TROPEX: AN ALGORITHM FOR EXTRACTING LINEAR TERMS IN DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural networks with rectified linear (ReLU) activations are piecewise linear functions, partitioning the input space into an astronomically high number of linear regions. Attempts have been made to obtain insight into the nature of networks by counting and probing linear regions. However, due to their high computational complexity, previous algorithms have only been applied to study the linear regions of small networks, almost exclusively using the simple MNIST data set for training, thereby limiting the conclusiveness of the observations. We propose TropEx, a tropical algebra-inspired algorithm for extracting linear terms based on data, enabling an analysis of their coefficients and allowing for a wide range of new experiments. In particular, we examine qualitative differences in the features convolutional neural networks (CNNs) and fully-connected networks (FCNs) extract from images: CNNs seem to base their decisions on object shapes, whereas FCNs seem to heavily rely on the general colour distribution of an image. Moreover, comparing findings across different data sets lets us conclude that in contrast to previous practice, results on linear regions from MNIST may not generalise to more complex data sets.

1 INTRODUCTION

Many of the most widely used neural network architectures, including VGG (Simonyan & Zisserman, 2014), GoogLeNet (Szegedy et al., 2015) and ResNet (He et al., 2016), make use of rectified linear activations (ReLU, (Hahnloser et al., 2000; Glorot et al., 2011), i.e., $\sigma(x) = \max\{x, 0\}$)) and are therefore piecewise linear functions. Despite these functions being apparently simple, there is a lack of theoretical understanding of the factors that contribute to the success of these architectures. Previous attempts of understanding piecewise linear network functions have focused on estimating the number of **linear terms**, which are the linear pieces that constitute the network function. A **linear region** is being defined as a maximally connected subset of the domain on which the network function is linear. Since computing the exact number of linear regions is intractable, work has focused on obtaining upper and lower bounds for this number(Arora et al., 2016; Serra et al., 2018; Pascanu et al., 2013; Raghu et al., 2017; Montufar et al., 2014; Montúfar, 2017; Zhang et al., 2018). To our knowledge, the currently best upper and lower bounds were calculated by Serra et al. (2018) and Raghu et al. (2017) show these bounds are asymptotically tight.

All mentioned papers share the intuition that the number of linear regions of neural networks measure their expressivity. Since the bounds grow linearly in width and exponentially in depth, deep networks are interpreted to have greater representational power. However, these bounds are staggeringly high: The upper bound on the number of linear regions in (Serra et al., 2018) exceeds 10^{300} even for the smallest networks we experimented on. (There are approximately 10^{80} atoms in the universe.) For slightly larger networks, the upper bound exceeds 10^{17000} whereas the lower bound exceeds 10^{83} linear regions. This raises the question how many linear regions are actually necessary to conserve the performance attained on both training and unseen test data. For instance, it might of course be that the network divides its domain into linear regions that do not overlap with the data distribution. Further, the number of training samples is generally much smaller than the estimated number of linear regions, which means that lots of linear regions contain no evidence of data.

There are indications that a high number of linear regions is not required for good network performance. Frankle & Carbin (2019) point out that smaller networks perform similarly well as large

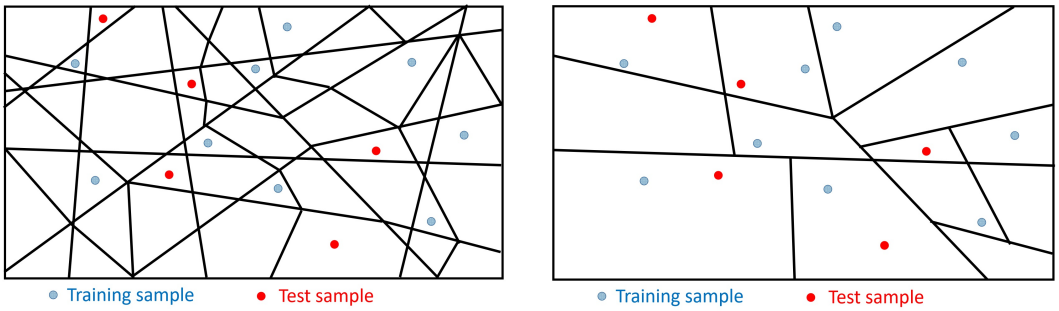

Figure 1: A ReLU network function before *(left)* and after *(right)* extraction. *Left*: Hyperplanes separate the input space into linear regions. Most of them do not contain any data points. Each data point occupies its own linear region. *Right:* After extraction, the function remains unchanged on the linear regions of training samples. Most test samples now fall into a region of a training sample.

neural networks, when a suitable initialization of the smaller network can be found from training the larger one. Hence, the expressivity of the large network is helpful to explore the parameter space, but the small, less expressive network is sufficient to achieve high accuracy. Lee et al. (2019) and Croce et al. (2018) modify the training loss to encourage network functions to have larger linear regions with the goal of robustness to adversarial attacks. Hanin & Rolnick (2019b;a) argue that in practice there are less linear regions than expected from the bounds and empirically investigate this for 2D-cross sections for the MNIST dataset. All these observations question the explanatory power of astronomically high bounds for the number of linear regions.

