\boldsymbol{h}^{l-1} \in \mathcal{D}^{l-1}} \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l \\ M_{i,+}^l &\geq \max_{\boldsymbol{h}^{l-1} \in \mathcal{D}^{l-1}} \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l. \end{aligned}$$

This big-$M$ formulation for ReLU-based networks has been used extensively in the literature (Bunel et al., 2018, Cheng et al., 2017, Dutta et al., 2018, Fischetti and Jo, 2018, Kumar et al., 2019, Lomuscio and Maganti, 2017, Serra and Ramalingam, 2020, Serra et al., 2018, Tjeng et al., 2019, Xiao et al., 2019).

The big-$M$ formulation is compact, with one binary variable and $\mathcal{O}(1)$ general inequality constraints for each neuron. Applied for each unit in the network, this leads to a MILP formulation with $\mathcal{O}(\sum_{l \in \mathbb{L}} n_l) = \mathcal{O}(L n_{\max})$ binary variables and general inequality constraints, where $n_{\max} = \max_{l \in \mathbb{L}} n_L$. However, it has

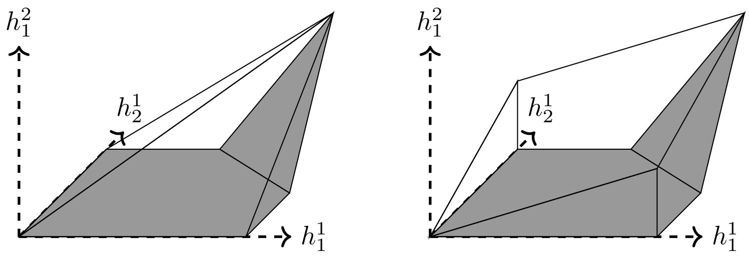

Figure 9: **Left:** The convex hull of a ReLU neuron (5), and **Right:** the convex relaxation offered by the big-$M$ formulation (4) Adapted from Anderson et al. Anderson et al. (2020, 2019)

been observed (Anderson et al., 2019, 2020) that this big-$M$ formulation is not strong in the sense that its LP relaxation does not, in general, capture the convex hull of the graph of a given unit; see Figure 9 for an illustration. In fact, this LP relaxation can be arbitrarily bad (Anderson et al., 2019, Example 2), even in fixed input dimension. As MILP solvers often bound the objective function between the best feasible point and its tightest optimal continuous relaxation, a weak formulation can negatively impact performance, often substantially.

It is worth dwelling on where this lack of strength comes from. If the input $\boldsymbol{h}^{l-1}$ is one dimensional, the big-$M$ formulation is *locally* ideal (Vielma, 2015): the extreme points of the linear programming relaxation (4a-4d) naturally satisfy the integrality constraints (4e). However, this fails to hold in the general multivariate input case. To see why, observe that the bounds on the input variables $\boldsymbol{h}^{l-1}$ are only coupled with the logic involving the binary variable $z$ only in an aggregated sense, through the coefficients $M^l_{i,-}$ and $M^l_{i,+}$. In other words, the "shape" of the pre-activation domain is not incorporated directly into the big-$M$ formulation. Furthermore, the strength of this formulation highly depends on the big-$M$ coefficients. These coefficients can be obtained using techniques ranging from basic interval arithmetic to optimization-based bounds tightening. Grimstad and Andersson (2019) show how constraints external to the neural network can yield tighter bounds via optimization- or feasibility-based bounds tightening. Rössig and Petkovic (2021) compare several methods for deriving bounds and further develop optimization-based bounds tightening based on pairwise dependencies between variables.

### 4.2.2 A stronger extended formulation

A much stronger MILP formulation can be constructed through a classical method, the extended formulation for disjunctions (Balas, 1998, Jeroslow and Lowe, 1984). This formulation for a given ReLU neuron takes the following form (Anderson et al., 2019, Section 2.2):

$$
\begin{aligned}
(\boldsymbol{h}^{l-1}, h_i^l) &= (x^+, y^+) + (x^-, y^-) && \text{(5a)}\\
y^- = 0 &\geq \boldsymbol{w}_i^l x^- + b_i^l(1-z) && \text{(5b)}\\
y^+ &= \boldsymbol{w}_i^l x^+ + b_i^l z \geq 0 && \text{(5c)}\\
L^{l-1}(1-z) &\leq x^- \leq U^{l-1}(1-z) && \text{(5d)}\\
L^{l-1}z &\leq x^+ \leq U^{l-1}z && \text{(5e)}\\
z &\in \{0,1\}. && \text{(5f)}
\end{aligned}
$$

This formulation requires one binary variable and $\mathcal{O}(n_{l-1})$ general linear constraints and auxiliary continuous variables. It is also locally ideal, i.e., as strong as possible. While the number of variables and constraints for an individual unit seems quite tame, applying this formulation for unit in a network leads to a formulation with $\mathcal{O}(n_0 + \sum_{l\in\mathbb{L}} n_l n_{l-1}) = \mathcal{O}(|\mathbb{L}|n_{\max}^2)$ continuous variables and linear constraints. Moreover, while the formulation for *an individual unit* is locally ideal, the composition of many locally ideal formulations will, in general, fail to be ideal itself. Consider that, while each node can be modeled as a two-part disjunction, the full network requires exponentially many disjuncts, each corresponding to one activation pattern.

Despite its strength and relatively modest increase in size relative to the big-$M$ formulation (4), it has been empirically observed that this formulation often performs worse than expected (Anderson et al., 2019, Vielma, 2019), both in the verification setting and more broadly.

### 4.2.3 A class of intermediate formulations

The previous sections observed that the big-$M$ formulation (4) is compact, but may offer a weak convex relaxation, while the extended formulation (5) offers the tightest possible convex relaxation for an individual unit, at the expense of a much larger formulation. Kronqvist et al. (2022, 2021) present a strategy for obtaining formulations intermediate to (4) and (5) in terms of both size and strength. The key idea is to partition $\boldsymbol{w}_i^l \boldsymbol{h}^{l-1}$ into a number of aggregated variables, $\boldsymbol{w}_i^l \boldsymbol{h}^{l-1} = \sum_{p=1}^{P} \hat{x}_p$. Each auxiliary variable $\hat{x}_p$ is defined as a sum of a subset of the $j$-th weighted inputs $\hat{x}_p = \sum_{j\in\mathbb{S}_p} w_{i,j}^l h_j^{l-1}$, with $\mathbb{S}_1, ..., \mathbb{S}_P$ partitioning $\{1, ..., n_{l-1}\}$. This technique can be applied to the ReLU function, giving the convex hull over the directions defined by $\hat{x}_p$ (Tsay et al., 2021):

$$\left(\sum_{j\in\mathbb{S}_p} w^l_{i,j} h^{l-1}_j, h^l_i\right) = (\hat{x}^+_p, y^+) + (\hat{x}^-_p, y^-) \tag{6a}$$

$$y^- = 0 \geq \sum_p \hat{x}^-_p + b^l_i(1-z) \tag{6b}$$

$$y^+ = \sum_p \hat{x}^+_p + b^l_i z \geq 0 \tag{6c}$$

$$\hat{\boldsymbol{M}}^l_{i,-}(1-z) \leq \hat{x}^- \leq \hat{\boldsymbol{M}}^l_{i,+}(1-z) \tag{6d}$$

$$\hat{\boldsymbol{M}}^l_{i,-} z \leq \hat{x}^+ \leq \hat{\boldsymbol{M}}^l_{i,+} z \tag{6e}$$

$$z \in \{0,1\}. \tag{6f}$$

Here, the $p$-th elements of $\hat{\boldsymbol{M}}^l_{i,-}$ and $\hat{\boldsymbol{M}}^l_{i,+}$ must satisfy the inequalities

$$\hat{M}^l_{i,-,p} \leq \min_{\boldsymbol{h}^{l-1}\in\mathcal{D}^{l-1}} \sum_{j\in\mathbb{S}_p} w^l_{i,j} h^{l-1}_j$$

$$\hat{M}^l_{i,+,p} \geq \max_{\boldsymbol{h}^{l-1}\in\mathcal{D}^{l-1}} \sum_{j\in\mathbb{S}_p} w^l_{i,j} h^{l-1}_j.$$

These coefficients can be derived using techniques analagous to those for the big-$M$ formulation (note that tighter bounds may be derived by considering $\hat{x}^-$ and $\hat{x}^+$ separately). Observe that when $P = 1$, we recover the same tightness as the big-$M$ formulation (4), as, intuitively, the formulation is built over a single "direction" corresponding to $\boldsymbol{w}^l_i \boldsymbol{h}^{l-1}$. Conversely, when $P = n_{l-1}$, we recover the tightness of the extended formulation (5), as each direction corresponds to a single element of $\boldsymbol{h}^{l-1}$. Tsay et al. (2021) study partitioning strategies and show that intermediate values of $P$ result in formulations that can outperform the two extremes, by balancing formulation size and strength.

### 4.2.4 Cutting plane methods: Trading variables for inequalities

The extended formulation (5) achieves its strength through the introduction of auxiliary continuous variables. However, it is possible to produce a formulation of equal strength by projecting out these auxiliary variables, leaving an ideal formulation in the "original" $(\boldsymbol{h}^{l-1}, h^l_i, z)$ variable space. While in general this projection may be difficult computationally, for the simple structure of a single ReLU neuron it is possible to characterize in closed form. The formulation is given by Anderson et al. (2020, 2019) as

$$h_i^l \geq \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l \tag{7a}$$

$$h_i^l \leq \sum_{j \in J} w_{i,j}^l (h_i^{l-1} - \breve{L}_j^l (1-z)) + \left( b + \sum_{j \notin J} w_{i,j}^l \breve{U}_j \right) z \quad \forall J \subseteq \llbracket n_{l-1} \rrbracket \tag{7b}$$

$$(\boldsymbol{h}^{l-1}, h_i^l) \in \mathcal{D}^{l-1} \times \mathbb{R}_{\geq 0} \tag{7c}$$

$$z_i^l \in \{0, 1\}, \tag{7d}$$

where notationally, for each $j \in \llbracket n_{l-1} \rrbracket$, we take

$$\breve{L}_j^{l-1} = \begin{cases} L_j^{l-1} & w_{i,j}^l \geq 0 \\ U_j^{l-1} & w_{i,j}^l < 0 \end{cases} \quad \text{and} \quad \breve{U}_j^{l-1} = \begin{cases} U_j^{l-1} & w_{i,j}^l \geq 0 \\ L_j^{l-1} & w_{i,j}^l < 0 \end{cases}$$

We note a few points of interest about this formulation. First, it is ideal, and so recovers the convex hull of a ReLU activation function, coupled with its preactivation affine function and bounds on each of the inputs to that affine function. Second, it can be shown that, under very mild conditions, each of the exponentially many constraints in (7b) are necessary to ensure this property; none are redundant and can be removed without affecting the relaxation quality. Third, note that by selecting only those constraints in (7b) corresponding to $J = \emptyset$ and $J = \llbracket n_{l-1} \rrbracket$, we recover the big-$M$ formulation (4) in the case where $\mathcal{D}^{l-1} = [L^{l-1}, U^{l-1}]$. This suggests a practical approach for using this large family of inequalities: Start with the big-$M$ formulation, and then dynamically generate violated inequalities from (7b) as-needed in a cutting plane procedure. As shown by Anderson et al. (2020), this separation problem is separable in the input variables, and hence can be completed in $\mathcal{O}(n_{l-1})$ time.

The cutting plane strategy is in general compatible with weaker formulations, such as relaxation-based verification (Zhang et al., 2022) and formulations from the class (6). In fact, Tsay et al. (2021) show that the intermediate formulations in (6) effectively pre-select a number of inequalities from (7b), in terms of their continuous relaxations. While adding these constraints results in a tighter continuous relaxation, the added constraints can eventually significantly increase the model size. Practical implementations may therefore only perform cut generation at a limited number of branch-and-bound search nodes (De Palma et al., 2021, Tsay et al., 2021).

**A subtlety when using** (7) This third point above raises a subtlety discussed in the literature (De Palma et al., 2021, Appendix F). Often, additional structural information is known about $\mathcal{D}^{l-1}$ beyond bounds on the variables. In this case, it is typically possible to derive tighter values for the big-$M$ coefficients. In this case, when using a separation-based approach it is preferable to initialize the formulation with these tightened big-$M$ constraints, and then proceed with the cutting plane approach as normal from there.

## 4.3 Scaling further: Convex relaxations and linear programming

The above demonstrate MILP as a powerful framework for exactly modeling complex, nonconvex trained neural networks, but standard solvers are often not sufficiently scalable to adequately handle large-scale networks. A natural approach to increase the scalability, then, is to *relax* the network in some manner, and then apply convex optimization methods. For the verification problem discussed in Section 4.1.1, this yields what is known as an *incomplete verifier*: any certification of robustness provided can be trusted (no false positives), but there may be robust instances that the method cannot prove are (some false negatives). In other words, over-approximation produces a verifier that is sound, but not complete.

While a variety of methods exist for accomplishing this, in this section we briefly outline techniques relevant to polyhedral theory. In particular, we focus on some techniques for building convex polyhedral relaxations. The most natural convex relaxation for a MILP formulation is its linear programming (LP) relaxation, constructed by dropping any integrality constraints. For example, the LP relaxation of (4) is given by the system (4a-4d). This is a compact linear programming relaxation for a ReLU-based network, and is the basis for methods due to Bunel et al. (2020a) and Ehlers (2017).

### 4.3.1 Projecting the big-$M$ and ideal MILP formulations

This section examines projections of the linear relaxations of formulations (4) and (7).

**(Projecting the big-$M$).** Note that the LP relaxation given by (4a–4d) maintains the variables $z_i^l$ in the formulation, though they are no longer required to satisfy integrality. Since these variables are "auxiliary" and are no longer necessary to encode the nonconvexity of the problem, they can be projected out without altering the quality of the convex relaxation. Doing this yields what is commonly known as the "triangle" or "$\Delta$" relaxation (Salman et al., 2019):

$$h_i^l \geq \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l \tag{8a}$$

$$h_i^l \leq \frac{M_{i,+}^l}{M_{i,+}^l - M_{i,-}^l} (\boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l) \tag{8b}$$

$$(\boldsymbol{h}^{l-1}, h_i^l) \in [L^{l-1}, U^{l-1}] \times \mathbb{R}_{\geq 0}. \tag{8c}$$

While the LP relaxation (8) for an individual unit is compact, modern neural network architectures regularly comprise millions of units. The resulting LP relaxation for the entire network may then require millions of variables and constraints. Additionally, unless special precautions are taken, many of these constraints will be relatively dense. All this quickly leads to LP that are beyond the scope of modern off-the-shelf LP solvers. As a result, researchers have explored alternative schemes for scaling LP-based methods to these larger networks. Salman et al. (2019) present a framework for LP-based methods (LP solvers, propagation, dual methods), which we review in the following subsections. However, they do not account for the ideal formulation developed in later works (Anderson et al., 2020, De Palma et al., 2021).

**(Projecting the ideal).** Figure 9 shows that the triangle (big-$M$) relaxation fails to recover the convex hull of the ReLU activation function and the multivariate input to the affine pre-activation function. We can similarly project the LP relaxation of the ideal formulation (7) into the space of input/output variables (Anderson et al., 2020), yielding a description for the convex hull of $\{(\boldsymbol{h}^{l-1}, h_i^l) | L^{l-1} \leq \boldsymbol{h}^{l-1} \leq U^{l-1},\ h_i^l = \sigma(\boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l)\}$:

$$h_i^l \geq \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l \tag{9a}$$

$$h_i^l \leq \sum_{k \in I} w_{i,k}^l (x_k - \breve{L}_k) + \frac{\ell(I)}{\breve{U}_h - \breve{L}_h}(x_h - \breve{L}_h) \quad \forall (I, h) \in \mathcal{J} \tag{9b}$$

$$(\boldsymbol{h}^{l-1}, h_i^l) \in [L^{l-1}, U^{l-1}] \times \mathbb{R}_{\geq 0}, \tag{9c}$$

where $l(I) \coloneqq \sum_{k \in I} w_{i,k}^l \breve{L}_k + \sum_{k \notin I} w_{i,k}^l \breve{U}_k + b_i^l$ and

$$\mathcal{J} \coloneqq \left\{ (I, h) \in 2^{[\![n_{l-1}]\!]} \times [\![n_{l-1}]\!] \;\middle|\; l(I) \geq 0,\ l(I \cup \{h\} < 0,\ w_{i,k}^l \neq 0 \forall k \in I \right\}.$$

Anderson et al. (2020) also show that the inequalities (9b) can be separated over in $\mathcal{O}(n_{l-1})$ time. Interestingly, in contrast to (7), the number of facet-defining inequalities depends heavily on the affine function. While in the worst case the number of inequalities will grow exponentially in the input dimension, there exist instances where the convex hull can be fully described with only $\mathcal{O}(n_{l-1})$ inequalities.

### 4.3.2 Dual decomposition methods

A first approach for greater scalability for LP-based methods is decomposition, a standard technique in the large-scale optimization community. Indeed, the cutting plane approach of Section 4.2.4 can be viewed as a decomposition method operating in the original variable space. However, the method is initialized with the big-$M$ formulation for each neuron, and hence this initial model will be of size roughly equal to the size of the network. Therefore, it should be understood to use decomposition to provide a tighter verification bound, rather than for providing greater scalability to larger networks.

In contrast, dual decomposition can be used to scale inexact verification methods to larger networks. Such methods maintain dual feasible solutions throughout the algorithm, meaning that upon termination they yield valid dual bounds on the verification instance, and hence serve as incomplete verifiers.

Wong and Kolter (2018), Wong et al. (2018) use as their starting point the triangle relaxation (8) for each neuron, and then take the standard LP dual of the (relaxed) verification problem. Alternatively, Dvijotham et al. (2018b) propose a Lagrangian-based approach for decomposing the original nonlinear formulation of the problem (3). Crucially, since the complicating constraints coupling the layers in the network are imposed as objective penalties instead of “hard” constraints, the optimization problem (given fixed dual variables) decomposes along each layer and the subproblems induced by the separability can be solved in closed form. This approach dualizes separately the equations characterizing the pre-activation and post-activation functions:

$$\begin{aligned}
\max_{\mu,\lambda} \quad \min_{\boldsymbol{h},\hat{\boldsymbol{h}}} \quad & \left(\boldsymbol{W}^L \boldsymbol{h}^{L-1} + \boldsymbol{b}^L\right) + \sum_{k=1}^{L-1} \Big( \mu_k^T (\hat{\boldsymbol{h}}^k - \boldsymbol{W}^k \boldsymbol{h}^{k-1} - \boldsymbol{b}^k) + \lambda_k^T (\boldsymbol{h}^k - \sigma(\hat{\boldsymbol{h}}^k) \Big) \\
\text{s.t.} \quad & L^k \le \hat{\boldsymbol{h}}^k \le U^k \quad \forall k \in [\![n-1]\!] \\
& \sigma(L^k) \le \boldsymbol{h}^k \le \sigma(U^k) \quad \forall k \in [\![n-1]\!].
\end{aligned}$$

Here, the $\hat{\boldsymbol{h}}$ variables track the pre-activation values for the neurons in the network. The dual variables $\mu_k^T$ correspond to the equality constraints defining the pre-activation values, $\hat{\boldsymbol{h}}^k = \boldsymbol{W}^k \boldsymbol{h}^{k-1} + \boldsymbol{b}^k$. Likewise, the dual variables $\lambda_k^T$ correspond to enforcing the ReLU activation function, $\boldsymbol{h}^k = \sigma(\hat{\boldsymbol{h}}^k) = \max(0, \hat{\boldsymbol{h}}^k)$. Any feasible solution for the neural network is feasible for this dualized problem, making the multiplier terms for $\mu_k^T$ and $\lambda_k^T$ zero. Thus, the inner problem gives a lower bound for the original problem—a property known as *weak duality.* The outer (dual) problem optimizing over the Lagrangian multipliers then seeks to maximize this lower bound, i.e., to give the tightest possible lower bound. This can be solved using a subgradient-based method, or learned along with the model parameters in a “predictor-verifier” approach (Dvijotham et al., 2018a).

On the other hand, this approach can be combined with other relaxation-based methods. The Lagrangian decomposition can be applied to dualize only the coupling constraints between layers, and a convex relaxation used for the activation function (Bunel et al., 2020a):

$$\begin{aligned}
\max_{\lambda} \quad \min_{\boldsymbol{h},\hat{\boldsymbol{h}}} \quad & \left(\boldsymbol{W}^L \boldsymbol{h}^{L-1} + \boldsymbol{b}^L\right) + \sum_{k=1}^{L-1} \Big( \lambda_k^T (\boldsymbol{h}^k - \sigma(\hat{\boldsymbol{h}}^k) \Big) \\
\text{s.t.} \quad & L^k \le \hat{\boldsymbol{h}}^k \le U^k \quad \forall k \in [\![n-1]\!] \\
& \hat{\boldsymbol{h}}^k = \boldsymbol{W}^k \boldsymbol{h}^{k-1} + b^k \quad \forall k \in [\![n-1]\!] \\
& \boldsymbol{h}^k \ge 0 \quad \forall k \in [\![n-1]\!] \\
& \boldsymbol{h}_i^k \ge \hat{\boldsymbol{h}}_i^k \quad \forall k \in [\![n-1]\!], \forall i \in [\![n_k]\!] \\
& \boldsymbol{h}_i^k \le \frac{U_i^k (\hat{\boldsymbol{h}}_i^k - L_i^k)}{U_i^k - L_i^k} \quad \forall k \in [\![n-1]\!], \forall i \in [\![n_k]\!].
\end{aligned}$$

Note that the final three constraints apply the big-$M$/triangle relaxation (8) to each ReLU activation function. The dual problem can then be solved via subgradient-based methods, proximal algorithms, or, more recently, a projected

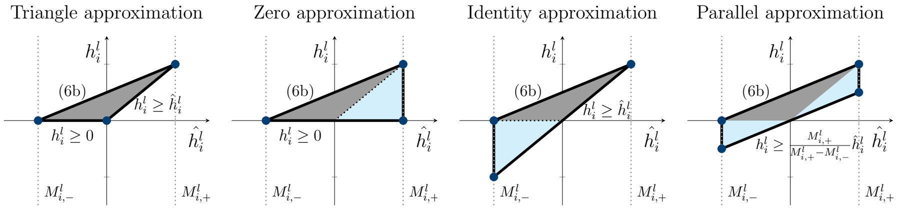

Figure 10: Convex approximations for the ReLU function commonly used by propagation algorithms, given as a function of the preactivation function $\hat{h}_i^l$. The ReLU applies $h_i^l = \max(0, \hat{h}_i^l)$.

gradient descent method applied to a nonconvex reformulation of the problem (Bunel et al., 2020c).

More recently, De Palma et al. (2021) presented a dual decomposition approach based on (7). However, creating a dual formulation from the exponential number of constraints produces an exponential number of dual variables. The authors therefore propose to maintain an "active set" of dual variables to keep the problem sparse. A selection algorithm (e.g., selecting entries that maximize an estimated super-gradient) can then be used to append the active set. Similar to the above discussion on cut generation, the frequency of appending the active set should be chosen strategically.

### 4.3.3 Fourier-Motzkin elimination and propagation algorithms

Alternatively, one can project out *all* of the decision variables. For example, in order to solve the linear programming problem $\min_{x \in \mathcal{X}} c \cdot x$, we can augment the problem with a new decision variable to $\min_{(x,y) \in \Gamma} y$ for $\Gamma := \{ (x, y) \in \mathcal{X} \times \mathbb{R} : y = c \cdot x \}$, and project out the $x$ variables. The transformed problem is the a trivial univariate optimization problem: $\min_{y \in \text{Proj}_y(\Gamma)} y$.

Of course, the complexity of the approach described is hidden in the projection step, or building $\text{Proj}_y(\Gamma)$. The most well-known algorithm for computing projections of linear inequality systems is Fourier-Motzkin elimination, described by Dantzig and Eaves (1973), which is notorious for its practical inefficiency. The process effectively comprises replacing variables from a set of inequalities with all possible implied inequalities, which can produce many unnecessary constraints. However, it turns out that neural network verification problems are well-structured in such a way that Fourier-Motzkin elimination can be performed very efficiently: for instance, by imposing one inequality upper bounding and one inequality lower bounding each ReLU function. Note that while Section 3.2 describes the use of Fourier-Motzkin elimination to obtain *exact* input-output relationships in linear regions of neural networks, here we are interested in obtaining linear *bounds* for a nonlinear function.

In fact, this general approach was independently developed in the verification community. While MILP research has focused on formulations tighter than the big-M, such as (5) and (6), the verification community often prefers greater scalability at the price of weaker convex relaxations. The continuous relaxation of the big-M is equivalent to the triangle relaxation (8): the optimal convex relaxation for a single input, or in terms of the aggregated pre-activation function, as shown in Figure 10. However, the lower bound involves two linear constraints, which is not used in several propagation-based verification tools owing to scalability or compatibility.

Such tools use methods such as abstract transformers to propagate polyhedral bounds, i.e., *zonotopes*, through the layers of a neural network. DeepZ (Singh et al., 2018), Fast-Lin (Weng et al., 2018), and Neurify (Wang et al., 2018a) employ a parallel approximation, with the latter also implementing a branch-and-bound procedure towards completeness. Subsequently, DeepPoly (Singh et al., 2019b) and CROWN (Zhang et al., 2018a) select between the zero and identity approximations by minimizing over-approximation area. OSIP (Hashemi et al., 2021) selects between the three approximations using optimization: approximations for a layer are select jointly to minimise bounds for the following layer. These technologies are also compatible with interval bounds, propagating box domains (Mirman et al., 2018). Interestingly, bounds on neural network weights can also be propagated using similar methods, allowing reachability analysis of Bayesian neural networks (Wicker et al., 2020).

Tjandraatmadja et al. (2020) provide an interpretation of these propagation techniques through the lens of Fourier-Motzkin elimination. Consider the problem of propagating bounds through a ReLU neural network: for a node $h_i^l = \max\{0, \hat{h}_i^l\}$, convex bounds for $h_i^l$ can be obtained given bounds for $\hat{h}_i^l$ (Figure 10). Assuming the inputs are outputs of ReLU activations in the previous layer, $\hat{h}_i^l = \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l$. Computing an upper bound can then be expressed as:

$$\begin{aligned} \max_{\boldsymbol{h}^{l-1}} \quad & \boldsymbol{w}_i^l \boldsymbol{h}^{l-1} + b_i^l \\ \text{s.t.} \quad & \mathcal{L}_k(\boldsymbol{h}^{l-2}) \leq h_k^{l-1} \leq \mathcal{G}_k(\boldsymbol{h}^{l-2}) \forall k \in \{1, ..., n_{l-1}\}. \end{aligned}$$

As the objective function is linear, the solution of this problem can be computed by propagation without explicit optimization. For each element in $\boldsymbol{h}^{l-1}$, we only need to consider the associated objective coefficient in $\boldsymbol{w}_i^l$ to determine whether $\mathcal{L}_k(\boldsymbol{h}^{l-2}) \leq h_k^{l-1}$ or $h_k^{l-1} \leq \mathcal{G}_k(\boldsymbol{h}^{l-2})$ will be the active inequality at the optimal solution. We can thus replace $h_k^{l-1}$ with $\mathcal{L}_k(\boldsymbol{h}^{l-2})$ or $\mathcal{G}_k(\boldsymbol{h}^{l-2})$ accordingly. This projection is mathematically equivalent to applying Fourier-Motzkin elimination, while avoiding redundant inequalities resulting from the 'non-selected' bounding function. Repeating this procedure for each layer results in a convex relaxation for the outputs that only involves the input variables. We naturally observe the desirability of simple lower bounds $\mathcal{L}_k(\boldsymbol{h}^{l-2})$: imposing two-part lower bounds in each layer would increase the number of propagated constraints in an exponential manner, similar to Fourier-Motzkin elimination.

**A path towards completeness** Given an input-output bound, the reachable set can be refined by splitting the input space (Henriksen and Lomuscio, 2021, Rubies-Royo et al., 2019)—a strategy similar to spatial branch and bound. In other words, completeness is achieved by branching in the input space, rather than activation patterns: this strategy is especially effective when the input space is low dimensional (Strong et al., 2022). For example, ReluVal (Wang et al., 2018b) propagates symbolic intervals and implements splitting procedures on the input domain. As the interval extension of ReLU is Lipschitz continuous, the method converges to arbitrary accuracy in a finite number of splits.

## 4.4 Generalizing the single neuron model

### 4.4.1 Extending to other domains

In general, we will expect that the effective input domain $\mathcal{D}^{l-1}$ for a given unit may be quite complex. For the first layer ($l = 1$) this may derive from explicitly stated constraints on the inputs of the networks, while for later layers this will typically derive from the complex nonlinear transformations applied by the preceding layers. For example, in the context of surrogate models Yang et al. (2022) propose bounding the input to the convex hull of the training data set, while other works (Schweidtmann et al., 2022, Shi et al., 2022) propose machine learning-inspired techniques for learning the trust region implied by the training data. In effect, these methods assume a trained model is locally accurate around training data, which is a property similar to that which verification seeks to prove.

Nevertheless, most research focuses on hyperrectangular input domains, largely motivated by practical considerations: i) there are efficient, well-studied methods for computing valid (though not necessarily optimally tight) variable bounds, ii) characterizing the exact effective domain may be computationally impractical, and iii) and the hyperrectangular structure makes analysis simpler for complex formulations like those presented in Section 4.2.4. We note that Jordan et al. (2019) use polyhedral analyses to perform verification over arbitrary (including non-polyhedral) norms, by fitting a $p$-norm ball in the decision region and checking adjacent linear regions. On the other hand, robust optimization can be employed to find $p$-norm adversarial regions (rather than verifying robustness), as opposed to a single point adversary (Maragno et al., 2023).

Anderson et al. (2020) present two closely related frameworks for constructing ideal and hereditarily sharp formulations for ReLU units with arbitrary polyhedral input domains. This characterization is derived from Lagrangian duality, and requires an infinite number of constraints (intuitively, one for each choice of dual multipliers). Nonetheless, separation can still be done over this infinite family of inequalities via a subgradient-based algorithm; this approach will be tractable if optimization over $\mathcal{D}^{l-1}$ is tractable. Many propagation algorithms are also fully compatible with arbitrary polyhedral input domains, as the projected problem (i.e., a linear input-output relaxation) remains an LP. Singh et al. (2021) show that simplex input domains can actually be beneficial, creating tighter formulations by propagating constraints on the inputs through the network layers. Similarly, optimization-based bound tightening problems based on solving LPs can embed constraints defining polyhedral input domains.

In certain cases, additional structural information about the input domain can be used to reduce this semi-infinite description to a finite one. For example, this can be done when $\mathcal{D}^{l-1}$ is a Cartesian product of unit simplices (Anderson et al., 2020) (note that this generalizes the box domain case, wherein each simplex is one-dimensional). This particular structure is particularly useful for modeling input domains with combinatorial constraints. For example, a network trained to predict binding propensity of a given length-$n$ DNA sequence is naturally modeled via an input domain that is the product of $n$ 4-dimensional simplices–one simplex for each letter in the sequence, each of which is selected from an alphabet of length 4.

### 4.4.2 Extending to other activation functions

The big-$M$ formulation technique can be any piecewise linear activation function. While much of the literature focuses on the ReLU due to its widespread popularity, models for other activation functions have been explored in the literature. For example, multiple papers (Serra et al., 2018, Appendix K) (Tjeng et al., 2019, Appendix A.2) present a big-$M$ formulation for the maxout activation function. Adapting a formulation from Anderson et al. (2020) (Anderson et al., 2020, Proposition 10), a formulation for the maxout unit is

$$\begin{aligned} y_i^l &\leq u_j(\boldsymbol{h}^{l-1}) + M_{i,j}^l(1 - z_j) \quad \forall j \in [\![k]\!] \\ y_i^l &\geq u_j(\boldsymbol{h}^{l-1}) \quad \forall j \in [\![k]\!] \\ \sum_{j=1}^{k} z_j &= 1 \\ (\boldsymbol{h}^{l-1}, v_i^l, z) &\in \mathcal{D}^{l-1} \times \mathbb{R} \times \{0,1\}^k, \end{aligned}$$

where each $M_{i,j}^l$ is selected such that

$$M_{i,j}^l \geq \max_{\tilde{\boldsymbol{h}} \in \mathcal{D}^{l-1}} u_j(\tilde{\boldsymbol{h}}).$$

We can observe that the big-$M$ formulation can also handle other discontinuous activation functions, such as a binary/sign activations (Han and Gómez, 2021) or more general quantized activations (Nguyen and Huchette, 2022). Nevertheless, the binary activation function naturally lends itself towards Boolean satisfiability, and most work therefore focuses on alternative methods such as SAT (Cheng et al., 2018, Jia and Rinard, 2020, Narodytska et al., 2018).

While this survey focuses on neural networks with piecewise linear activation functions, we note that recent research has also studied smooth activation functions with a similar aim. For example, optimization over smooth activation functions can be handled by piecewise linear approximation and conversion to

MILP (Sildir and Aydin, 2022). Researchers have also studied convex/concave bounds for nonlinear activation functions, which can then be embedded in spatial branch-and-bound procedures (Schweidtmann and Mitsos, 2019, Wilhelm et al., 2022). In contrast to MILP formulations for ReLU neural networks, these problems are typically nonlinear programs that must be solved via spatial branch and bound.

Propagation methods (Singh et al., 2018, Zhang et al., 2018a) can also naturally handle general activation functions: given convex polytopic bounds for an activation function, these tools can propagate them through network layers using the same techniques. For example, Fastened CROWN (Lyu et al., 2020) employs a set of search heuristics to quickly select linear upper and lower bounds on ReLU, sigmoid, and hyperbolic tangent activation functions. Tighter polyhedral bounds can be employed, such as piecewise linear upper and lower bounds (Benussi et al., 2022).