We propose **TropEx**, a tropical algebra-based algorithm extracting linear terms of the network function $\mathcal{N}$ (Figure 1) using a data set $\mathcal{X}$. TropEx outputs an extracted network function $\mathcal{N}^{(\mathcal{X})}$ containing only the linear terms corresponding to regions on which data lies. As a result, $\mathcal{N}$ and $\mathcal{N}^{(\mathcal{X})}$ agree on neighbourhoods of all data points. Our goal is to create a starting point for the study of generalization from a new viewpoint, i.e., the perspective of linear regions.

Our contributions are as follows:
- A computational framework representing tropical functions (Definition B.4) as matrices.
- TropEx, an algorithm to extract linear terms from piecewise linear network functions.
- An open source implementation of the algorithm. A link to the code will be added upon publication.
- Application of TropEx to fully connected (FCN) and convolutional networks (CNN) reveals that (i) consistently all training and test samples fall into different linear regions ; (ii) FCNs and CNNs differ in how they use linear regions free of training data, where CNNs, in contrast to FCNs, tend to learn linear functions that are both diverse and meaningful on unseen data.

2 Background and Overview

It was recently shown by Charisopoulos & Maragos (2018); Zhang et al. (2018) that ReLU neural network functions are the same as *tropical rational maps*. Tropical rational maps are exactly those functions where each entry in the output vector can be written as a difference of maxima

$$\mathcal{N}_i(\mathbf{x}) = \max\{a_{i1}^+(\mathbf{x}), \dots, a_{in}^+(\mathbf{x})\} - \max\{a_{i1}^-(\mathbf{x}), \dots, a_{im}^-(\mathbf{x})\}, \tag{1}$$

where each $a_{ij}^+, a_{ij}^- : \mathbb{R}^d \to \mathbb{R}$ is an affine function with only positive coefficients, taking the form $\mathbf{x} \mapsto \sum_k w_k x_k + w_0$ with all $w_i \in \mathbb{R}_{\geq 0}$. Since the number of terms in (1) dwarfs the number of atoms in the universe, it is impossible to obtain this expression in practice. Therefore, we only extract those terms that correspond to linear regions of data points. For a fixed data point $\mathbf{x} \in \mathcal{X}$, the maximum of the network outputs can be written as $\max_i \mathcal{N}_i(\mathbf{x}) = a_{\mathbf{x}}^+(\mathbf{x}) - a_{\mathbf{x}}^-(\mathbf{x})$, where $a_{\mathbf{x}}^+, a_{\mathbf{x}}^-$ are the affine functions such that $a_{\mathbf{x}}^+(\mathbf{x}) \geq a_{ij}^+(\mathbf{x}), a_{\mathbf{x}}^-(\mathbf{x}) \geq a_{ij}^-(\mathbf{x})$ for all i, j. TropEx extracts $a_{\mathbf{x}}^+$ and $a_{\mathbf{x}}^-$. The extracted terms can be used to construct a reduced tropical map $\mathcal{N}^{(\mathcal{X})}$, given by

$$\mathcal{N}_i^{(\mathcal{X})}(\mathbf{x}) = \max\{a_{\mathbf{x}_{i_1}}^+(\mathbf{x}), \dots, a_{\mathbf{x}_{i_{D_i}}}^+(\mathbf{x})\} - \max\{a_{\mathbf{x}_{i_1}}^-(\mathbf{x}), \dots, a_{\mathbf{x}_{i_{D_i}}}^-(\mathbf{x})\}, \tag{2}$$

where there are D_i data points $\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_{D_i}}$ given label i by the original network. Being a tropical rational map, the function $\mathcal{N}^{(\mathcal{X})}$ is again a ReLU neural network function by Zhang et al. (2018).

The maximal component of the extracted network $\mathcal{N}^{(\mathcal{X})}$ agrees with the maximal component of the original network $\mathcal{N}$ (and hence the assigned label) in the neighbourhood of any data point $\mathbf{x} \in \mathcal{X}$.

3 METHOD

3.1 MATRIX REPRESENTATION OF TROPICAL RATIONAL MAPS

If one were to represent tropical rational maps symbolically on a computer, computations would be too slow. Therefore, we present tropical rational maps as multi-dimensional arrays.

Definition 3.1. Given an affine function $a : \mathbb{R}^{d_0} \to \mathbb{R}; \mathbf{x} \mapsto \sum_k w_k x_k + w_0$, we will call the vector $(w_0, w_1, \dots, w_{d_0})$ its *coefficient vector*, the scalar w_0 its *constant part* and the vector $(w_1, \dots, w_{d_0})$ its *variable part*.