### 4.4.3 Extending to adversarial training

As described in Section 1, the *training* of neural networks seeks to minimise a measure of distance between the output $y$ and the correct output $\hat{y}$. For instance, if this distance is prescribed as loss function $\mathcal{L}(y, \hat{y})$, this corresponds to solving the *training* optimization problem:

$$\min_{\{\boldsymbol{W}^l\}_{l \in \mathbb{L}}, \{\boldsymbol{b}^l\}_{l \in \mathbb{L}}} \mathcal{L}(y, \hat{y}). \tag{10}$$

Further details about the training problem and solution methods are described in the following section. Here, we briefly outline how verification techniques can be embedded in training. Specifically, solutions or bounds to the verification problem (Section 4.1.1) provide a metric of how robust a trained neural network is to perturbations. These metrics can be embedded in the training problem to obtain a more robust network during training, often resulting in a bilevel training problem. For instance, the verification problem (3) can be embedded as a lower-level problem, giving the robust optimization problem:

$$\min_{\{\boldsymbol{W}^l\}_{l \in \mathbb{L}}, \{\boldsymbol{b}^l\}_{l \in \mathbb{L}}} \quad \max_{||x - \hat{x}|| \leq \epsilon} \quad \mathcal{L}(y = f(x), \hat{y}).$$

Solving these problems generally involves either bilevel optimization, or computing an adversarial solution/bound at each training step, conceptually similar to the robust cutting plane approach. Madry et al. (2018) proposed this formulation and solved the nonconvex inner problem using gradient descent, thereby losing a formal certification of robustness. These approaches may also benefit from reformulation strategies, such as by taking the dual of the inner problem and using any feasible solution as a bound (Wong and Kolter, 2018). The resulting models are not only more robust, but several works have also found it to be empirically easier to verify robustness in them (Mirman et al., 2018, Wong and Kolter, 2018).

Alternatively, robustness can be induced by designing an additional penalty term for the training loss function, in a similar vein to regularization. For example:

$$\min_{\{\boldsymbol{W}^l\}_{l\in\mathbb{L}},\{\boldsymbol{b}^l\}_{l\in\mathbb{L}}} \kappa\mathcal{L}(y,\hat{y}) + (1-\kappa)\mathcal{L}_{\text{robust}}(\cdot).$$

Additionally, if these robustness penalties are differentiable, they can be embedded into standard gradient descent based optimization algorithms (Dvijotham et al., 2018b, Mirman et al., 2018). In the above formulation, the parameter $\kappa$ controls the relative weighting between fitting the training data and satisfying some robustness criterion, and its value can be scheduled during training, e.g., to first focus on model accuracy (Gowal et al., 2018). In these cases, over-approximation of the reachable set is less problematic, as it merely produces a model *more* robust than required. Nevertheless, Balunović and Vechev (2020) improve relaxation tightness by searching for adversarial examples in the "latent" space between hidden layers, reducing the number of propagation steps. Zhang et al. (2020) provide an implementation that that tightens relaxations by also propagating bounds backwards through the network.

# 5 Linear Programming and Polyhedral Theory in Training

In the previous sections, we have almost exclusively focused on tasks involving neural networks that have already been constructed, i.e., we have assumed that the training step has already concluded (with the exception of Section 4.4.3). In this section, we focus on the training phase, whose goal is to construct a neural network that can represent the relationship between the input and output of a given set of data points.

Let us consider a set of points, or sample, $(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i)_{i=1}^D$, and assume that these points are related via a function $\hat{f}$, i.e., $\hat{f}(\tilde{\boldsymbol{x}}_i) = \tilde{\boldsymbol{y}}_i \; i = 1, \ldots, D$. In the training phase, we look for $\hat{f}$ in a pre-defined class (e.g. neural networks with a specific architecture) that approximates the relation $\hat{f}(\tilde{\boldsymbol{x}}_i) = \tilde{\boldsymbol{y}}_i$. Typically, this is done by solving an Empirical Risk Minimization problem

$$\min_{\hat{f}\in F} \frac{1}{D}\sum_{i=1}^{D} \ell(\hat{f}(\tilde{\boldsymbol{x}}_i), \tilde{\boldsymbol{y}}_i) \tag{11}$$

where $\ell$ is a loss function and $F$ is the class of functions we are restricted to. We usually assume the class $F$ is parametrized by $(\boldsymbol{W}, \boldsymbol{b}) \in \Theta$ (the network weights and biases), so we are further assuming that there exists a function $f(\cdot,\cdot,\cdot)$ (the network architecture) such that

$$\forall \hat{f} \in F,\, \exists(\boldsymbol{W}, \boldsymbol{b}) \in \Theta,\, \hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}),$$

and thus, the optimization is performed over the space of parameters. In many cases, $\Theta = \mathbb{R}^N$—the parameters are unrestricted real numbers—but we will see some cases when a different parameter space can be used.

As mentioned in the introduction, nowadays, most of the practically successful *training* algorithms for neural networks, i.e., that solve or approximate (11), are based on Stochastic Gradient Descent (SGD). From a fundamental perspective, optimization problem (11) is typically a *non-convex, unconstrained* problem that needs to be solved efficiently and where finding a *local minimum* is sufficient. Thus, it is not too surprising that linear programming appears to be an unsuitable tool in this phase, in general. Nonetheless, there are some notable and surprising exceptions to this, which we review here.

Linear programming played an interesting role in training neural networks before SGD became the predominant training method and provided an efficient approach for constructing neural networks with 1 hidden layer in the 90s. This method has some common points in their polyhedral approach with the first known algorithm that can solve (11) to provable optimality for a 1-hidden-layer ReLU neural network, which was proposed in 2018. Recently, a stream of work has exploited similar polyhedral structures to obtain convex optimization reformulations of regularized training problems of ReLU networks. Linear programming tools have also been used within SGD-type methods in order to compute optimal *step-sizes* in the optimization of (11) or to strictly enforce structure in $\Theta$. From a different perspective, a *data-independent* polytope was used to describe approximately all training problems that can arise from an uncertainty set. Additionally, a back-propagation-like algorithm for training neural networks, which solves mixed-integer linear problems in each layer, was proposed as an alternative to SGD. Furthermore, when the neural network weights are required to be discrete, the applicability of SGD is impaired, and mixed-integer linear models have been proposed to tackle the corresponding training problems.

In what follows, we review these roles of (mixed-integer) linear programming and polyhedral theory within training contexts. We refer the reader to the book by Goodfellow et al. (2016) and the surveys by Curtis and Scheinberg (2017), Bottou et al. (2018), and Wright (2018) for in-depth descriptions and analyses of the most commonly used training methods for neural networks.

We remark that solving the training problem to global optimality for ReLU neural networks is computationally complex. Even in architectures with just one hidden node, the problem is NP-hard (Dey et al., 2020, Goel et al., 2021). Also see Blum and Rivest (1992), Boob et al. (2022), Chen et al. (2022c), Froese et al. (2022), Froese and Hertrich (2023) for other hardness results. Furthermore, it has been recently shown that training ReLU networks is $\exists\mathbb{R}$-complete (Abrahamsen et al., 2021, Bertschinger et al., 2022), which implies that it is likely that the problem of optimally training ReLU neural networks is not even in NP. Therefore, it is not strange to see that some of the methods we review below, even when they are solving hard problems as sub-routines (like mixed-integer linear problems), either make some non-trivial assumptions or relax some requirements. For example, boundedness and/or integrality of the weights, architecture restrictions such as the output dimension, or not having optimality guarantees.

It is worth mentioning that, in contrast, for LTUs, exact exponential-time training algorithms are known for much more general architectures than in the

ReLU case (Khalife and Basu, 2022, Ergen et al., 2023). These are out of scope for this survey, though we will provide a high-level overview of some of them, as they share some similarities to approaches designed for ReLU networks.

## 5.1 Training neural networks with a single hidden layer

Following the well-known XOR limitation of the perceptron (Minsky and Papert, 1969), a natural interest arose in the development of training algorithms that could handle at least one hidden layer. In this section, we review training algorithms that can successfully minimize the training error in a one-hidden-layer setting and rely on polyhedral approaches.

### 5.1.1 Problem setting and solution scheme

Suppose we have a sample of size $D$ $(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i)_{i=1}^D$ where $\tilde{\boldsymbol{x}}_i \in \mathbb{R}^n$ and $\tilde{\boldsymbol{y}}_i \in \mathbb{R}$. In a training phase, we would like to find a neural network function $f(\cdot,\cdot,\cdot)$ that represents in the best possible way the relation $f(\tilde{\boldsymbol{x}}_i, \boldsymbol{W}, \boldsymbol{b}) = \tilde{\boldsymbol{y}}_i$.

Note that when a neural network $\hat{f}$ has only one hidden layer, its behavior is almost completely determined by the sign of each component of the vector

$$\boldsymbol{W}^1 x - \boldsymbol{b}^1.$$

These are the cases of ReLU activations $\sigma(z) = \max\{0, z\}$ and LTU activations $\sigma(z) = \operatorname{sgn}(z)$. The training algorithms we show here heavily exploit this observation and construct $(\boldsymbol{W}^1, \boldsymbol{b}^1)$ by embedding in this phase a *hyperplane partition* problem based on the sample $(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i)_{i=1}^D$. While the focus of this survey is mainly devoted to ReLU activations, we also discuss some selected cases with LTU activations as they share some similar ideas.

### 5.1.2 LTU activations and variable number of nodes

One stream of work dedicated to developing training algorithms for one-hidden-layer networks concerned the use of *backpropagation* (Rumelhart et al., 1986, LeCun et al., 1989, Werbos, 1974). In the early 90s, an alternative family of methods was proposed, which was heavily based on linear programs (see e.g. Bennett and Mangasarian (1990, 1992), Roy et al. (1993), Mukhopadhyay et al. (1993)). These approaches can construct a 1-hidden-layer network without the need for an *a-priori* number of nodes in the network. We illustrate the high-level idea of these next, based on the survey by Mangasarian (1993).

Suppose that $\tilde{\boldsymbol{y}}_i \in \{-1, 1\}$, thus the NN we construct will be a classifier. The training phase can be tackled via the construction of a *polyhedral partition* of $\mathbb{R}^n$ such that no two points (or few) $\tilde{\boldsymbol{x}}_i$ and $\tilde{\boldsymbol{x}}_j$ such that $\tilde{\boldsymbol{y}}_i \neq \tilde{\boldsymbol{y}}_j$ lie in the same element of the partition. To achieve this, the following approach presented by Bennett and Mangasarian (1992) can be followed. Let $Y = \{i \in [D] : \tilde{\boldsymbol{y}}_i = 1\}$

and $N = [D] \setminus Y$, and consider the following optimization problem

$$\min_{\boldsymbol{w},b,y,z} \quad \frac{1}{|Y|}\sum_{i\in Y} y_i + \frac{1}{|N|}\sum_{i\in N} z_i \tag{12a}$$

$$\boldsymbol{w}^\top \tilde{\boldsymbol{x}}_i - b + y \geq 1 \qquad \forall i \in Y \tag{12b}$$

$$-\boldsymbol{w}^\top \tilde{\boldsymbol{x}}_i + b + z \geq 1 \qquad \forall i \in N \tag{12c}$$

$$y, z \geq 0. \tag{12d}$$

This LP aims at finding a hyperplane $\boldsymbol{w}^\top x = b$ separating the data according to their value of $\tilde{\boldsymbol{y}}_i$. Since the data may not be separable, the LP is minimizing the following classification error

$$\frac{1}{|Y|}\sum_{i\in Y}(-\boldsymbol{w}^\top \tilde{\boldsymbol{x}}_i + b + 1)_+ + \frac{1}{|N|}\sum_{i\in N}(\boldsymbol{w}^\top \tilde{\boldsymbol{x}}_i - b + 1)_+.$$

The LP (12) is a linear reformulation of the latter minimization problem, where the auxiliary values $y, z$ take the value of each element in the sum.

Once the LP (12) is solved, we obtain 2 halfspaces classifying our data points. In order to obtain a richer classification and lower error, we can iterate the procedure by means of the Multi-Surface Method Tree (MSMT, see Bennett (1992)), which solves a sequence of LPs as (12) in order to produce a polyhedral partition of $\mathbb{R}^n$. Let us illustrate how this procedure works in a simplified case: assume that solving (12) results in a vector $\boldsymbol{w}_1$ such that

$$\{i \,:\, \boldsymbol{w}_1^\top \tilde{\boldsymbol{x}}_i \geq b_1\} \subseteq Y \quad \wedge \quad \{i \,:\, \boldsymbol{w}_1^\top \tilde{\boldsymbol{x}}_i \leq a_1\} \subseteq N,$$

for some $a_1, b_1 \in \mathbb{R}^n$ with $b_1 > a_1$. We can remove the sets $\{(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i) \,:\, \boldsymbol{w}_1^\top \tilde{\boldsymbol{x}}_i \geq b_1\}$ and $\{(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i) \,:\, \boldsymbol{w}_1^\top \tilde{\boldsymbol{x}}_i \leq a_1\}$ from the data-set and redefine (12) accordingly, in order to obtain a new vector $\boldsymbol{w}_2$ and scalars $b_2, a_2$ that would be used to classify within the region $\{x \in \mathbb{R}^n \,:\, a_1 < \boldsymbol{w}_1^\top \boldsymbol{x} < b_1\}$.

This procedure can be iterated, and the polyhedral partition of $\mathbb{R}^n$ induced by the resulting hyperplanes can be easily transformed into a Neural Network with 1 hidden layer and LTU activations (see Bennett and Mangasarian (1990) for details). We illustrate this transformation with the following example: suppose that after 3 iterations we have the following regions, with the arrow indicating to which class each region is associated to:

$$\{\boldsymbol{x} \in \mathbb{R}^n \,:\, \boldsymbol{w}_1^\top \boldsymbol{x} \geq b_1\} \to Y, \tag{13a}$$

$$\{\boldsymbol{x} \in \mathbb{R}^n \,:\, \boldsymbol{w}_1^\top \boldsymbol{x} \leq a_1\} \to N, \tag{13b}$$

$$\{\boldsymbol{x} \in \mathbb{R}^n \,:\, a_1 < \boldsymbol{w}_1^\top \boldsymbol{x} < b_1,\, \boldsymbol{w}_2^\top \boldsymbol{x} \geq b_2\} \to Y, \tag{13c}$$

$$\{\boldsymbol{x} \in \mathbb{R}^n \,:\, a_1 < \boldsymbol{w}_1^\top \boldsymbol{x} < b_1,\, \boldsymbol{w}_2^\top \boldsymbol{x} \leq a_2\} \to N, \tag{13d}$$

$$\{\boldsymbol{x} \in \mathbb{R}^n \,:\, a_1 < \boldsymbol{w}_1^\top \boldsymbol{x} < b_1,\, a_2 < \boldsymbol{w}_2^\top \boldsymbol{x} < b_2,\, \boldsymbol{w}_3^\top x \geq (a_3 + b_3)/2\} \to Y, \tag{13e}$$

$$\{\boldsymbol{x} \in \mathbb{R}^n \,:\, a_1 < \boldsymbol{w}_1^\top \boldsymbol{x} < b_1,\, a_2 < \boldsymbol{w}_2^\top \boldsymbol{x} < b_2,\, \boldsymbol{w}_3^\top x < (a_3 + b_3)/2\} \to N. \tag{13f}$$

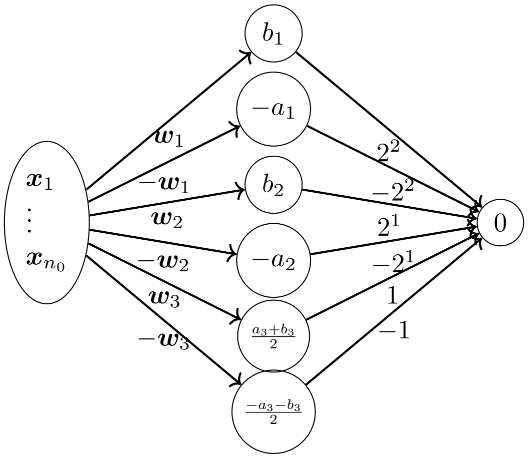

Figure 11: Illustration of Neural Network with LTU activations using MSMT. Inside each node of the hidden layer, we show the thresholds used in each LTU activation.

Since regions (13e) and (13f) are the last defined by the algorithm (under some stopping criterion), they both use $(a_3 + b_3)/2$ in order to obtain a well-defined partition of $\mathbb{R}^n$. In Figure 11 we show a one-hidden-layer neural network that represents such a classifier. The structure of the neural network can be easily extended to handle more regions. For other details, we refer the reader to Bennett (1992), and for variants and extensions see Mangasarian (1993) and references therein.

Some key features of this procedure are the following:

- Each solution of (12), i.e., each new hyperplane, can be represented as a new node in the hidden layer of the resulting neural network.
- The addition of a new hyperplane comes with a reduction in the current loss; this can be iterated until a target loss is met.
- Thanks to the universal approximation theorem (Hornik et al., 1989), with enough nodes in the hidden layer, we can always obtain a neural network $\hat{f}$ with zero classification error. Although this can lead to over-fitting.

The work of Roy et al. (1993) and Mukhopadhyay et al. (1993) follow a related idea, although the classifiers which are built are quadratic functions. To illustrate the approach, we use the same set-up for (12). The approach in Roy et al. (1993) and Mukhopadhyay et al. (1993) aims at finding a function

$$f_{\boldsymbol{V},\boldsymbol{w},b}(x) = \boldsymbol{x}^\top \boldsymbol{V} \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{x} + b$$

such that

$$f_{\boldsymbol{V},\boldsymbol{w},b}(\tilde{\boldsymbol{x}}_i) \geq 0 \Longleftrightarrow i \in Y.$$

Since this may not be possible, the authors propose solving the following LP

$$\min_{W,w,b,\epsilon} \quad \epsilon \tag{14a}$$

$$\tilde{\boldsymbol{x}}_i^\top \boldsymbol{V} \tilde{\boldsymbol{x}}_i + \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_i + b \geq \epsilon \qquad \forall i \in Y \tag{14b}$$

$$\tilde{\boldsymbol{x}}_i^\top \boldsymbol{V} \tilde{\boldsymbol{x}}_i + \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_i + b \leq -\epsilon \qquad \forall i \notin Y \tag{14c}$$

$$\epsilon \geq \epsilon_0 \tag{14d}$$

for some fixed tolerance $\epsilon_0 > 0$. When this LP is infeasible, the class $Y$ is partitioned into $Y_1$ and $Y_2$, and an LP as (14) is solved for both $Y_1$ and $Y_2$. The algorithm then follows iteratively (see below for comments on these iterations). In the end, the algorithm will construct $k$ quadratic functions $f_1, \ldots, f_k$, which the authors call "masking functions", that will classify an input $\boldsymbol{x}$ in the class $Y$ if and only if

$$\exists i \in [k],\ f_i(\boldsymbol{x}) \geq 0.$$

In order to represent the resulting classifier as a single-layer neural network, the authors proceed in a similar manner to a linear classifier; the input layer of the resulting neural network not only includes each entry of $\boldsymbol{x}$, but also the bilinear terms $\boldsymbol{x}\boldsymbol{x}^\top$. Using this input, the classifier built by (14) can be thought of as a linear classifier (much like a polynomial regression can be cast as a linear regression).

As a last comment on the work of Roy et al. (1993) and Mukhopadhyay et al. (1993), the authors' algorithm does not iterate in a straightforward fashion. They add clustering iterations alternating with the steps described above in order to (a) identify outliers and remove them from the training set, and (b) subdivide the training data when (14) is infeasible. These additions allow them to obtain a polynomial-time algorithm.

The methods described in this section are able to produce a neural network with arbitrary quality, however, there is no guarantee on the size of the resulting neural network. When the size of the network is fixed the story changes, which is the case we describe next.

### 5.1.3 Fixed number of nodes and ReLU activations

As mentioned at the beginning of this section, training a neural network is a complex optimization problem in general, with some results indicating that the problem is likely to not even be in NP (Abrahamsen et al., 2021, Bertschinger et al., 2022).

Nonetheless, by restricting the network architecture sufficiently and allowing exponential running times, exact algorithms can be conceived. An important step in the construction of such algorithms was taken by Arora et al. (2018). In this work, the authors studied the training problem in detail, providing the first optimization algorithm capable of solving the training problem to provable optimality for a fixed network architecture with one hidden layer and with an output dimension of 1. As we anticipated, this algorithm shares some similarities with the previous approach.

Let us consider now a ReLU activation. Also, we no longer assume $\tilde{\boldsymbol{y}}_i \in \{-1, 1\}$, but we keep the output dimension as 1. The problem considered by Arora et al. (2018) reads

$$\min_{\boldsymbol{W},\boldsymbol{b}} \frac{1}{D} \sum_{i=1}^{D} \ell(\boldsymbol{W}^2(\sigma(\boldsymbol{W}^1 \tilde{\boldsymbol{x}}_i + \boldsymbol{b}^1)), \tilde{\boldsymbol{y}}_i), \tag{15}$$

with $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ a convex loss. Note that this problem, even if $\ell$ is convex, is a non-convex optimization problem.

**Theorem 1 (Arora et al. (2018))** *Let $n_1$ be the number of nodes in the hidden layer. There exists an algorithm to find a global optimum of* (15) *in time $O(2^{n_1} D^{n_0 \cdot n_1} poly(D, n_0, n_1))$.*

Roughly speaking, the algorithm works by noting that one can assume the weights in $\boldsymbol{W}^2$ are in $\{-1, 1\}$, since $\sigma$ is positively-homogeneous. Thus, problem (15) can be restated as

$$\min_{\boldsymbol{W}^1,\boldsymbol{b}^1,s} \frac{1}{D} \sum_{i=1}^{D} \ell(s(\sigma(\boldsymbol{W}^1 \tilde{\boldsymbol{x}}_i + \boldsymbol{b}^1)), \tilde{\boldsymbol{y}}_i) \tag{16}$$

where $s \in \{-1, 1\}^{n_1}$. In order to handle the non-linearity, Arora et al. (2018) “guess” the values of $s$ and the sign of each component of $\boldsymbol{W}^1 \tilde{\boldsymbol{x}}_i + \boldsymbol{b}^1$. Enforcing a sign for each component of $\boldsymbol{W}^1 \tilde{\boldsymbol{x}}_i + \boldsymbol{b}^1$ is similar to the approach discussed in the previous section: it fixes how the input part of the data $(\tilde{\boldsymbol{x}}_i)_{i=1}^D$ is partitioned in polyhedral regions by a number of hyperplanes. The difference is that, in this case, the number of hyperplanes to be used is assumed to be fixed.

Using the hyperplane arrangement theorem (see e.g. (Matousek, 2002, Proposition 6.1.1)), there are at most $D^{n_0 n_1}$ ways of fixing the signs of $\boldsymbol{W}^1 \tilde{\boldsymbol{x}}_i + \boldsymbol{b}^1$. Additionally, there are at most $2^{n_1}$ possible vectors in $\{-1, 1\}^{n_1}$. Once these components are fixed, (16) can be solved as an optimization problem with a convex objective function and a polyhedral feasible region imposing the desired signs in $\boldsymbol{W}^1 \tilde{\boldsymbol{x}}_i + \boldsymbol{b}^1$. This results in the $O(2^{n_1} D^{n_0 n_1} \mathrm{poly}(D, n_0, n_1))$ running time. This algorithm was recently generalized to concave loss functions by Froese et al. (2022).

Dey et al. (2020) developed a polynomial-time approximation algorithm in this setting for the case of $n_1 = 1$ (i.e., one ReLU neuron) and square loss. This approximation algorithm has a better performance when the input dimension is much larger than the sample size, i.e. $n_0 \gg D$. The approach by Dey et al. (2020) also relies on fixing the signs of $\boldsymbol{W}^1 \tilde{\boldsymbol{x}}_i + \boldsymbol{b}^1$, and then solving multiple convex optimization problems, but in different strategy than that of Arora et al. (2018); in particular, Dey et al. (2020) only explore a polynomial number of the possible “fixings”, which yields the approximation.

We note that the result by Arora et al. (2018) shows that the training problem on their architecture is in NP. This is in contrast to Bertschinger et al.

(2022), who show that training a neural network with one hidden layer is likely to not be in NP. The big difference lies in the assumption on the output dimension: in the case of Bertschinger et al. (2022), the output dimension is two. It is quite remarkable that such a sharp complexity gap is produced by a small change in the output dimension.

### 5.1.4 An exact training algorithm for arbitrary LTU architectures

Recently, Khalife and Basu (2022) presented a new algorithm, akin to that in Arora et al. (2018), capable of solving the training problem to global optimality for any fixed LTU architecture with a convex loss function $\ell$. In this case, no assumption on the network's depth is made. The algorithm runs in polynomial time on the sample size $D$ when the architecture is fixed.

We will not describe this approach in detail, as it heavily relies on the structure given by LTU activations, which is intricate and beyond the scope of this survey. Although we note that it shares some high-level similarities to the algorithm of Arora et al. (2018) for ReLU activations, such as "guessing" the behavior of the neurons' activity and then solving multiple convex optimization problems. However, the structural and algorithmic details are considerably different.

It is important to note that this result reveals the big gap between what is known for LTU versus ReLU activations in terms of their training problems. In the case of the former, there is an exact algorithm for arbitrary architectures; in the case of the latter, the known results are much more restricted and strong computational limitations exist.

## 5.2 Convex reformulations in regularized training problems

For the case when the training problem is regularized, the following stream of work has developed several convex reformulations of it. Pilanci and Ergen (2020) presented the first convex reformulation of a training problem for the case with one hidden layer and one-dimensional outputs. As the approach described in Section 5.1.3, this reformulation uses hyperplane arrangements according to the activation patterns of the ReLU units, but instead of using them algorithmically directly, they use them to find their convex reformulations. This framework was further extended to CNNs by Ergen and Pilanci (2021c). Higher-dimensional outputs in neural networks with one hidden layer were considered in Ergen and Pilanci (2020, 2021a), Sahiner et al. (2021). This convex optimization perspective was also applied in Batch Normalization by Ergen et al. (2022).

These approaches provide polynomial-time algorithms when some parameters (e.g., the input dimension $n_0$) are considered constant. We note that this does not contradict the hardness result of Froese and Hertrich (2023), as the latter does not include a regularizing term. We explain below where the regularizing term plays an important role. Training via convex optimization was further developed to handle deeper regularized neural networks in Ergen and Pilanci (2021b,d,e).

In what follows, we review the convex reformulation in Pilanci and Ergen (2020) (one hidden layer and one-dimensional output) to illustrate some of the base strategies behind these approaches. We refer the reader to the previously mentioned articles for the most recent and intricate developments, as well as numerical experiments.

As before, let $n_1$ be the number of nodes in the hidden layer. Let us consider the following regularized training problem; to simplify the discussion, we omit biases.

$$\min_{\boldsymbol{W}} \quad \frac{1}{2}\left\|\sum_{j=1}^{n_1} \boldsymbol{W}_j^2 \sigma(\tilde{\boldsymbol{X}}\boldsymbol{W}_j^1) - \tilde{\boldsymbol{y}}\right\|^2 + \frac{\beta}{2}\sum_{j=1}^{n_1}(\|\boldsymbol{W}_j^1\|^2 + (\boldsymbol{W}_j^2)^2) \tag{17}$$

Here, $\beta > 0$, $\tilde{\boldsymbol{X}}$ is a matrix whose $i$-th row is $\tilde{\boldsymbol{x}}_i$ and $\boldsymbol{W}_j^1$ is the vector of weights going into neuron $j$. Thus, $\tilde{\boldsymbol{X}}\boldsymbol{W}_j^1$ is a vector whose $i$-th component is the input to neuron $j$ when evaluating the network on $\tilde{\boldsymbol{x}}_i$. $\boldsymbol{W}_j^2$ is a scalar: it is the weight on the arc from neuron $j$ to the output neuron (one-dimensional). Note that there is a slight notation overload: $(\boldsymbol{W}_j^2)^2$ is the square of the scalar $\boldsymbol{W}_j^2$. However, we will quickly remove this (pontentially confusing) term.

Problem (17) is a regularized version of (15) when $\ell$ is the squared difference. We modified its presentation to match the structure in Pilanci and Ergen (2020). The authors first prove that (17) is equivalent to

$$\min_{\|\boldsymbol{W}_j^1\|\leq 1} \min_{\boldsymbol{W}_j^2} \quad \frac{1}{2}\left\|\sum_{j=1}^{n_1} \boldsymbol{W}_j^2 \sigma(\tilde{\boldsymbol{X}}\boldsymbol{W}_j^1) - \tilde{\boldsymbol{y}}\right\|^2 + \beta\sum_{j=1}^{n_1}|\boldsymbol{W}_j^2|$$

Then, through a series of reformulations and duality arguments, the authors first show that this problem is lower bounded by

$$\begin{aligned}
\max \quad & -\frac{1}{2}\left\|v - \tilde{\boldsymbol{y}}\right\|^2 + \frac{1}{2}\|\tilde{\boldsymbol{y}}\|^2 && && \text{(18a)}\\
\text{s.t} \quad & |v^\top \sigma(\tilde{\boldsymbol{X}}u)| \leq \beta && \forall u,\ \|u\| \leq 1 && \text{(18b)}\\
& v \in \mathbb{R}^D && && \text{(18c)}
\end{aligned}$$

Problem (18) has $D$ variables and infinitely many constraints. The authors show that this lower bound is tight when the number of neurons in the hidden layer is large enough; specifically, they require $n_1 \geq m^*$, where $m^* \in \{1, \ldots, D\}$ is defined as the number of Dirac deltas in an optimal solution of a dual of (18) (see Pilanci and Ergen (2020) for details).

Regarding the presence of infinitely many constraints, the authors address this by considering all possible patterns of signs of $\tilde{\boldsymbol{X}}u$ (similarly to Arora et al. (2018), as discussed in Section 5.1.3). For each fixed sign pattern (hyperplane arrangement), they apply a duality argument which allows them to recast the constraint $\max_{u\in\mathcal{B}} |v^\top \sigma(\tilde{\boldsymbol{X}}u)| \leq \beta$ as a finite collection of second-order cone constraints with $\beta$ on the right-hand side.

Finally, using that $\beta > 0$, they show that the reformulated problem satisfies Slater's condition, and thus from strong duality they obtain the following convex optimization problem, which has the same objective value as (18).

$$\begin{aligned}
\min \quad & \frac{1}{2}\left\|\sum_{j=1}^{P} M_i\tilde{\boldsymbol{X}}(v_i - w_i) - \tilde{\boldsymbol{y}}\right\|^2 + \beta\sum_{j=1}^{P}(\|v_i\| + \|w_i\|) && && \text{(19a)}\\
\text{s.t} \quad & (2M_i - I_D)\tilde{\boldsymbol{X}}v_i \geq 0 && \forall i \in [P] && \text{(19b)}\\
& (2M_i - I_D)\tilde{\boldsymbol{X}}w_i \geq 0 && \forall i \in [P] && \text{(19c)}\\
& v_i \in \mathbb{R}^{n_0} && \forall i \in [P] && \text{(19d)}\\
& w_i \in \mathbb{R}^{n_0} && \forall i \in [P] && \text{(19e)}
\end{aligned}$$

Here, $I_D$ is the $D \times D$ identity matrix, $P$ is the number of possible activation patterns for $\tilde{\boldsymbol{X}}$, and each $M_i$ is a $D \times D$ binary diagonal matrix whose diagonal indicates the $i$-th possible sign pattern of $\tilde{\boldsymbol{X}}u$. This means that $(M_i)_{j,j}$ is 1 if and only if $\tilde{\boldsymbol{x}}_j^\top u \geq 0$ in the $i$-th sign pattern of $\tilde{\boldsymbol{X}}u$. Moreover, the authors provide a formula to recover a solution of (17) from a solution of (19).

Using that $P \leq 2r(e(D-1)/r)^r$, where $r = \operatorname{rank}(\tilde{\boldsymbol{X}})$, the authors note that the formulation (19) yields a training algorithm with complexity $O({n_0}^3 r^3 (D/r)^{3r})$. Note that if one fixes $r$, the resulting algorithm runs polynomial time. In particular, fixing $n_0$ fixes the rank of $\tilde{\boldsymbol{X}}$ and results in a polynomial time algorithm as well. In contrast, the algorithm by Arora et al. (2018) discussed in Section 5.1.3 remains exponential even after fixing $n_0$. Moreover, Froese and Hertrich (2023) showed that the training problem is NP-Hard even for fixed $n_0$. This apparent contradiction is explained by two key components of the convex reformulation: the regularization term and the presence of a "large enough" number of hidden neurons. This facilitates the exponential improvement of the training algorithm with respect to Arora et al. (2018).

## 5.3 Frank-Wolfe in DNN training algorithms

Another stream of work that has included components of linear programming in DNN training involves the Frank-Wolfe method. We briefly describe this method in the non-stochastic version next. In this section, we omit the biases $\boldsymbol{b}$ to simplify the notation.

Gradient descent (and its variants) is designed for problems of the form

$$\min_{\boldsymbol{W}\in\mathbb{R}^N} \mathcal{L}(\boldsymbol{W}) \tag{20}$$

and it is based on iterations of the form

$$\boldsymbol{W}(i+1) = \boldsymbol{W}(i) - \alpha_i \nabla\mathcal{L}(\boldsymbol{W}(i)) \tag{21}$$

where $\alpha_i$ is known as the *learning rate*. In the stochastic versions, $\nabla\mathcal{L}(\boldsymbol{W}(i))$ is replaced by a stochastic gradient. In this setting, these algorithms would find a local minimum, which is global when $\mathcal{L}$ is convex.

In the presence of constraints $\boldsymbol{W} \in \Theta$, however, this strategy may not work directly. A regularizing term is typically used in the objective function instead of a constraint, that "encourages" $\boldsymbol{W} \in \Theta$ but does not enforce it. If we strictly require that $\boldsymbol{W} \in \Theta \neq \mathbb{R}^n$, and $\Theta$ is a convex set, one could modify (21) to

$$\boldsymbol{W}(i+1) = \mathrm{proj}_\Theta \left(\boldsymbol{W}(i) - \alpha_i \nabla\mathcal{L}(\boldsymbol{W}(i))\right). \tag{22}$$

and thus ensure that all iterates $\boldsymbol{W}(i) \in \Theta$. Unfortunately, a projection is a costly routine. An alternative to this projection is the Frank-Wolfe method (Frank et al., 1956). Here, a direction $\boldsymbol{d}_i$ is computed via the following linear-objective convex optimization problem

$$\boldsymbol{d}_i \in \arg\min_{\boldsymbol{d}\in\Theta} \boldsymbol{v}_i^\top \boldsymbol{d} \tag{23}$$

where normally $\boldsymbol{v}_i = \nabla\mathcal{L}(\boldsymbol{W}(i))$ (we consider variants below). The update is then computed as

$$\boldsymbol{W}(i+1) = \boldsymbol{W}(i) + \alpha_i(\boldsymbol{d}_i - \boldsymbol{W}(i)), \tag{24}$$

for $\alpha_i \in [0,1]$. Note that, by convexity, we are assured that $\boldsymbol{W}(i+1) \in \Theta$ as long as $\boldsymbol{W}(0) \in \Theta$. In many applications, $\Theta$ is polyhedral, which makes (23) a linear program. Moreover, for simple sets $\Theta$, problem (23) admits closed-form solutions.