We can represent functions $\mathcal{N}_i : \mathbb{R}^{d_0} \to \mathbb{R}$ as in equation (1) in the following way: Let the rows of the matrix $\mathbf{A}_i^+ \in \mathbb{R}^{n \times d_0}$ and the vector $\mathbf{a}_i^+ \in \mathbb{R}^{n \times 1}$ be the variable and the constant parts of the affine functions a_{ij}^+, respectively. (Analogously for $\mathbf{A}_i^-$ and $\mathbf{a}_i^-$.) We can then define $(\mathbf{A}_i^+, \mathbf{a}_i^+)(\mathbf{x}) = \max\{\mathbf{A}_i^+ \mathbf{x} + \mathbf{a}_i^+\}$, where the maximum is taken over the rows of the resulting column vector. If we define the formal quotient[1] of matrix-vector pairs by $(\mathbf{A}_i^+, \mathbf{a}_i^+)/(\mathbf{A}_i^-, \mathbf{a}_i^-)(\mathbf{x}) = \max\{\mathbf{A}_i^+ \mathbf{x} + \mathbf{a}_i^+\} - \max\{\mathbf{A}_i^- \mathbf{x} + \mathbf{a}_i^-\}$, then $\mathcal{N}_i(\mathbf{x}) = (\mathbf{A}_i^+, \mathbf{a}_i^+)/(\mathbf{A}_i^-, \mathbf{a}_i^-)(\mathbf{x})$, giving us a matrix-representation of the function $\mathcal{N}_i$. The entire network function can then be represented by a list $\big((\mathbf{A}_i^+, \mathbf{a}_i^+)/(\mathbf{A}_i^-, \mathbf{a}_i^-)\big)_{1 \le i \le s}$.

The advantage of the proposed matrix representation of tropical rational maps are natural operations performing calculations that arise for (concatenations of) layers of neural networks (see supplements). A dense layer $\ell : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ without ReLU activation is represented as a list $\big((\mathbf{A}_i^+, \mathbf{a}_i^+)/(\mathbf{A}_i^-, \mathbf{a}_i^-)\big)_{1 \le i \le d_2}$. Denoting by $\mathbf{W}^{\text{pos}}$ and $\mathbf{W}^{\text{neg}}$ the positive and negative part of a matrix $\mathbf{W}$, respectively, i.e. $w_{ij}^{\text{pos}} = \max\{w_{ij}, 0\}$ and $w_{ij}^{\text{neg}} = \max\{-w_{ij}, 0\}$, the matrix representation of a single neuron $n_i(\mathbf{x}) = \max\{\mathbf{w} \cdot \mathbf{x} + b, 0\}$ is given by

$$\mathbf{A}_i^+ = \begin{pmatrix} \mathbf{w}^{\text{pos}} \\ \mathbf{w}^{\text{neg}} \end{pmatrix}, \ \mathbf{a}_i^+ = \begin{pmatrix} b^{\text{pos}} \\ b^{\text{neg}} \end{pmatrix}; \quad \mathbf{A}_i^- = \mathbf{w}^{\text{neg}}, \ \mathbf{a}_i^- = b^{\text{neg}}.$$

3.2 EXTRACTING LINEAR TERMS OF A CLASSIFICATION NETWORK

We now consider a classification neural network $\mathcal{N}$ with s labels. We now show that we can represent a network $\mathcal{N}$ constant matrix-pair $(\mathbf{A}^-, \mathbf{a}^-)$ in the denominator.

Lemma 3.2. *Let $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}^s$ be the function of a ReLU neural network for classification with s output neurons. Then there are affine functions a_{ij}^+, a_j^- such that*

$$\mathcal{N}(\mathbf{x}) = \begin{pmatrix} \max\{a_{11}^+(\mathbf{x}), \dots, a_{1n_1}^+(\mathbf{x})\} \\ \vdots \\ \max\{a_{1s}^+(\mathbf{x}), \dots, a_{1n_s}^+(\mathbf{x})\} \end{pmatrix} - \max\{a_1^-(\mathbf{x}), \dots, a_m^-(\mathbf{x})\}, \tag{3}$$

where the maxima of the $a_i^-(\mathbf{x})$ on the right is subtracted from each entry of the vector on the left.[2]

In terms of our matrices, $\mathcal{N}$ can be represented by a list $\big((\mathbf{A}_i^+, \mathbf{a}_i^+)/(\mathbf{A}^-, \mathbf{a}^-)\big)_{1 \le i \le s}$ of matrix-pairs. The proof is given in Section C.2 of the supplementary material.

How to perform classification with an expression as in (3)**.** The label that a network assigns a data point is the argmax of the entries of $\mathcal{N}(\mathbf{x})$. Subtracting the expression on the right in (3) from each

[1] This is in line with the tropical algebra notation: a *tropical quotient* is the same as a usual difference.

[2] This implies that every classification ReLU neural network can be represented by a convex function.