In the context of deep neural network training, two notable applications of Frank-Wolfe have appeared. Firstly, the Deep Frank Wolfe algorithm, by Berrada et al. (2018), which modifies iteration (21) with an optimization problem that can be solved using Frank-Wolfe in its dual. Secondly, the use of a stochastic version of Frank-Wolfe in the training problem (11) by Pokutta et al. (2020) and Xie et al. (2020a), which enforces structure in the neural network weights directly. We review these next, starting with the latter.

### 5.3.1 Stochastic Frank-Wolfe

Note that problem (11) is of the form (20) with

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{D}\sum_{i=1}^{D} \ell(f(\tilde{\boldsymbol{x}}_i, \boldsymbol{W}), \tilde{\boldsymbol{y}}_i).$$

We remind the reader that we are omitting the biases in this section to simplify notation, as they can be incorporated as part of $\boldsymbol{W}$.

Usually, some structure of the weights is commonly desired, (e.g. sparsity or boundedness), which traditionally have been incorporated as regularizing terms in the objective, as mentioned above. The recent work by Xie et al. (2020a) and

Pokutta et al. (2020), on the other hand, enforce structure on $\Theta$ directly using Frank-Wolfe —more precisely, stochastic versions of it.

Xie et al. (2020a) use a stochastic Frank-Wolfe approach to impose an $\ell_1$-norm constraint on the weights and biases $\boldsymbol{W}$ when training a neural network with 1 hidden layer. Note that $\ell_1$ constraints are polyhedral. Their algorithm is designed for a general Online Convex Optimization setting, where "losses" are revealed in each iteration. However, in their computational experiments, they included tests in an offline setting given by a DNN training problem.

The approach follows the Frank-Wolfe method described above closely. The key difference lies in the estimation of the stochastic gradient they use, which is not standard and it is one of the most important aspects of the algorithm. Instead of using $\boldsymbol{v}_i = \nabla\mathcal{L}(\boldsymbol{W}(i))$ in (23), the following *stochastic recursive estimator* of the gradient is used:

$$\begin{aligned}\boldsymbol{v}_0 =& \tilde{\nabla}\mathcal{L}(\boldsymbol{W}(0))\\ \boldsymbol{v}_i =& \tilde{\nabla}\mathcal{L}(\boldsymbol{W}(i)) + (1-\rho_i)(v_{i-1} - \tilde{\nabla}\mathcal{L}(\boldsymbol{W}(i-1)))\end{aligned}$$

where $\tilde{\nabla}\mathcal{L}$ is a stochastic gradient, and $\rho_i$ is a parameter. The authors show that the gradient approximation error of this estimator converges to 0 at a sublinear rate, with high probability. This is important for them to analyze the "regret bounds" they provide for the online setting.

The experimental results in Xie et al. (2020a) in DNN training are very positive. They test their approach in the MNIST and CIFAR10 datasets and outperform existing state-of-the-art approaches in terms of suboptimality, training accuracy, and test accuracy.

Pokutta et al. (2020) implement and test several variants of stochastic versions of Frank-Wolfe in the training of neural networks, including the approach by Xie et al. (2020a). Pokutta et al. (2020) focus their experiments on their main proposed variant, which they refer to simply as Stochastic Frank-Wolfe (SFW). This variant uses

$$\boldsymbol{v}_i = (1-\rho_i)\boldsymbol{v}_{i-1} + \rho_i \tilde{\nabla}\mathcal{L}(\boldsymbol{W}(i)),$$

where $\rho_i$ is a momentum parameter. The authors propose many different options for $\Theta$ including $\ell_1, \ell_2$ and $\ell_\infty$ balls, and $K$-sparse polytopes. Of these, only the $\ell_2$ ball is non-polyhedral.

Overall, the computational experiments are promising for SFW. The authors advocate for this algorithm arguing that it provides excellent computational performances while being simple to implement and competitive with other state-of-the-art algorithms.

### 5.3.2 Deep Frank-Wolfe

Another application of Frank-Wolfe within DNN training was proposed by Berrada et al. (2018). While this approach does not make heavy use of linear programming techniques, the application of Frank-Wolfe is quite novel, and they do rely on one linear program needed when performing an update as (24).

The authors note that (21) can also be written as the solution to the following *proximal* problem (Bubeck et al., 2015):

$$\boldsymbol{W}(i+1) = \arg\min_{\boldsymbol{W}} \left\{ \frac{1}{2\alpha_i} \|\boldsymbol{W} - \boldsymbol{W}(i)\|^2 + \mathcal{T}_{\boldsymbol{W}(i)}(\mathcal{L}(\boldsymbol{W})) \right\} \tag{25}$$

where $\mathcal{T}_{\boldsymbol{W}(i)}$ represents the first-order Taylor expansion at $\boldsymbol{W}(i)$. We are omitting regularizing terms since they do not play a fundamental role in the approach; all this discussion can be directly extended to include regularizers. Berrada et al. (2018) note that (25) linearizes the loss function, and propose the following *loss-preserving proximal* problem to replace (25):

$$\boldsymbol{W}(i+1) = \arg\min_{\boldsymbol{W}} \left\{ \frac{1}{2\alpha_i} \|\boldsymbol{W} - \boldsymbol{W}(i)\|^2 + \frac{1}{D} \sum_{i=1}^{D} \ell(\mathcal{T}_{\boldsymbol{W}(i)}(f(\tilde{\boldsymbol{x}}_i, \boldsymbol{W})), \tilde{\boldsymbol{y}}_i) \right\} \tag{26}$$

Using the results by Lacoste-Julien et al. (2013), the authors argue that (26) is amenable to Frank-Wolfe in the dual when $\ell$ is piecewise linear and convex (e.g. the hinge loss). To be more specific, the authors show that in this case, and assuming $\alpha_i = \alpha$, there exists $\boldsymbol{A}, \boldsymbol{b}$ such that the dual of (26) is simply

$$\max_{\beta} \quad \frac{-1}{2\alpha} \|\boldsymbol{A}\beta\|^2 + \boldsymbol{b}^\top \beta \tag{27a}$$

$$\text{s.t.} \quad \boldsymbol{1}^\top \beta = 1 \tag{27b}$$

$$\beta \geq 0 \tag{27c}$$

The authors consider applying Frank-Wolfe to this last problem, and to recover the primal solution using the primal-dual relation $\boldsymbol{W} = -\boldsymbol{A}\beta$, which is a consequence of KKT. The Frank-Wolfe iteration (24) in the notation of (27) would look like

$$\beta_{i+1} = \beta_i + \gamma_i(\boldsymbol{d}_i - \beta_i). \tag{28}$$

Here, $\boldsymbol{d}_i$ is feasible for (27) and obtained using a linear programming oracle, and $\gamma_i$ the Frank-Wolfe step-length. Note that the feasible region of (27) is a simplex: exploiting this, the authors show that an optimal $\gamma_i$ can be computed in closed-form: here, "optimal" refers to a minimizer of (27) when restricted to points of the form $\beta_i + \gamma_i(\boldsymbol{d}_i - \beta_i)$.

With all these considerations, the bottleneck in this application of Frank-Wolfe is obtaining $\boldsymbol{d}_i$; recall that this Frank-Wolfe routine is embedded within a single iteration of the overall training algorithm; therefore, in each iteration of the training algorithm, possibly multiple computations of $\boldsymbol{d}_i$ would be required in order to solve (27) to optimality. To alleviate this, the authors propose to only perform one iteration of Frank-Wolfe: they set $\boldsymbol{d}_0$ to be the stochastic gradient and compute a closed-form expression for $\beta_1$. This is the basic ingredient of the Deep Frank Wolfe (DFW). It is worth noting that this algorithm is not guaranteed to converge, however, its empirical performance is competitive.

Other two important considerations are taken into account the implementation of this algorithm: smoothing of the loss function (as the Hinge loss is piecewise linear) and the adaptation of Nesterov's Momentum to this new setting. We refer the reader to the corresponding article for these details. One of the key features of DFW is that it only requires one hyperparameter ($\alpha$) to be tuned.

The authors test DFW in image classification and natural language inference. Overall, the results obtained by DFW are very positive: in most cases, it can outperform adaptive gradient methods, and it is competitive with SGD while converging faster.

## 5.4 Polyhedral encoding of multiple training problems

One of the questions raised by Arora et al. (2018) (see Section 5.1.3) was whether the dependency on $D$ of their algorithm could be improved since it is typically the largest coefficient in a training problem. This question was studied by Bienstock et al. (2023), who show that, in an approximation setting, a more ambitious goal is achievable: there is a polyhedral encoding of multiple training problems whose size has a mild dependency on $D$.

As in the previous section, we omit the biases $\boldsymbol{b}$ to simplify notation, as all parameters can be included in $\boldsymbol{W}$. Let us assume the class of neural networks $F$ in (11) are restricted to have bounded parameters (we assume they lie in the interval $[-1, 1]$), and let us assume the sample has been normalized in such a way that $(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i) \in [-1, 1]^{n_0+n_{L+1}}$. Furthermore, let $N$ be the dimension of $\Theta$ (the number of parameters in the neural network). With this notation, we define the following.

**Definition 2** *Consider the ERM problem* (11) *with parameters $D, \Theta, \ell, f$ — sample size, parameter space, loss function, network architecture, respectively. For a function $g$, let $\mathcal{K}_\infty(g)$ be the Lipschitz constant of $g$ using the infinity norm. We define the* Architecture Lipschitz Constant *$\mathcal{K}(D, \Theta, \ell, f)$ as*

$$\mathcal{K}(D, \Theta, \ell, f) \doteq \mathcal{K}_\infty(\ell(f(\cdot, \cdot), \cdot)) \tag{29}$$

*over the domain $[-1, 1]^{n_0} \times \Theta \times [-1, 1]^{n_{L+1}}$.*

Using this definition, and the boundedness of parameters, a straightforward approximate training algorithm can be devised whose running time is linear in $D$. Simply do a grid search in the parameters' space, and evaluate all data points in each possible parameter. It is not hard to see that, to achieve $\epsilon$-optimality, such an algorithm would run in time which is linear in $D$ and exponential in $\mathcal{K}(D, \Theta, \ell, f)/\epsilon$. What was proved by Bienstock et al. (2023) is that one can take a step further and represent multiple training problems at the same time.

**Theorem 2 (Bienstock et al. (2023))** *Consider the ERM problem* (11) *with parameters $D, \Theta, \ell, f$, and let $\mathcal{K} := \mathcal{K}(D, \Theta, \ell, f)$ be the corresponding network*

Table 4: Summary of polyhedral encoding sizes for various architectures. DNN refers to a fully-connected Deep Neural Network, CNN to a Convolutional Neural Network, and ResNet to a Residual Network. $G$ is the graph defining the Network, $\Delta$ is the maximum in-degree in $G$, $L$ is the number of hidden layers, and $n_{\max}$ is the maximum width of a layer.

| Type | Loss | Size of polytope | Notes |
|---|---|---|---|
| DNN | Absolute/Quadratic/Hinge | $O\big((n_{L+1}{n_{\max}}^{O(L^2)}/\epsilon)^{n_0+n_{L+1}+N}D\big)$ | $N \in O(\lvert E(G)\rvert)$ |
| DNN | Cross Entropy w/ Soft-Max | $O\big((n_{L+1}\log(n_{L+1}){n_{\max}}^{O(k^2)}/\epsilon)^{n_0+n_{L+1}+N}D\big)$ | $N \in O(\lvert E(G)\rvert)$ |
| CNN | Absolute/Quadratic/Hinge | $O\big((n_{L+1}{n_{\max}}^{O(L^2)}/\epsilon)^{n_0+n_{L+1}+N}D\big)$ | $N \ll \lvert E(G)\rvert$ |
| ResNet | Absolute/Quadratic/Hinge | $O\big((n_{L+1}\Delta^{O(L^2)}/\epsilon)^{n_0+n_{L+1}+N}D\big)$ | |
| ResNet | Cross Entropy w/ Soft-Max | $O\big((n_{L+1}\log(n_{L+1})\Delta^{O(L^2)}/\epsilon)^{n_0+n_{L+1}+N}D\big)$ | |

*architecture. Consider $\epsilon > 0$ arbitrary. There exists a polytope $P_\epsilon$ of size*[5]

$$O(D\,(2\mathcal{K}/\epsilon)^{n_0+n_{L+1}+N})$$

*with the following properties:*

1. *$P_\epsilon$ can be constructed in time $O((2\mathcal{K}/\epsilon)^{n_0+n_{L+1}+N}\,D)$ plus the time required for $O((2\mathcal{K}/\epsilon)^{n_0+n_{L+1}+N})$ evaluations of the loss function $\ell$ and $f$.*

2. *For* any *sample $(\tilde{X},\tilde{Y}) = (\tilde{\boldsymbol{x}}_i,\tilde{\boldsymbol{y}}_i)_{i=1}^D$, $(\tilde{\boldsymbol{x}}_i,\tilde{\boldsymbol{y}}_i) \in [-1,1]^{n_0+n_{L+1}}$, there is a face $\mathcal{F}_{\tilde{X},\tilde{Y}}$ of $P_\epsilon$ such that optimizing a linear function over $\mathcal{F}_{\tilde{X},\tilde{Y}}$ yields an $\epsilon$-approximation to the ERM problem* (11)*.*

3. *The face $\mathcal{F}_{\tilde{X},\tilde{Y}}$ arises by simply substituting-in actual data for the data-variables $x, y$, which is used to fixed variables in the description of $P_\epsilon$.*

This result is very abstract in nature but possesses some interesting features. Firstly, it encodes (approximately) *every* possible training problem arising from data in $[-1,1]^{n_0+n_{L+1}}$ using a benign dependency on $D$: the polytope size depends only linearly on $D$, while a discretized enumeration of all the possible samples of size $D$ would be exponential in $D$. Secondly, every possible ERM problem appears in a *face* of the polytope; this suggests a strong geometric structure across different ERM problems. And lastly, this result is applicable to a wide variety of network architectures; in order to obtain an architecture-specific result, it suffices to compute the corresponding value of $\mathcal{K}$ and plug it in. Regarding this last point, the authors computed the constant $\mathcal{K}$ for various well-known architectures and obtained the results of Table 4.

The proof of this result relies on a graph theoretical concept called *treewidth*. This parameter is used for measuring structured sparsity, and in Bienstock and Muñoz (2018) it was proved that any optimization problem admits an approximate polyhedral reformulation whose size is exponential only in the treewidth parameter. On a high level, the neural network result is obtained by noting that (11) connects different sample points only through a sum; therefore, the

[5] Here, the size of the polytope is the number of variables and constraints describing it.

following reformulation of the optimization problem can be considered, which decouples the different data points:

$$\min_{\boldsymbol{W}\in\Theta,\boldsymbol{L}} \left\{ \frac{1}{D}\sum_{d=1}^{D} \boldsymbol{L}_d \;\middle|\; \boldsymbol{L}_d \,=\, \ell(f(\tilde{\boldsymbol{x}}_d, \boldsymbol{W}), \tilde{\boldsymbol{y}}_d) \quad \forall\, d \in [D] \right\} \tag{30}$$

This reformulation does not seem useful at first, however, it has a *treewidth* that does not depend on $D$, even if the data points are considered variables. From this point, the authors are able to obtain the polytope whose size does not depend exponentially on $D$, and which is capable of encoding all possible ERM problems. The face structure the polytope has is more involved, and we refer the reader to Bienstock et al. (2023) for these details.

It is worth mentioning that the polytope size provided by Bienstock et al. (2023) in the setting of Arora et al. (2018) is

$$O((2\mathcal{K}_\infty(\ell) n_1^{O(1)}/\epsilon)^{(n_0+1)(n_1)} D) \tag{31}$$

where $\mathcal{K}_\infty(\ell)$ is the Lipschitz constant of the loss function with respect to the infinity norm over a specific domain. These two results are not completely comparable, but they give a good idea of how good the size of polytope constructed in Bienstock et al. (2023) is. The dependency on $D$ is better in the polytope size, the polytope encodes multiple training problems, and the result is more general (it applies to almost any architecture); however, the polytope only gives an approximation, and its construction requires boundedness.

## 5.5 Backpropagation through MILP

In the work by Goebbels (2021), a novel use of Mixed-Integer Linear Programming is proposed in training ReLU networks: to serve as an alternative to SGD. This new algorithm works as backpropagation, as it updates the weights of the neural network iteratively starting from the last layer. The key difference is that each update in a layer amounts to solving a MILP.

Let us focus only on one hidden layer at a time (of width $n$), so we can assume we have an architecture as in Figure 11. Furthermore, we assume we have some target output vectors $\{\boldsymbol{T}_d\}_{d=1}^{D}$ (when processing the last hidden layer in the backpropagation, this corresponds to $\{\tilde{\boldsymbol{y}}_d\}_{d=1}^{D}$) and some layer input $\{\boldsymbol{I}_d\}_{d=1}^{D}$ (when processing the last hidden layer, this corresponds to evaluating the neural network on $\{\tilde{\boldsymbol{x}}_d\}_{d=1}^{D}$ up to the second-to-last hidden layer). The algorithm proposed by Goebbels (2021) solves the following optimization problem to update the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ of the given layer:

$$
\begin{aligned}
\min_{\boldsymbol{W},\hat{\boldsymbol{h}},\boldsymbol{b},\boldsymbol{h},\boldsymbol{z}} \quad & \sum_{d=1}^{D}\sum_{j=1}^{n} |\boldsymbol{T}_{d,j} - \boldsymbol{h}_{d,j}| && && \text{(32a)}\\
\text{s.t.} \quad & \hat{\boldsymbol{h}}_{d,j} = (\boldsymbol{W}\boldsymbol{I}_d)_j + \boldsymbol{b}_j && d=1,\dots,D,\ j=1,\dots,n && \text{(32b)}\\
& \hat{\boldsymbol{h}}_{d,j} \le M\boldsymbol{z}_{d,j} && d=1,\dots,D,\ j=1,\dots,n && \text{(32c)}\\
& \hat{\boldsymbol{h}}_{d,j} \ge -M(1-\boldsymbol{z}_{d,j}) && d=1,\dots,D,\ j=1,\dots,n && \text{(32d)}\\
& |\hat{\boldsymbol{h}}_{d,j} - \boldsymbol{h}_{d,j}| \le M(1-\boldsymbol{z}_{d,j}) && d=1,\dots,D,\ j=1,\dots,n && \text{(32e)}\\
& \boldsymbol{h}_{d,j} \le M\boldsymbol{z}_{d,j} && d=1,\dots,D,\ j=1,\dots,n && \text{(32f)}\\
& \boldsymbol{h}_{d,j} \ge 0 && d=1,\dots,D,\ j=1,\dots,n && \text{(32g)}\\
& \boldsymbol{z}_{d,j} \in \{0,1\} && d=1,\dots,D,\ j=1,\dots,n. && \text{(32h)}
\end{aligned}
$$

Here $M$ is a large constant that is assumed to bound the input to any neuron. Note that problem (32) can easily be linearized. This optimization problem finds the weights ($\boldsymbol{W}$) and biases ($\boldsymbol{b}$) that minimize the difference between the "real" output of the network for each sample ($\boldsymbol{h}_d$) and the target output ($\boldsymbol{T}_d$). The auxiliary variables $\hat{\boldsymbol{h}}_{d,j}$ represent the input to the each neuron —so $\boldsymbol{h}_{d,j} = \sigma(\hat{\boldsymbol{h}}_{d,j})$— and $\boldsymbol{z}_{d,j}$ indicates if the $j$-th neuron is activated on input $\boldsymbol{I}_d$.

When processing intermediate layers, the definition $\boldsymbol{I}_d$ can easily be adapted from what we mentioned above. However, the story is different for the case of $\boldsymbol{T}_d$. When processing the last layer, as previously mentioned, $\boldsymbol{T}_d$ simply corresponds to $\tilde{\boldsymbol{y}}_d$. For intermediate layers, to define $\boldsymbol{T}_d$, the author proposes to use a similar optimization problem to (32), but leaving $\boldsymbol{W}$ and $\boldsymbol{b}$ fixed and having $\boldsymbol{I}_d$ as variables; this defines "optimal inputs" of a layer. These optimal inputs are then used as target outputs $\boldsymbol{T}_d$ when processing the preceding layer, and thus the algorithm is iterated. For details, see Goebbels (2021).

The computational results in that paper show that a similar level of accuracy to that of gradient descent can be achieved. However, the use of potentially expensive MILPs impairs the applicability of this approach to large networks. Nonetheless, it shows an interesting new avenue for training whose running times may be improved in future implementations.

## 5.6 Training binarized neural networks using MILP

As mentioned before, the training problem of a DNN is an unrestricted non-convex optimization problem, which is typically continuous as the weights and biases frequently are allowed to have any real value. Nonetheless, if the weights and biases are required to be integer-valued, the training problem becomes a discrete optimization problem, for which gradient-descent-based methods may find some difficulties in their applicability.

In this context, Icarte et al. (2019) proposed a MILP formulation for the training problem of binarized neural networks (BNNs): these are neural networks where the weights and biases are restricted to be in $\{-1, 0, 1\}$ and where the activations are LTU (i.e. sign functions). Later on, Thorbjarnarson and Yorke-Smith (2020, 2023) used a similar technique to allow more general integer-valued weights. We review the core feature in these formulations that yield a *linear* formulation of the training problem.

Let us focus on an intermediate layer $i$ with width $n$, and let us omit biases to simplify the discussion. Using a DNN's layer-wise architecture, one usually aims at describing:

$$\hat{\boldsymbol{h}}^i_{d,j} = (\boldsymbol{W}^i \boldsymbol{h}^{i-1}_d)_j \qquad d = 1, \ldots, D,\ j = 1, \ldots, n \tag{33a}$$

$$\boldsymbol{h}^i_{d,j} = \sigma(\hat{\boldsymbol{h}}^i_{d,j}) \qquad d = 1, \ldots, D,\ j = 1, \ldots, n. \tag{33b}$$

We remind the reader that $D$ is the cardinality of the training set. Additionally, for each data point indexed by $d$ and each layer $i$, each variable $\boldsymbol{h}^i_d$ is the output vector of all the neurons of the layer and each variable $\hat{\boldsymbol{h}}^i_{d,j}$ is the input of neuron $j$. Besides the difficulty posed by the activation function, one important issue with system (33) is the non-linearity of the products between the $\boldsymbol{W}$ and $\boldsymbol{h}$ variables. Nonetheless, this issue disappears when each entry of $\boldsymbol{W}$ and $\boldsymbol{h}$ is bounded and integer, as in the case of BNNs.

Let us begin with reformulating (33b). We can introduce auxiliary variables $\boldsymbol{u}^i_{d,j} \in \{0, 1\}$ that will indicate if the neuron is active. We also introduce a tolerance $\varepsilon > 0$ to determine the activity of a neuron. Using this, we can (approximately) reformulate (33b) *linearly* using big-M constraints:

$$\boldsymbol{h}^i_{d,j} = 2\boldsymbol{u}^i_{d,j} - 1 \qquad d = 1, \ldots, D,\ j = 1, \ldots, n \tag{34a}$$

$$\hat{\boldsymbol{h}}^i_{d,j} \geq -M(1 - \boldsymbol{u}^i_{d,j}) \qquad d = 1, \ldots, D,\ j = 1, \ldots, n \tag{34b}$$

$$\hat{\boldsymbol{h}}^i_{d,j} \leq -\varepsilon + M\boldsymbol{u}^i_{d,j} \qquad d = 1, \ldots, D,\ j = 1, \ldots, n \tag{34c}$$

where $M$ is a large constant. As for (33a), note that

$$\hat{\boldsymbol{h}}^i_{d,j} = \sum_{k=1} \boldsymbol{W}^i_{j,k} \boldsymbol{h}^{i-1}_{d,k}.$$

Therefore, using (34a), we see that it suffices to describe each product $\boldsymbol{W}^i_{j,k} \boldsymbol{u}^{i-1}_{d,k}$ linearly. We can introduce new variables $\boldsymbol{z}^i_{j,k,d}$ and note that

$$\boldsymbol{z}^{i-1}_{j,k,d} = \boldsymbol{W}^i_{j,k} \boldsymbol{u}^{i-1}_{d,k}$$

if and only if the three variables satisfy

$$\begin{aligned} |\boldsymbol{z}^{i-1}_{j,k,d}| &\leq \boldsymbol{u}^{i-1}_{d,k} \\ |\boldsymbol{z}^{i-1}_{j,k,d} - \boldsymbol{W}^i_{j,k}| &\leq 1 - \boldsymbol{u}^{i-1}_{d,k} \\ \boldsymbol{u}^{i-1}_{d,k} &\in \{0, 1\}. \end{aligned}$$

This last system can be easily converted to a linear system, and thus the training problem in this setting can be cast as a mixed-integer linear optimization problem.

Other works have also relied on similar formulations to train neural networks. Icarte et al. (2019) introduce different objective functions that can be used along with the linear system to produce a MILP that can train BNNs. They also introduce a Constraint-Programming-based model and a hybrid model, and then compare all of them computationally. Thorbjarnarson and Yorke-Smith (2020) introduce more MILP-based training models that leverage piecewise linear approximations of well-known non-linear loss functions and that can handle integer weights beyond $\{-1, 0, 1\}$. A similar setting is studied by Sildir and Aydin (2022), where piecewise linear approximations of non-linear activations are used, and integer weights are exploited to formulate the training problem as a MILP. Finally, Bernardelli et al. (2022) rely on a multi-objective MIP model for training BNNs; from here, they create a BNN ensemble to produce robust classifiers.

From these articles, we can conclude that the MILP-based approach to training their neural networks can result in high-quality neural networks, especially in terms of generalization. However, many of these MILP-based methods currently do not scale well, as opposed to gradient-descent-based methods. We believe that, even though there are some theoretical limitations to the efficiency of MILP-based methods, there is a considerable practical improvement potential with using them in neural network training.

# 6 Conclusion

The rapid advancement of neural networks and their ubiquity has given rise to numerous new challenges and opportunities in deep learning: we need to design them in more reliable ways, to better understand their limits, and to test their robustness, among other challenges. While, traditionally, continuous optimization has been the predominant technology used in the optimization tasks in deep learning, some of these new challenges have made discrete optimization tools gain a remarkable importance.

In this survey, we have reviewed multiple areas where polyhedral theory and linear optimization have played a critical role. For example, in understanding the expressiveness of neural networks, in optimizing trained neural networks (e.g. for verification purposes), and even in designing new training algorithms. We hope this survey can provide perspective in a rapidly-changing field, and motivate further developments in both deep learning and discrete optimization. There is still much to be explored in the intersection of these fields.

**Acknowledgments** We thank Christian Tjandraatmadja and Toon Tran for early feedback on the manuscript and asking questions that helped shaping it.

Thiago Serra was supported by the National Science Foundation (NSF) award IIS 2104583. Calvin Tsay was supported by the Engineering & Physical Sciences Research Council (EPSRC) grant EP/T001577/1.

# References

M. Abrahamsen, L. Kleist, and T. Miltzow. Training neural networks is er-complete. In *Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. In *International Conference on Learning Representations (ICLR) Workshop*, 2015.

A. Amrami and Y. Goldberg. A simple geometric proof for the benefit of depth in ReLU networks. *arXiv:2101.07126*, 2021.

R. Anderson, J. Huchette, C. Tjandraatmadja, and J. Vielma. Strong mixed-integer programming formulations for trained neural networks. In *Integer Programming and Combinatorial Optimization (IPCO)*, 2019.

R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. P. Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1-2):3–39, 2020.

C. Anil, J. Lucas, and R. Grosse. Sorting out Lipschitz function approximation. In *International Conference on Machine Learning (ICML)*, 2019.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2017.

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations (ICLR)*, 2018.

S. Aziznejad, H. Gupta, J. Campos, and M. Unser. Deep neural networks with trainable activations and controlled Lipschitz constant. *IEEE Transactions on Signal Processing*, 68:4688–4699, 2020.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3–44, 1998.

E. Balas. *Disjunctive Programming.* Springer Cham, 2018.

E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming*, 58(1-3):295–324, 1993.

E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.

R. Balestriero and R. G. Baraniuk. A spline theory of deep networks. In *International Conference on Machine Learning (ICML)*, 2018.

M. Balunović and M. Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations (ICLR)*, 2020.

B. Batten, P. Kouvaros, A. Lomuscio, and Y. Zheng. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2184–2190, 2021.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends®in Machine Learning*, 2(1):1–127, 2009.

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

K. P. Bennett. Decision tree construction via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1992.

K. P. Bennett and O. L. Mangasarian. Neural network training via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1990.

K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1 (1):23–34, 1992.

E. Benussi, A. Patane, M. Wicker, L. Laurenti, and M. Kwiatkowska. Individual fairness guarantees for neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 651–658, 2022.

D. Bergman, T. Huang, P. Brooks, A. Lodi, and A. U. Raghunathan. Janos: an integrated predictive and prescriptive modeling framework. *INFORMS Journal on Computing*, 34(2):807–816, 2022.

A. M. Bernardelli, S. Gualandi, H. C. Lau, and S. Milanesi. The bemi stardust: a structured ensemble of binarized neural networks. *arXiv preprint arXiv:2212.03659*, 2022.

L. Berrada, A. Zisserman, and M. P. Kumar. Deep Frank-Wolfe for neural network optimization. *arXiv:1811.07591*, 2018.

D. Bertschinger, C. Hertrich, P. Jungeblut, T. Miltzow, and S. Weber. Training fully connected neural networks is $\exists\mathbb{R}$-complete. *arXiv:2204.01368*, 2022.

A. Bhosekar and M. Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering*, 108:250–267, 2018.

M. Bianchini and F. Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 2014.

G. Biau, M. Sangnier, and U. Tanielian. Some theoretical insights into Wasserstein GANs. *Journal of Machine Learning Research*, 22, 2021.

D. Bienstock and G. Muñoz. Lp formulations for polynomial optimization problems. *SIAM Journal on Optimization*, 28(2):1121–1150, 2018.

D. Bienstock, G. Muñoz, and S. Pokutta. Principled deep neural network training through linear programming. *Discrete Optimization*, 49:100795, 2023.

A. L. Blum and R. L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127, 1992.

P. Bohra, J. Campos, H. Gupta, S. Aziznejad, and M. Unser. Learning activation functions in deep (spline) neural networks. *IEEE Open Journal of Signal Processing*, 1:295–309, 2020.

P. Bonami, A. Lodi, A. Tramontani, and S. Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 151:191–223, 2015.

D. Boob, S. S. Dey, and G. Lan. Complexity of training ReLU neural network. *Discrete Optimization*, 44:100620, 2022.

E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. 1990.

S. Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 2015.

R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. Torr, and M. Pawan Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124, pages 370–379, 2020a.

R. Bunel, P. Mudigonda, I. Turkaslan, P. Torr, J. Lu, and P. Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020b.

R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda. A unified view of piecewise linear neural network verification. *Neural Information Processing Systems (NeurIPS)*, 31, 2018.

R. R. Bunel, O. Hinder, S. Bhojanapalli, and K. Dvijotham. An efficient nonconvex reformulation of stagewise convex optimization problems. *Neural Information Processing Systems (NeurIPS)*, 33:8247–8258, 2020c.

R.-A. Burtea and C. Tsay. Safe deployment of reinforcement learning using deterministic optimization over neural networks. In *Computer Aided Chemical Engineering*, volume 52, pages 1643–1648. Elsevier, 2023.

J. Cai, K.-N. Nguyen, N. Shrestha, A. Good, R. Tu, X. Yu, S. Zhe, and T. Serra. Getting away with more network pruning: From sparsity to geometry and linear regions. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2023.

F. Ceccon, J. Jalving, J. Haddad, A. Thebelt, C. Tsay, C. D. Laird, and R. Misener. Omlt: Optimization & machine learning toolkit. *Journal of Machine Learning Research*, 23(349):1–8, 2022.

V. Charisopoulos and P. Maragos. A tropical approach to neural networks with piecewise linear activations. *arXiv:1805.08749*, 2018.

A. Chaudhry, N. Khan, P. Dokania, and P. Torr. Continual learning in low-rank orthogonal subspaces. In *Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

H. Chen, Y. G. Wang, and H. Xiong. Lower and upper bounds for numbers of linear regions of graph convolutional networks. *arXiv:2206.00228*, 2022a.

K.-L. Chen, H. Garudadri, and B. D. Rao. Improved bounds on neural complexity for representing piecewise linear functions. In *Neural Information Processing Systems (NeurIPS)*, 2022b.

S. Chen, A. R. Klivans, and R. Meka. Learning deep ReLU networks is fixed-parameter tractable. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707. IEEE, 2022c.

T. Chen, J.-B. Lasserre, V. Magron, and E. Pauwels. Semialgebraic optimization for Lipschitz constants of ReLU networks. In *Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

W. Chen, X. Gong, and Z. Wang. Neural architecture search on ImageNet in four GPU hours: A theoretically inspired perspective. In *International Conference on Learning Representations (ICLR)*, 2021a.

W. Chen, X. Gong, Y. Wei, H. Shi, Z. Yan, Y. Yang, and Z. Wang. Understanding and accelerating neural architecture search with training-free and theory-grounded metrics. *arXiv:2108.11939*, 2021b.

C. Cheng, G. Nührenberg, and H. Ruess. Maximum resilience of artificial neural networks. In *Automated Technology for Verification and Analysis (ATVA)*, pages 251–268, 2017.

C.-H. Cheng, G. Nührenberg, C.-H. Huang, and H. Ruess. Verification of binarized neural networks via inter-neuron factoring: (short paper). In *International Conference on Verified Software: Theories, Tools, and Experiments (VSTTE)*, pages 279–290. Springer, 2018.

M.-S. Cheon. An outer-approximation guided optimization approach for constrained neural network inverse problems. *Mathematical Programming*, 196 (1-2):173–202, 2022.

L. Chu, X. Hu, J. Hu, L. Wang, and J. Pei. Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2018.

D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. Multi column deep neural network for traffic sign classification. *Neural Networks*, 2012.

S. Cohan, N. H. Kim, D. Rolnick, and M. van de Panne. Understanding the evolution of linear regions in deep reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2022.

R. Collobert. *Large Scale Machine Learning*. PhD thesis, University Paris 6, 2004.

M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. *Neural Information Processing Systems (NeurIPS)*, 28, 2015.

F. Craighero, F. Angaroni, A. Graudenzi, F. Stella, and M. Antoniotti. Investigating the compositional structure of deep neural networks. In *Machine Learning, Optimization, and Data Science (LOD)*, pages 322–334, 2020a.

F. Craighero, F. Angaroni, A. Graudenzi, F. Stella, and M. Antoniotti. Understanding deep learning with activation pattern diagrams. In *Proceedings of the Italian Workshop on Explainable Artificial Intelligence co-located with 19th International Conference of the Italian Association for Artificial Intelligence*, 2020b.

F. Croce and M. Hein. A randomized gradient-free attack on ReLU networks. In *German Conference on Pattern Recognition (GCPR)*, 2018.