Algorithm 3.1 TropEx: Extracting Linear Terms of a Neural Network

Inputs:
Neural Network $\mathcal{N}$
Data set $\mathcal{X} = \{(\mathbf{x}_{i_k}, i)\}_{1 \leq i \leq s}$, with D_i points of label i
Output:
Extracted Network Function $\mathcal{N}^{(\mathcal{X})} = \left((\mathbf{A}_i^+, \mathbf{a}_i^+)/(\mathbf{A}^-, \mathbf{a}^-)\right)_{1 \leq i \leq s}$

$\mathbf{W}, \mathbf{b} \leftarrow$ weight matrix and bias vector of last layer ℓ
$\mathbf{C}^-, \mathbf{c}^- \leftarrow$ column sums of $\mathbf{W}^{\text{neg}}, \mathbf{b}^{\text{neg}}$
$\mathbf{C}^+ \leftarrow \mathbf{W} + \mathbf{C}^-, \mathbf{c}^+ \leftarrow \mathbf{b} + \mathbf{c}^-$
for $i = 1$ to s **do**
    $\mathbf{A}_i^+ \leftarrow \text{rep}(\mathbf{C}_{i\bullet}^+, D_i),\ \mathbf{a}_i^+ \leftarrow \text{rep}(\mathbf{c}_i^+, D_i)$ ▷ Repetition D_i times along the rows.
$\mathbf{A}^- \leftarrow \text{rep}(\mathbf{C}^-, D),\ \mathbf{a}^- \leftarrow \text{rep}(\mathbf{c}^-, D)$ ▷ $D =$ total no of data points
$\mathbf{A}_{\text{max}} \leftarrow$ maxima of the columns of all $\mathbf{A}_i^+$ and $\mathbf{A}^-$ stacked
for last layer ℓ in $\mathcal{N}$ not yet used **do**
    $(\mathcal{N}^{(\mathcal{X})}, \mathbf{A}_{\text{max}}) \leftarrow \text{merge}_\ell\ (\mathcal{N}^{(\mathcal{X})}, \mathbf{A}_{\text{max}})$ according to Table 1

Type	Operation	Type	Operation
BNorm	$\gamma, \beta, \mu, \sigma, \epsilon \leftarrow$ Batchnorm parameters $s \leftarrow \gamma/\sqrt{\sigma^2 + \epsilon}, t \leftarrow -\mu \cdot s + \beta$ $\mathbf{A} \leftarrow s\mathbf{A}; \mathbf{a} \leftarrow \mathbf{a} + \mathbf{A}t$ $\mathbf{A}_{\text{max}} \leftarrow \lvert s \rvert \mathbf{A}_{\text{max}}$	Maxpool	$\mathbf{A}, \mathbf{A}_{\text{max}} \leftarrow$ repeat to input shape of ℓ $\mathbf{A}_k \leftarrow$ set 0 according to activations of $\overline{\ell}(\mathbf{x}_k)$
Conv	$F, \mathbf{b} \leftarrow$ filter, bias of ℓ $K \leftarrow \text{ConvTrans}(\mathbf{A}_{\text{max}}, F^{\text{neg}})$ $\mathbf{A}_{\text{max}} \leftarrow K + \text{ConvTrans}(\mathbf{A}_{\text{max}}, F^{\text{pos}})$ $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{A}\mathbf{b}$ $\mathbf{A} \leftarrow K + \text{ConvTrans}(\mathbf{A}, F)$	Dense	$\mathbf{W}, \mathbf{b} \leftarrow$ weights, bias of ℓ $K \leftarrow \mathbf{A}_{\text{max}}\mathbf{W}^{\text{neg}}$ $\mathbf{A}_{\text{max}} \leftarrow K + \mathbf{A}_{\text{max}}\mathbf{W}^{\text{pos}}$ $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{A}\mathbf{b}$ $\mathbf{A} \leftarrow K + \mathbf{A}\mathbf{W}$
Flatten	$\mathbf{A}, \mathbf{A}_{\text{max}} \leftarrow$ reshape to input shape of ℓ	L-ReLU	$\alpha \leftarrow$ Leaky ReLU parameter $\mathbf{A}_{kj} \leftarrow \alpha \cdot \mathbf{A}_{kj}$ if $\overline{\ell}(\mathbf{x}_k)_j \leq 0$
ReLU	$\mathbf{A}_{kj} \leftarrow 0$ if $\overline{\ell}(\mathbf{x}_k)_j = 0$		

Table 1: Merge operations $(\mathcal{N}^{(\mathcal{X})}, \mathbf{A}_{\text{max}}) \mapsto (\mathcal{N}^{(\mathcal{X})}, \mathbf{A}_{\text{max}})$, according to type of layer ℓ. $\mathbf{A}_k$ denotes the slice of $\mathbf{A}$ corresponding to data point k. $\overline{\ell}$ denotes all of the network up to and including ℓ. Read expressions like $\mathbf{A} \leftarrow K + \mathbf{a}\mathbf{W}$ as $\mathbf{A}_i^+ \leftarrow K + \mathbf{A}_i^+\mathbf{W}; \mathbf{A}^- \leftarrow K + \mathbf{A}^-\mathbf{W}$ for all i.

entry, leaves the argmax unchanged, so we can discard this expression for classification and only use the vector on the left. In terms of our matrices, the label is given by

$$\text{argmax}_{1 \leq i \leq s} \max\{\mathbf{A}_i\mathbf{x} + \mathbf{a}_i\},$$

where the maximum is taken over the rows of the resulting vector.

How to get the extracted function $\mathcal{N}^{(\mathcal{X})}$ from the full network $\mathcal{N}$. For each data point $\mathbf{x}_k$ of label i, TropEx extracts the affine functions a_{ij}^+ and a_l^- from the network representation in (3) such that $a_{ij}^+(\mathbf{x}_k) \geq a_{\tilde{i}\tilde{j}}^+(\mathbf{x}_k), a_l^-(\mathbf{x}_k) \geq a_{\tilde{l}}(\mathbf{x}_k)$ for all $\tilde{i}, \tilde{j}, \tilde{l}$. We start with the last layer of the network and inductively merge new layers into the existing matrix pairs. The merge operations depend on the type of the layer and are shown in Table 1. Putting things together gives Algorithm 3.1, which has the extracted function $\mathcal{N}^{(\mathcal{X})}$ as its output. The run-time and storage complexities per data point correspond to 3 forward passes through the network. Theorem 3.3 states that TropEx indeed results in a selection of linear terms based on a dataset of points and that the extracted tropical function agrees with the network on neighbourhoods of all training samples. Its proof and the complete derivation of the algorithm are in the appendix, where we develop a framework that allows tropical calculations on matrices that correspond to manipulations of tropical functions they represent. For illustrative purposes, we also present a worked-out example of applying TropEx to a toy neural network there.

Theorem 3.3. *Let $\mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_s) : \mathbb{R}^d \to \mathbb{R}^s$ be the function of a ReLU neural network for classification into s classes. Let $\mathcal{N}^{(\mathcal{X})}$ be the network obtained from Algorithm 3.1, applied to $\mathcal{N}$ using a data set $\mathcal{X} = \{(\mathbf{x}_{i_k}, i)\}_{1 \leq i \leq s}$ of D_i data points $\mathbf{x}_{i_k}$ given label i by $\mathcal{N}$. (There are D points in total). Then, (1) for all labels i, $\mathcal{N}_i^{(\mathcal{X})}(\mathbf{x}) = \max\{a_{i1}^+(\mathbf{x}), \ldots, a_{iD_i}^+(\mathbf{x})\} - \max\{a_1^-(\mathbf{x}), \ldots, a_D^-(\mathbf{x})\}$,*

where the a_{ij}^+ and a_j^- are extracted from a representation of $\mathcal{N}_i$ as in equation (3)*; and (2) every data point $\mathbf{x}_k$ has a neighbourhood U_k on which the maximum of the extracted function agrees with the maximal network output:*

$$\max_{1\leq i\leq s} \mathcal{N}_i^{(\mathcal{X})}(\mathbf{x}) = \max_{1\leq i\leq s} \mathcal{N}_i(\mathbf{x}) \quad \textit{for all } \mathbf{x} \in U_k.$$

In particular, both $\mathcal{N}^{(\mathcal{X})}$ and $\mathcal{N}$ classify all points in U_k with the same label.

4 EXPERIMENTS

Setup We train neural networks on MNIST (LeCun et al., 2010) and CIFAR10 (Krizhevsky, 2009). After training, we use training data points $\mathbf{x}^{(tr)}$ to extract linear terms $a^+_{\mathbf{x}^{(tr)}}$ and $a^-_{\mathbf{x}^{(tr)}}$ and construct an extracted network $\mathcal{N}^{(\mathcal{X})}$ as in equation (2). For some experiments, we also extract linear terms $a^+_{\mathbf{x}^{(te)}}$ and $a^-_{\mathbf{x}^{(te)}}$ corresponding to test data points $\mathbf{x}^{(te)}$. Regarding the architecture, we use fully-connected networks, AllCNN-C from Springenberg et al. (2014) and variations of VGG-B from Simonyan & Zisserman (2014). Section D in the appendix summarizes all architectures we used in our experiments. It is not our goal to train networks to state of the art-performance, but rather to compare variations of simple networks which are composed of the layers shown in Table 1. All layers have ReLU activations except for the last layer where we apply a softmax output function into the ten respective classes.[3] We train five networks of each architecture to ensure the consistency of our results. Further details on the training setup can be found in the appendix.

Counting of Linear Regions At first, we investigate how training and test samples are distributed over the linear regions of the neural networks. For each data point $\mathbf{x}$, let $a_{\mathbf{x}} = a^+_{\mathbf{x}} - a^-_{\mathbf{x}}$ be network function corresponding to the linear region on which $\mathbf{x}$ lies. Since $a_{\mathbf{x}} \neq a_{\mathbf{x}'}$ for all training and test points $\mathbf{x}, \mathbf{x}'$, we see that all points lie in different linear regions.[4] This is not a result of overfitting during training: All data points also occupy different regions when we count the number of linear regions after 1, 3, 5, 10, 15, 20, 30, 40 and 50 epochs of training on CIFAR10, and from epochs 1 to 20 on MNIST. Therefore we conclude that generalization capabilities of neural networks cannot be explained by test samples falling into the same linear region as training samples (or, in other words, by test samples inducing the same activation pattern as training samples).