F. Croce, M. Andriushchenko, and M. Hein. Provable robustness of relu networks via maximization of linear regions. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

F. Croce, J. Rauber, and M. Hein. Scaling up the randomized gradient-free adversarial attack reveals overestimation of robustness using established attacks. *International Journal of Computer Vision*, 128:1028–1046, 2020.

K. L. Croxton, B. Gendron, and T. L. Magnanti. A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems. *Management Science*, 49(9):1268–1273, 2003.

F. E. Curtis and K. Scheinberg. Optimization methods for supervised machine learning: From linear models to deep learning. In *INFORMS TutORials in Operations Research*, pages 89–114. INFORMS, 2017.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 1989.

E. Danna, M. Fenelon, Z. Gu, and R. Wunderling. Generating multiple solutions for mixed integer programming problems. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 280–294. 2007.

G. B. Dantzig. On the significance of solving linear programming problems with some integer variables. *Econometrica, Journal of the Econometric Society*, pages 30–44, 1960.

G. B. Dantzig and B. C. Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. S. Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Neural Information Processing Systems (NeurIPS)*, 33: 5318–5331, 2020.

I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova. Nonlinear approximation and (deep) ReLU networks. *Constructive Approximation*, 55: 127–172, 2022.

A. De Palma, H. Behl, R. R. Bunel, P. Torr, and M. P. Kumar. Scaling the convex barrier with active sets. In *International Conference on Learning Representations (ICLR)*, 2021.

A. Delarue, R. Anderson, and C. Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Neural Information Processing Systems (NeurIPS)*, 33:609–620, 2020.

Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim. An analysis of adversarial attacks and defenses on autonomous driving models. In *2020 IEEE international conference on pervasive computing and communications (PerCom)*, pages 1–10. IEEE, 2020.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

S. S. Dey, G. Wang, and Y. Xie. Approximation algorithms for training one-node relu neural networks. *IEEE Transactions on Signal Processing*, 68:6696–6706, 2020.

S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. A comprehensive survey and performance analysis of activation functions in deep learning. *arXiv:2109.14545*, 2021.

S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep feedforward networks. In *NASA Formal Methods: 10th International Symposium, (NFM)*, 2018.

K. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelovic, B. O'Donoghue, J. Uesato, and P. Kohli. Training verified learners with learned verifiers. *arXiv:1805.10265*, 2018a.

K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 1, page 3, 2018b.

N. Dym, B. Sober, and I. Daubechies. Expression of fractals through neural network functions. *IEEE Journal on Selected Areas in Information Theory*, 1(1):57–66, 2020.

R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis (ATVA)*, pages 269–286. Springer, 2017.

M. ElAraby, G. Wolf, and M. Carvalho. Identifying efficient sub-networks using mixed integer programming. In *OPT Workshop*, 2020.

T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21, 2019.

T. Ergen and M. Pilanci. Convex geometry of two-layer relu networks: Implicit autoencoding and interpretable models. In S. Chiappa and R. Calandra, editors, *International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4024–4033. PMLR, 26–28 Aug 2020.

T. Ergen and M. Pilanci. Convex geometry and duality of over-parameterized neural networks. *The Journal of Machine Learning Research*, 22(1):9646–9708, 2021a.

T. Ergen and M. Pilanci. Global optimality beyond two layers: Training deep relu networks via convex programs. In *International Conference on Machine Learning (ICLR)*, pages 2993–3003. PMLR, 2021b.

T. Ergen and M. Pilanci. Implicit convex regularizers of cnn architectures: Convex optimization of two-and three-layer networks in polynomial time. In *International Conference on Learning Representations (ICLR)*, 2021c.

T. Ergen and M. Pilanci. Path regularization: A convexity and sparsity inducing regularization for parallel relu networks. *arXiv preprint arXiv:2110.09548*, 2021d.

T. Ergen and M. Pilanci. Revealing the structure of deep neural networks via convex duality. In *International Conference on Machine Learning*, pages 3004–3014. PMLR, 2021e.

T. Ergen, A. Sahiner, B. Ozturkler, J. M. Pauly, M. Mardani, and M. Pilanci. Demystifying batch normalization in reLU networks: Equivalent convex optimization models and implicit regularization. In *International Conference on Learning Representations*, 2022.

T. Ergen, H. I. Gulluk, J. Lacotte, and M. Pilanci. Globally optimal training of neural networks with threshold activation functions. In *International Conference on Learning Representations (ICLR)*, 2023.

K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

F.-L. Fan, R. Lai, and G. Wang. Quasi-equivalence of width and depth of neural networks. *arXiv:2002.02515*, 2020.

F.-L. Fan, W. Huang, X. Zhong, L. Ruan, T. Zeng, H. Xiong, and F. Wang. Deep relu networks have surprisingly simple polytopes. *arXiv:2305.09145*, 2023.

M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. J. Pappas. Efficient and accurate estimation of Lipschitz constants for deep neural networks. In *Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

M. Fazlyab, M. Morari, and G. J. Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 67(1):1–15, 2020.

J. Ferlez and Y. Shoukry. AReN: Assured ReLU NN architecture for model predictive control of LTI systems. In *HSCC*, 2020.

C. Ferrari, M. N. Mueller, N. Jovanović, and M. Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations (ICLR)*, 2022.

S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam, and I. S. Kohane. Adversarial attacks on medical machine learning. *Science*, 363(6433): 1287–1289, 2019.

M. Fischetti and J. Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 2018.

J. Fourier. Solution d'une question particuliére du calcul des inégalités. *Nouveau Bulletin des Sciences par la Société Philomatique de Paris*, 1826.

M. Frank, P. Wolfe, et al. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

V. Froese and C. Hertrich. Training neural networks is NP-hard in fixed dimension. *arXiv preprint arXiv:2303.17045*, 2023.

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

K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3), 1989.

M. Gamba, S. Carlsson, H. Azizpour, and M. Björkman. Hyperplane arrangements of trained ConvNets are biased. *arXiv:2003.07797*, 2020.

M. Gamba, A. Chmielewski-Anders, J. Sullivan, H. Azizpour, and M. Björkman. Are all linear regions created equal? In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

C. Gambella, B. Ghaddar, and J. Naoum-Sawaya. Optimization problems for machine learning: A survey. *European Journal of Operational Research*, 290 (3):807–828, 2021.

J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid. VectorNet: Encoding HD maps and agent dynamics from vectorized representation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

B. Geißler, A. Martin, A. Morsi, and L. Schewe. Using piecewise linear functions for solving MINLPs. In *Mixed Integer Nonlinear Programming*, pages 287–314. Springer, 2012.

L. Glass, W. Hilali, and O. Nelles. Compressing interpretable representations of piecewise linear neural networks using neuro-fuzzy models. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

S. Goebbels. Training of ReLU activated multilayerd neural networks with mixed integer linear programs. Technical report, Hochschule Niederrhein, Fachbereich Elektrotechnik & Informatik, 2021.

S. Goel, A. Klivans, P. Manurangsi, and D. Reichman. Tight hardness results for training depth-2 relu networks. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

M. Goerigk and J. Kurtz. Data-driven robust optimization using deep neural networks. *Computers & Operations Research*, 151:106087, 2023.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *International Conference on Machine Learning (ICML)*, 2013.

I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Neural Information Processing Systems (NeurIPS)*, volume 27, 2014.

D. Gopinath, H. Converse, C. S. Pasareanu, and A. Taly. Property inference for deep neural networks. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019.

A. Goujon, A. Etemadi, and M. Unser. The role of depth, width, and activation complexity in the number of linear regions of neural networks. *arXiv:2206.08615*, 2022.

S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv:1810.12715*, 2018.

A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2014.

J. E. Grigsby and K. Lindsey. On transversality of bent hyperplane arrangements and the topological expressiveness of ReLU neural networks. *SIAM Journal on Applied Algebra and Geometry*, 6(2), 2022.

J. E. Grigsby, K. Lindsey, and D. Rolnick. Hidden symmetries of ReLU networks. In *International Conference on Machine Learning (ICML)*, 2023.

B. Grimstad and H. Andersson. ReLU networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, 2019.

I. E. Grossmann and J. P. Ruiz. Generalized disjunctive programming: A framework for formulation and alternative algorithms for MINLP optimization. In *Mixed Integer Nonlinear Programming*, pages 93–115, New York, NY, 2012. Springer New York.

R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. Douglas, and S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405, 2000.

S. Han and A. Gómez. Single-neuron convexification for binarized neural networks, 2021. URL `https://optimization-online.org/?p=17148`.

B. Hanin and D. Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning (ICML)*, 2019a.

B. Hanin and D. Rolnick. Deep ReLU networks have surprisingly few activation patterns. In *Neural Information Processing Systems (NeurIPS)*, volume 32, 2019b.

B. Hanin and M. Sellke. Approximating continuous functions by ReLU nets of minimal width. *arXiv:1710.11278*, 2017.

V. Hashemi, P. Kouvaros, and A. Lomuscio. OSIP: Tightened bound propagation for the verification of ReLU neural networks. In *International Conference on Software Engineering and Formal Methods (SEFM)*, pages 463–480. Springer, 2021.

F. He, S. Lei, J. Ji, and D. Tao. Neural networks behave as hash encoders: An empirical study. *arXiv:2101.05490*, 2021.

J. He, L. Li, J. Xu, and C. Zheng. ReLU deep neural networks and linear finite elements. *Journal of Computational Mathematics*, 38:502–527, 2020.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

P. Henriksen and A. Lomuscio. DEEPSPLIT: an efficient splitting method for neural network verification via indirect effect analysis. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2549–2555, 2021.

P. Henriksen, F. Leofante, and A. Lomuscio. Repairing misclassifications in neural networks using limited data. In *ACM/SIGAPP Symposium On Applied Computing (SAC)*, 2022.

C. Hertrich, A. Basu, M. D. Summa, and M. Skutella. Towards lower bounds on the depth of ReLU neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2021.

G. Hinton, L. Deng, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 2012.

P. Hinz. Using activation histograms to bound the number of affine regions in ReLU feed-forward neural networks. *arXiv:2103.17174*, 2021.

P. Hinz and S. van de Geer. A framework for the construction of upper bounds on the number of affine linear regions of ReLU feed-forward neural networks. *IEEE Transactions on Information Theory*, 65(11):7304–7324, 2019.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79: 2554–2558, 1982.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 1989.

T. Hu, Z. Shang, and G. Cheng. Sharp rate of convergence for deep neural network classifiers under the teacher-student setting. *arXiv:2001.06892*, 2020a.

X. Hu, W. Liu, J. Bian, and J. Pei. Measuring model complexity of neural networks with curve activation functions. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2020b.

X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian. Model complexity of deep learning: a survey. *Knowledge and Information Systems*, 63:2585–2619, 2021.

X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

J. Huchette and J. P. Vielma. Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools. *Operations Research*, 2022.

T. Huster, C.-Y. J. Chiang, and R. Chadha. Limitations of the Lipschitz constant as a defense against adversarial examples. In *ECML PKDD Workshops*, 2018.

W.-L. Hwang and A. Heinecke. Un-rectifying non-linear networks for signal representation. *IEEE Transactions on Signal Processing*, 68:196–210, 2020.

R. T. Icarte, L. Illanes, M. P. Castro, A. A. Cire, S. A. McIlraith, and J. C. Beck. Training binarized neural networks using mip and cp. In *International Conference on Principles and Practice of Constraint Programming*, pages 401–417. Springer, 2019.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

R. G. Jeroslow and J. K. Lowe. *Modelling with integer variables.* Springer, 1984.

K. Jia and M. Rinard. Efficient exact verification of binarized neural networks. *Neural Information Processing Systems (NeurIPS)*, 33:1782–1795, 2020.

T. T. Johnson, D. M. Lopez, P. Musau, H.-D. Tran, E. Botoeva, F. Leofante, A. Maleki, C. Sidrane, J. Fan, and C. Huang. ARCH-COMP20 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In *International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74, pages 107–139, 2020.

M. Jordan and A. G. Dimakis. Exactly computing the local Lipschitz constant of ReLU networks. In *Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

M. Jordan, J. Lewis, and A. G. Dimakis. Provable certificates for adversarial examples: Fitting a ball in the union of polytopes. In *Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

B. Karg and S. Lucia. Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics*, 50(9):3866–3878, 2020. doi: 10.1109/TCYB.2020.2999556.

G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification (CAV)*, pages 97–117. Springer, 2017.

G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification (CAV)*, pages 443–452. Springer, 2019.

J. Katz, I. Pappas, S. Avraamidou, and E. N. Pistikopoulos. Integrating deep learning models and multiparametric programming. *Computers & Chemical Engineering*, 136:106801, 2020.

C. Keup and M. Helias. Origami in N dimensions: How feed-forward networks manufacture linear separability. *arXiv:2203.11355*, 2022.

S. Khalife and A. Basu. Neural networks with linear threshold activations: structure and algorithms. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 347–360. Springer, 2022.

H. Khedr, J. Ferlez, and Y. Shoukry. Effective formal verification of neural networks using the geometry of linear regions. *arXiv:2006.10864*, 2020.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

A. Kody, S. Chevalier, S. Chatzivasileiadis, and D. Molzahn. Modeling the ac power flow equations with optimally compact neural networks: Application to unit commitment. *Electric Power Systems Research*, 213:108282, 2022.

P. Kouvaros, T. Kyono, F. Leofante, A. Lomuscio, D. Margineantu, D. Osipychev, and Y. Zheng. Formal analysis of neural network-based systems in the aircraft domain. In *International Symposium on Formal Methods (FM)*, pages 730–740. Springer, 2021.

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NeurIPS)*, volume 25, 2012.

J. Kronqvist, R. Misener, and C. Tsay. Between steps: Intermediate relaxations between big-M and convex hull formulations. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2021.

J. Kronqvist, R. Misener, , and C. Tsay. P-split formulations: A class of intermediate formulations between big-M and convex hull for disjunctive constraints. *arXiv:2202.05198*, 2022.

A. Kumar, T. Serra, and S. Ramalingam. Equivalent and approximate transformations of deep neural networks. *arXiv:1905.1142*, 2019.

S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *International Conference on Machine Learning (ICML)*, pages 53–61, 2013.

J. Lan, Y. Zheng, and A. Lomuscio. Tight neural network verification via semidefinite relaxations and linear reformulations. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 7272–7280, 2022.

F. Latorre, P. Rolland, and V. Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. In *International Conference on Learning Representations (ICLR)*, 2020.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G. Montavon, G. Orr, and K. Müller, editors, *Neural Networks: Tricks of the Trade*. Springer, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521, 2015.

G.-H. Lee, D. Alvarez-Melis, and T. S. Jaakkola. Towards robust, locally linear deep networks. In *International Conference on Learning Representations (ICLR)*, 2019.

J. Lee and D. Wilson. Polyhedral methods for piecewise-linear functions I: the lambda method. *Discrete Applied Mathematics*, 108(3):269–285, 2001.

F. Leofante, N. Narodytska, L. Pulina, and A. Tacchella. Automated verification of neural networks: Advances, challenges and perspectives. *arXiv:1805.09938*, 2018.

L. Li, T. Xie, and B. Li. Sok: Certified robustness for deep neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 94–115. IEEE Computer Society, 2022.

X. Liang and J. Xu. Biased ReLU neural networks. *Neurocomputing*, 423:71–79, 2021.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (in Finnish). Master's thesis, Univ. Helsinki, 1970.

W. Little. The existence of persistent states in the brain. *Mathematical Biosciences*, 19:101–120, 1974.

B. Liu and Y. Liang. Optimal function approximation with ReLU neural networks. *Neurocomputing*, 435:216–227, 2021.

C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

X. Liu, X. Han, N. Zhang, and Q. Liu. Certified monotonic neural networks. In *Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

M. Lombardi, M. Milano, and A. Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244:343–367, 2017.

A. Lomuscio and L. Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. *arXiv:1706.07351*, 2017.

A. Loukas, M. Poiitis, and S. Jegelka. What training reveals about neural network complexity. In *Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

L. Lueg, B. Grimstad, A. Mitsos, and A. M. Schweidtmann. reluMIP: Open source tool for MILP optimization of ReLU neural networks, 2021. URL `https://github.com/ChemEngAI/ReLU_ANN_MILP`.

Z. Lyu, C.-Y. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel. Fastened crown: Tightened neural network robustness certificates. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 5037–5044, 2020.

A. Maas, A. Hannun, and A. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

J. Makhoul, R. Schwartz, and A. El-Jaroudi. Classification capabilities of two-layer neural nets. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1989.

E. Malach and S. Shalev-Shwartz. Is deeper better only when shallow is good? In *Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

O. L. Mangasarian. Mathematical programming in neural networks. *ORSA Journal on Computing*, 5(4):349–360, 1993.

D. Maragno, H. Wiberg, D. Bertsimas, S. I. Birbil, D. d. Hertog, and A. Fajemisin. Mixed-integer optimization with constraint learning. *arXiv:2111.04469*, 2021.

D. Maragno, J. Kurtz, T. E. Röber, R. Goedhart, Ş. I. Birbil, and D. d. Hertog. Finding regions of counterfactual explanations via robust optimization. *arXiv:2301.11113*, 2023.

P. Maragos, V. Charisopoulos, and E. Theodosis. Tropical geometry and machine learning. *Proceedings of the IEEE*, 109(5):728–755, 2021.

M. Masden. Algorithmic determination of the combinatorial structure of the linear regions of ReLU neural networks. *arXiv:2207.07696*, 2022.