Accuracy of the Extracted Functions As predicted by Theorem 3.3, the maximum of the extracted function $\mathcal{N}^{(\mathcal{X})}$ agrees with the maximum of the original network on all training points for each of our networks. In particular, network and extracted function assign the same label to all training points. To investigate how well the coefficients of these linear regions generalize to unseen data, we compare test accuracy of network and extracted function in Table 2. We see a consistent difference between CNNs and FCNs across both data sets and all architectures: There is a drastic drop in the test accuracy of the CNNs, as opposed to a relatively small drop in the accuracy of the FCNs. Interestingly, for MNIST, the extracted tropical function has almost the same test accuracy as the original network. This is surprising as all known bounds on the number of linear regions of the original network suggest numbers of the order of 10^{80} up to over 10^{17000} from which we only observe 60.000 after reduction. Hence, for fully connected networks on a simple task, the coefficients used on training data generalize well to test data, but for complex data, the learned coefficients generalize worse. Moreover, the results reveal a difference between FCNs and CNNs that we further investigate.

Transformation Process Figure 2, left shows how test accuracy and agreement with the original network develop during the transformation from full network $\mathcal{N}$ to its extracted form $\mathcal{N}^{(\mathcal{X})}$. Starting with the last layer, TropEx iteratively merges layers into an extracted network function. The x-axis shows which layers have been merged to an extracted tropical function at each step. The graph shows accuracy values for passing test samples through the original network until the layer representing the input to the extracted tropical function and then applying this extracted function. At the right end of the plots all layers have been merged to the tropical function $\mathcal{N}^{(\mathcal{X})}$. There is a clear difference between fully-connected networks and CNNs, which is consistent over all networks (Figure 2, right).

[3] We also experimented with replacing ReLU with Leaky ReLU. Our observations are in line with what we describe for ReLU. More details can be found in the appendix, Section E.3.

[4] Except for the small 2-layer architecture, where the data points lie on 59,850 regions instead of 60,000.

	Name	$\mathcal{N}$ vs $\mathcal{N}^{(\mathcal{X})}$	$\mathcal{N}$ vs true	$\mathcal{N}^{(\mathcal{X})}$ vs true
MNIST	**FC4**	$97.7_{\pm 0.1}$	$98.1_{\pm 0.1}$	$97.7_{\pm 0.1}$
	FCN6	$97.7_{\pm 0.3}$	$98.1_{\pm 0.2}$	$97.5_{\pm 0.2}$
	Conv	$95.7_{\pm 0.3}$	$99.2_{\pm 0.1}$	$95.8_{\pm 0.2}$
CIFAR	**FC2**	$74.4_{\pm 3.7}$	$49.1_{\pm 0.8}$	$46.0_{\pm 0.4}$
	FC4	$52.3_{\pm 1.5}$	$51.8_{\pm 0.7}$	$40.4_{\pm 0.8}$
	FC8	$47.3_{\pm 4.4}$	$49.0_{\pm 0.4}$	$35.5_{\pm 1.9}$
	Wide	$57.8_{\pm 4.6}$	$46.7_{\pm 1.0}$	$38.6_{\pm 1.7}$
	AllCNN	$32.6_{\pm 0.1}$	$86.9_{\pm 0.1}$	$32.4_{\pm 0.1}$
	BaNorm	$27.6_{\pm 0.1}$	$70.2_{\pm 0.1}$	$27.7_{\pm 0.1}$
	Conv	$29.4_{\pm 0.9}$	$71.8_{\pm 0.7}$	$28.6_{\pm 1.2}$
	Deep	$37.7_{\pm 0.6}$	$71.2_{\pm 0.5}$	$38.0_{\pm 0.8}$
	Narrow	$27.5_{\pm 3.1}$	$63.1_{\pm 0.6}$	$26.5_{\pm 2.9}$
	NoMax	$29.9_{\pm 2.8}$	$63.2_{\pm 0.9}$	$28.4_{\pm 3.4}$

Table 2: Results on test data after extraction of linear terms. Values denote mean value and standard deviation over 5 networks in percent. Column 1: Agreement of original network $\mathcal{N}$ with extracted function $\mathcal{N}^{(\mathcal{X})}$. Columns 2&3: Multi-class accuracy for $\mathcal{N}$ and $\mathcal{N}^{(\mathcal{X})}$. FCN$i$ denotes a fully-connected network with i layers. AllCNN is derived from architecture C from (Springenberg et al., 2014). All other convolutional architectures are variations of VGG network B from (Simonyan & Zisserman, 2014). The detailed architectures are described in the appendix.

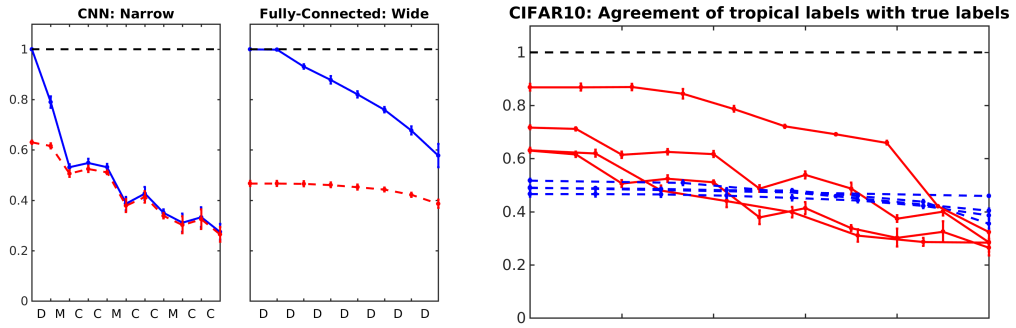

Figure 2: Left: Comparison of average test accuracy (dotted red) and agreement with labels assigned by the original network (blue) for networks *Narrow* and *Wide* while being transformed into tropical functions. The letters D,M,C denote dense, maxpooling and convolutional layers, respectively. Right: Average test accuracy for all CIFAR networks while being transformed into a tropical function. Fully-connected: dotted blue lines, CNNs: full red lines. The curves show a significant difference between the network types.

The fully connected network *Wide* and the convolutional network *Narrow* have exactly the same number of nodes after each parameter layer, but the drop in test accuracy for *Narrow* is almost 5 times the drop for *Wide* (36.6% vs 8.1%). Starting with a much superior performance, after reduction of linear terms the convolutional network is outperformed by the fully connected network. Since *Narrow* has only few connections between its nodes and *Wide* is fully-connected, it would be reasonable to assume that the number of linear regions of *Wide* is much greater than the number of linear regions of *Narrow*. Hence, it would be expected that an extraction of a fixed number of linear terms resulted in a smaller change of results for *Narrow* and the results were be reversed. This contradicts our intuition about how CNNs and FCNs work from a perspective of linear regions.

Network Training We compare the performance of extracted functions $\mathcal{N}^{(\mathcal{X})}$ during training of the network $\mathcal{N}$. Figure 3 displays the test accuracy (mean and standard deviation over 5 networks) of the extracted function and the agreement of label assignments of extracted form with the original network function for *Narrow* (CNN) and *Wide* (fully connected) for several epochs. The difference between fully connected and convolutional networks is here even more striking. For the CNN, the agreement between extracted tropical function and original network function falls rapidly after only

one epoch and only slightly reduces from there. For the fully connected network, the agreement decreases slowly over the entire 50 training epochs and it never reaches a value as low as the CNN after its first epoch.

The extracted network functions all share the number of linear terms, hence their difference in performance must be explained by the coefficient values. With this in mind, we attempt an interpretation of these results. We hypothesize that the structure on the weight space imposed by weight sharing in the CNN results in the incapability to use dominant linear coefficients for correct label assignments. Instead, the CNN uses the imposed structure on the weights for generalization (such as re-occurrence of filters at many positions). By extracting linear terms we remove some structure, which strongly changes the outcome on test data as the coefficients in linear regions alone are limited in meaning. In contrast, the FCN has full freedom to use linear coefficients tailored to detect local specifics (such as green dot at a location specific to a certain class). This leads the label prediction to strongly rely on specific weight combinations. As training progresses, to achieve higher accuracy, the network learns to use structure instead of "overfitting" on easy samples. Hence, structural use slowly outweighs the reliance on coefficient values and removal of this structure leads to a decrease in performance of the extracted function increasingly similar to the observations for CNNs. An experiment, where we visually inspect misclassified images (Appendix E.2) is in line with this interpretation.

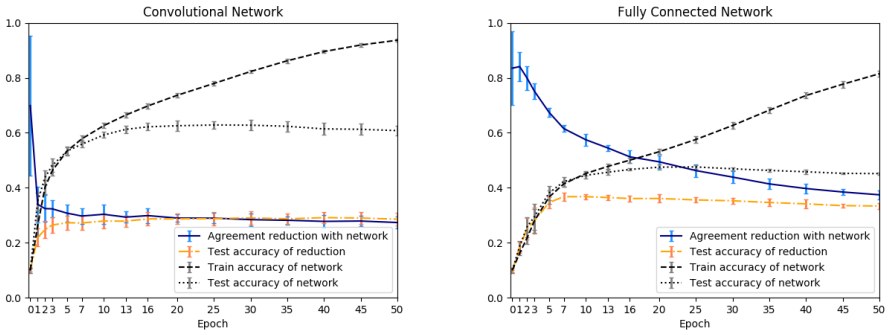

Figure 3: Mean and standard deviation of the performance of extracted tropical function during training over 5 networks. Fully connected and convolutional networks show significant differences.

Examining Function Coefficients With test samples falling into different regions than training samples, it is conceivable that neighboring regions could still be "nearly identical" and test samples would fall into such neighboring regions of training samples. To clarify, we ran an experiment to test similarity of linear regions for test samples before and after reduction in two different ways. We take a test sample and extract the linear coefficient of its linear region in both the original network $\mathcal{N}$ and in the extracted network $\mathcal{N}^{(\mathcal{X})}$. For these linear coefficient vectors, we calculate the (i) angle and (ii) Euclidean norm difference. Figure 4 shows both values for all test samples, where we differentiate between those test points $\mathbf{x}^{(te)}$ that get correctly classified by $\mathcal{N}^{(\mathcal{X})}$ (in blue) and those that get a wrong label (in red). For CNNs, the coefficient vectors of the both training and test affine functions are all close to orthogonal for correctly as well as incorrectly classified points, For FCNs, angle and distance of correctly classified points are smaller than for incorrectly classified ones. These results rule out the possibility of test samples falling into very similar neighboring regions. It further supports the interpretation that generalization in CNNs is more dependent on the structure imposed by, e.g. weight sharing, and less dependent on dominant linear coefficient values. While the angles observed for the CNN are almost consistently orthogonal, there is a clear correlation for the FCN between small angles and correct classification by the extracted function.