K. Matoba, N. Dimitriadis, and F. Fleuret. The theoretical expressiveness of maxpooling. *arXiv:2203.01016*, 2022.

J. Matousek. *Lectures on Discrete Geometry*, volume 212. Springer Science & Business Media, 2002.

K. McBride and K. Sundmacher. Overview of surrogate modeling in chemical process engineering. *Chemie Ingenieur Technik*, 91(3):228–239, 2019.

W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

H. N. Mhaskar and T. Poggio. Function approximation by deep networks. *Communications on Pure & Applied Analysis*, 19(8):4085–4095, 2020.

M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

M. Mirman, T. Gehr, and M. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, volume 80, pages 3578–3586, 2018.

R. Misener and C. A. Floudas. Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Mathematical Programming*, 136(1):155–182, 2012.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

G. Montúfar. Notes on the number of linear regions of deep neural networks. In *Sampling Theory and Applications (SampTA)*, 2017.

G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Neural Information Processing Systems (NeurIPS)*, volume 27, 2014.

G. Montúfar, Y. Ren, and L. Zhang. Sharp bounds for the number of regions of maxout networks and vertices of Minkowski sums. *SIAM Journal on Applied Algebra and Geometry*, 6(4), 2022.

T. Motzkin. *Beitrage zur theorie der linearen Ungleichungen*. PhD thesis, University of Basel, 1936.

S. Mukhopadhyay, A. Roy, L. S. Kim, and S. Govil. A polynomial time algorithm for generating neural networks for pattern classification: Its stability properties and some test results. *Neural Computation*, 5(2):317–330, 1993.

V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying properties of binarized deep neural networks. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

O. Nelles, A. Fink, and R. Isermann. Local linear model trees (LOLIMOT) toolbox for nonlinear system identification. In *IFAC Symposium on System Identification (SYSID)*, 2000.

Y. E. Nesterov. A method of solving a convex programming problem with convergence rate $o\big(\frac{1}{k^2}\big)$. *Doklady Akademii Nauk*, 269:543–547, 1983.

M. Newton and A. Papachristodoulou. Exploiting sparsity for neural network verification. In *Learning for Dynamics and Control (L4DC)*, pages 715–727. PMLR, 2021.

Q. Nguyen, M. C. Mukkamala, and M. Hein. Neural networks should be wide enough to learn disconnected decision regions. In *International Conference on Machine Learning (ICML)*, 2018.

T. Nguyen and J. Huchette. Neural network verification as piecewise linear optimization: Formulations for the composition of staircase functions. *arXiv:2211.14706*, 2022.

R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations (ICLR)*, 2018.

OpenAI. Introducing chatgpt, 2022. URL `https://openai.com/blog/chatgpt`.

OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680*, 2019.

M. Padberg. Approximating separable nonlinear functions via mixed zero-one programs. *Operations Research Letters*, 27(1):1–5, 2000.

T. P. Papalexopoulos, C. Tjandraatmadja, R. Anderson, J. P. Vielma, and D. Belanger. Constrained discrete black-box optimization using mixed-integer programming. In *International Conference on Machine Learning (ICML)*, volume 162, pages 17295–17322, 2022.

D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. SpecAugment: A simple data augmentation method for automatic speech recognition. In *Interspeech*, 2019.

S. Park, C. Yun, J. Lee, and J. Shin. Minimum width for universal approximation. In *International Conference on Learning Representations (ICLR)*, 2021a.

Y. Park, S. Lee, G. Kim, and D. M. Blei. Unsupervised representation learning via neural activation coding. In *International Conference on Machine Learning (ICML)*, 2021b.

R. Pascanu, G. Montúfar, and Y. Bengio. On the number of response regions of deep feedforward networks with piecewise linear activations. In *International Conference on Learning Representations (ICLR)*, 2014.

J.-C. P. Patrick L. Combettes. Lipschitz certificates for layered network structures driven by averaged activation operators. *arXiv:1903.01014*, 2019.

G. Perakis and A. Tsiourvas. Optimizing objective functions from trained relu neural networks via sampling. *arXiv:2205.14189*, 2022.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

M. Phuong and C. H. Lampert. Functional vs. parametric equivalence of ReLU networks. In *International Conference on Learning Representations (ICLR)*, 2020.

M. Pilanci and T. Ergen. Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks. In *International Conference on Machine Learning (ICML)*, pages 7695–7705. PMLR, 2020.

S. Pokutta, C. Spiegel, and M. Zimmer. Deep neural network training with frank-wolfe. *arXiv:2010.07243*, 2020.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17, 1964.

L. Pulina and A. Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Computer Aided Verification (CAV)*, pages 243–257, 2010.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Dickstein. On the expressive power of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

A. Raghunathan, J. Steinhardt, and P. S. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *Neural Information Processing Systems (NeurIPS)*, 31, 2018.

P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. In *ICLR Workshop Track*, 2018.

R. Raman and I. Grossmann. Modelling and computational techniques for logic based integer programming. *Computers & Chemical Engineering*, 18(7):563–578, 1994.

A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv:2204.06125*, 2022.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

H. Robinson, A. Rasheed, and O. San. Dissecting deep neural networks. *arXiv:1910.03879*, 2019.

D. Rolnick and K. Kording. Reverse-engineering deep ReLU networks. In *International Conference on Machine Learning (ICML)*, 2020.

F. Rosenblatt. The Perceptron — a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

A. Rössig and M. Petkovic. Advances in verification of relu neural networks. *Journal of Global Optimization*, 81:109–152, 2021.

K. Roth. A primer on multi-neuron relaxation-based adversarial robustness certification. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021.

A. Roy, L. S. Kim, and S. Mukhopadhyay. A polynomial time algorithm for the construction and training of a class of multilayer perceptrons. *Neural Networks*, 6(4):535–545, 1993.

V. Rubies-Royo, R. Calandra, D. M. Stipanovic, and C. Tomlin. Fast neural network verification via shadow prices. *arXiv:1902.07247*, 2019.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

M. Ryu, Y. Chow, R. Anderson, C. Tjandraatmadja, and C. Boutilier. Caql: Continuous action q-learning. In *International Conference on Learning Representations (ICLR)*, 2020.

A. Sahiner, T. Ergen, J. M. Pauly, and M. Pilanci. Vector-output re{lu} neural network problems are copositive programs: Convex analysis of two layer networks and polynomial-time algorithms. In *International Conference on Learning Representations (ICLR)*, 2021.

H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *Neural Information Processing Systems (NeurIPS)*, 32, 2019.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

B. Sattelberg, R. Cavalieri, M. Kirby, C. Peterson, and R. Beveridge. Locally linear attributes of ReLU neural networks. *arXiv:2012.01940*, 2020.

B. Say, G. Wu, Y. Q. Zhou, and S. Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 750–756, 2017.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

J. Schumann, P. Gupta, and S. Nelson. On verification & validation of neural network based controllers. In *Engineering Applications of Neural Networks (EANN)*, 2003.

R. Schwan, C. N. Jones, and D. Kuhn. Stability verification of neural network controllers using mixed-integer programming. *arXiv:2206.13374*, 2022.

A. M. Schweidtmann and A. Mitsos. Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications*, 180(3):925–948, 2019.

A. M. Schweidtmann, J. M. Weber, C. Wende, L. Netze, and A. Mitsos. Obey validity limits of data-driven models through topological data analysis and one-class classification. *Optimization and Engineering*, 23(2):855–876, 2022.

I. Seck, G. Loosli, and S. Canu. Linear program powered attack. In *International Joint Conference on Neural Networks (IJCNN)*, 2021.

T. Serra. Enumerative branching with less repetition. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, pages 399–416. Springer, 2020.

T. Serra and J. Hooker. Compact representation of near-optimal integer programming solutions. *Mathematical Programming*, 182:199–232, 2020.

T. Serra and S. Ramalingam. Empirical bounds on linear regions of deep rectifier networks. In *AAAI Conference on Artificial Intelligence*, 2020.

T. Serra, C. Tjandraatmadja, and S. Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2018.

T. Serra, A. Kumar, and S. Ramalingam. Lossless compression of deep neural networks. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2020.

T. Serra, X. Yu, A. Kumar, and S. Ramalingam. Scaling up exact neural network compression by ReLU stability. In *Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

C. Shi, M. Emadikhiav, L. Lozano, and D. Bergman. Careful! training relevance is real. *arXiv:2201.04429*, 2022.

C. Sidrane, A. Maleki, A. Irfan, and M. J. Kochenderfer. Overt: An algorithm for safety verification of neural network control policies for nonlinear systems. *Journal of Machine Learning Research*, 23(117):1–45, 2022.

H. Sildir and E. Aydin. A mixed-integer linear programming based training and feature selection method for artificial neural networks using piece-wise linear approximations. *Chemical Engineering Science*, 249:117273, 2022.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. *Neural Information Processing Systems (NeurIPS)*, 31, 2018.

G. Singh, R. Ganvir, M. Püschel, and M. Vechev. Beyond the single neuron convex barrier for neural network certification. *Neural Information Processing Systems (NeurIPS)*, 32, 2019a.

G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages (POPL)*, 3:1–30, 2019b.

H. Singh, M. P. Kumar, P. Torr, and K. D. Dvijotham. Overcoming the convex barrier for simplex inputs. In *Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

J. E. Smith and R. L. Winkler. The optimizer's curse: Skepticism and post-decision surprise in decision analysis. *Management Science*, 52(3):311–322, 2006.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

C. A. Strong, H. Wu, A. Zeljić, K. D. Julian, G. Katz, C. Barrett, and M. J. Kochenderfer. Global optimization of objective functions represented by ReLU networks. *Machine Learning*, pages 1–28, 2021.

C. A. Strong, S. M. Katz, A. L. Corso, and M. J. Kochenderfer. ZoPE: a fast optimizer for ReLU networks with low-dimensional inputs. In *NASA Formal Methods: 14th International Symposium, (NFM)*, pages 299–317. Springer, 2022.

A. Sudjianto, W. Knauth, R. Singh, Z. Yang, and A. Zhang. Unwrapping the black box of deep ReLU networks: Interpretability, diagnostics, and simplification. *arXiv:2011.04041*, 2020.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, 2013.

I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems (NeurIPS)*, volume 27, 2014.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Y. Takai, A. Sannai, and M. Cordonnier. On the number of linear functions composing deep neural network: Towards a refined definition of neural networks complexity. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

Q. Tao, L. Li, X. Huang, X. Xi, S. Wang, and J. A. Suykens. Piecewise linear neural networks and deep learning. *Nature Reviews Methods Primers*, 2, 2022.

M. Telgarsky. Representation benefits of deep feedforward networks. *arXiv:1509.08101*, 2015.

T. Thorbjarnarson and N. Yorke-Smith. On training neural networks with mixed integer programming. *arXiv:2009.03825*, 2020.

T. Thorbjarnarson and N. Yorke-Smith. Optimal training of integer-valued neural networks with mixed integer programming. *PLOS One*, 18(2):e0261029, 2023.

S. Tiwari and G. Konidaris. Effects of data geometry in early deep learning. In *Neural Information Processing Systems (NeurIPS)*, 2022.

C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. K. Patel, and J. P. Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Neural Information Processing Systems (NeurIPS)*, 33:21675–21686, 2020.

V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.

M. Trimmel, H. Petzka, and C. Sminchisescu. TropEx: An algorithm for extracting linear terms in deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2021.

C. Tsay and M. Baldea. 110th anniversary: using data to bridge the time and length scales of process systems. *Industrial & Engineering Chemistry Research*, 58(36):16696–16708, 2019.

C. Tsay, J. Kronqvist, A. Thebelt, and R. Misener. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. In *Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

H. Tseran and G. Montúfar. On the expected complexity of maxout networks. In *Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

M. Unser. A representer theorem for deep neural networks. *Journal of Machine Learning Research*, 20:1–30, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Neural Information Processing Systems (NeurIPS)*, 2017.

J. P. Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1):3–57, 2015.

J. P. Vielma. Small and strong formulations for unions of convex sets from the cayley embedding. *Mathematical Programming*, 177(1-2):21–53, 2019.

J. P. Vielma, S. Ahmed, and G. Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Operations Research*, 58(2):303–315, 2010.

M. J. Villani and N. Schoots. Any deep ReLU network is shallow. *arXiv:2306.11827*, 2023.

J. A. Vincent and M. Schwager. Reachable polyhedral marching (RPM): A safety verification algorithm for robotic systems with deep neural network components. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. StarCraft II: A new challenge for reinforcement learning. *arXiv:1708.04782*, 2017.

A. Virmaux and K. Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

M. Volpp, L. P. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *International Conference on Learning Representations (ICLR)*, 2020.

K. Wang, L. Lozano, D. Bergman, and C. Cardonha. A two-stage exact algorithm for optimization of neural network ensemble. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2021.

K. Wang, L. Lozano, C. Cardonha, and D. Bergman. Optimizing over an ensemble of trained neural networks. *INFORMS Journal on Computing*, 2023.

S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. *Neural Information Processing Systems (NeurIPS)*, 31, 2018a.

S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1599–1614, 2018b.

Y. Wang. Estimation and comparison of linear regions for relu networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.

J. Weng, N. Ahuja, and T. Huang. Cresceptron: a self-organizing neural network which grows adaptively. In *International Joint Conference on Neural Networks (IJCNN)*, 1992.

L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning (ICML)*, pages 5276–5285, 2018.

P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

M. Wicker, L. Laurenti, A. Patane, and M. Kwiatkowska. Probabilistic safety for Bayesian neural networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 1198–1207, 2020.

M. Wicker, J. Heo, L. Costabello, and A. Weller. Robust explanation constraints for neural networks. *arXiv:2212.08507*, 2022.

M. E. Wilhelm, C. Wang, and M. D. Stuber. Convex and concave envelopes of artificial neural network activation functions for deterministic global optimization. *Journal of Global Optimization*, pages 1–26, 2022.

E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, pages 5286–5295, 2018.

E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter. Scaling provable adversarial defenses. *Neural Information Processing Systems (NeurIPS)*, 31, 2018.

S. J. Wright. Optimization algorithms for data analysis. *The Mathematics of Data*, 25:49, 2018.

G. Wu, B. Say, and S. Sanner. Scalable planning with deep neural network learned transition models. *Journal of Artificial Intelligence Research*, 68:571–606, 2020.

H. Wu, A. Zeljić, G. Katz, and C. Barrett. Efficient neural network analysis with sum-of-infeasibilities. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 143–163. Springer, 2022.

W. Xiang, H.-D. Tran, and T. T. Johnson. Reachable set computation and safety verification for neural networks with ReLU activations. *arXiv:1712.08163*, 2017.

K. Xiao, V. Tjeng, N. Shafiullah, and A. Madry. Training for faster adversarial robustness verification via inducing ReLU stability. *International Conference on Learning Representations (ICLR)*, 2019.

J. Xie, Z. Shen, C. Zhang, B. Wang, and H. Qian. Efficient projection-free online methods with stochastic recursive gradient. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 6446–6453, 2020a.

Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. Self-training with noisy student improves ImageNet classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020b.

Y. Xie, G. Chen, and Q. Li. A general computational framework to measure the expressiveness of complex networks using a tighter upper bound of linear regions. *arXiv:2012.04428*, 2020c.

H. Xiong, L. Huang, M. Yu, L. Liu, F. Zhu, and L. Shao. On the number of linear regions of convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 2020.

S. Xu, J. Vaughan, J. Chen, A. Zhang, and A. Sudjianto. Traversing the local polytopes of ReLU neural networks. In *AAAI Workshop AdvML*, 2022.

D. Yang, P. Balaprakash, and S. Leyffer. Modeling design and control problems involving neural network surrogates. *Computational Optimization and Applications*, pages 1–42, 2022.

X. Yang, H.-D. Tran, W. Xiang, and T. Johnson. Reachability analysis for feed-forward neural networks using face lattices. *arXiv:2003.01226*, 2020.

X. Yang, T. Yamaguchi, H.-D. Tran, B. Hoxha, T. T. Johnson, and D. Prokhorov. Reachability analysis of convolutional neural networks. *arXiv:2106.12074*, 2021.

D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94, 2017.

R. R. Zakrzewski. Verification of a trained neural network accuracy. In *International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 1657–1662. IEEE, 2001.

T. Zaslavsky. *Facing Up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes.* American Mathematical Society, 1975.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning.* 2023. `https://d2l.ai`.

H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. *Neural Information Processing Systems (NeurIPS)*, 31, 2018a.

H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, and J. Z. Kolter. General cutting planes for bound-propagation-based neural network verification. In *Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

L. Zhang, G. Naitzat, and L.-H. Lim. Tropical geometry of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2018b.

R. Zhang. On the tightness of semidefinite relaxations for certifying robustness to adversarial examples. *Neural Information Processing Systems (NeurIPS)*, 33:3808–3820, 2020.

X. Zhang and D. Wu. Empirical studies on the properties of linear regions in deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

S. Zhao, C. Tsay, and J. Kronqvist. Model-based feature selection for neural networks: A mixed-integer programming approach. *arXiv:2302.10344*, 2023.

S. Zhou and A. P. Schoellig. An analysis of the expressiveness of deep neural network architectures based on their Lipschitz constants. *arXiv:1912.11511*, 2019.

R. Zhu, B. Lin, and H. Tang. Bounding the number of linear regions in local area for neural networks with ReLU activations. *arXiv:2007.06803*, 2020.

D. Zou, R. Balan, and M. Singh. On Lipschitz bounds of general convolutional neural networks. *IEEE Transactions on Information Theory*, 66(3):1738–1759, 2019.