5 Discussion and Conclusion

Potential for Future Work.

• *Adversarial Attacks.* Since large linear regions add adversarial robustness (Croce et al., 2018) and TropEx outputs a network function with maximally enlarged linear regions, it may provide a tool for

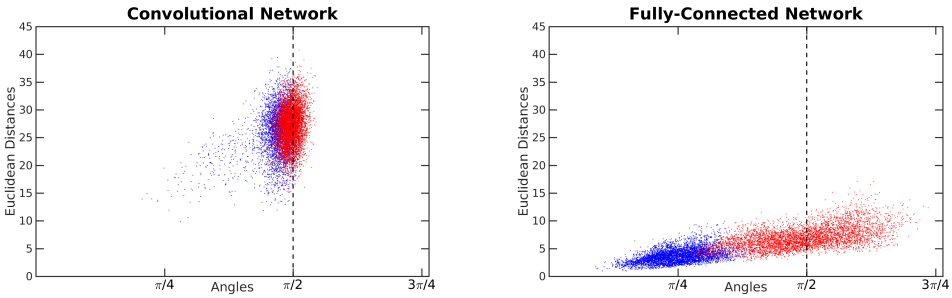

Figure 4: Angle and distance between coefficients of the linear regions of test data and training data used for correct (blue) and incorrect (red) classification by the extracted function. All angles for CNNs (left) are close to orthogonal, while FCNs (right) shows clear correlations between angles, distances and correctness of prediction.

studying various adversarial attacks. It would also be interesting to see whether adversarial attacks can harm the performance of the extracted function $\mathcal{N}^{(\mathcal{X})}$ on training data.

• *Network Pruning.* Our experiments show that removing the structure from a network $\mathcal{N}$ causes a significant drop in test accuracy while keeping the training accuracy unchanged. Zhang et al. (2018) show that tropical rational maps are the same as ReLU networks, so it is possible to obtain a new architecture $\mathcal{N}'$ from $\mathcal{N}^{(\mathcal{X})}$. From a perspective of linear regions, TropEx outputs a minimal function that is unchanged on all training data. Since different network architectures can describe the same network function, it remains to understand what additional structure needs to be imposed on the smaller architecture to obtain high test accuracy as well. This opens the possibility for a pruning technique with unchanged predictions on the data it was trained on.

• *Generalization theory.* Deep neural networks can learn data perfectly (Zhang et al., 2016), which results in a network function that correctly classifies any training sample. For such over-parameterized models, Belkin et al. (2018) study the performance on test data from a viewpoint of interpolation. Our extracted function $\mathcal{N}^{(\mathcal{X})}$ agrees with the network $\mathcal{N}$ on the data set $\mathcal{X}$, hence their difference in generalization must depend on how they interpolate between the training data $\mathcal{X}$. By observing consistent differences between CNNs and FCNs, our results suggest that the architecture plays a large role in this interpolation process. Tailored experiments with TropEx can give more insight on the connection between network architecture and interpolation performance and support the derivation of bounds on the generalization error for neural networks.

Conclusion

The function of a ReLU network is piecewise linear, with an astronomically high number of linear regions. We introduced TropEx, an algorithm for extracting linear regions based on data points. The derivation of the algorithm is based on a matrix representation of tropical functions. Applying TropEx to fully-connected and convolutional architectures shows a drastic drop in test accuracy for CNNs, compared to a relatively small drop for FCNs. The reason for this difference can be found in the coefficients of the extracted linear terms: For CNNs, the enlarged linear regions of training data on which test data lies after extraction differ in terms of angle and distance, whereas for FCNs they are more similar, in particular for those points where the label is unchanged after the extraction. These findings indicate that research on linear regions should shift its focus from counting to an understanding of their interplay.

REFERENCES

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

Mikhail Belkin, Daniel J Hsu, and Partha Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In *Advances in neural information processing systems*, pp. 2300–2311, 2018.

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

Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Juergen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *arXiv preprint arXiv:1003.0358*, 2010.

Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 84, 2018.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature volume 405*, pp. 947–951, 2000.

Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. In *Advances in Neural Information Processing Systems*, pp. 359–368, 2019a.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

Guang-He Lee, David Alvarez-Melis, and Tommi S. Jaakkola. Towards robust, locally linear deep networks. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Guido Montúfar. Notes on the number of linear regions of deep neural networks. *Presented at Mathematics of Deep Learning, Sampling Theory and Applications*, 2017.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pp. 2924–2932, 2014.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2847–2854, 2017.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. *Proceedings of the 35th International Conference on Machine Learning*, 80:4558–4566, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.