Multiclass Neural Network Minimization via Tropical Newton Polytope Approximation

Georgios Smyrnis [1] **Petros Maragos** [1]

Abstract

The field of tropical algebra is closely linked with the domain of neural networks with piecewise linear activations, since their output can be described via tropical polynomials in the max-plus semiring. In this work, we attempt to make use of methods stemming from a form of approximate division of such polynomials, which relies on the approximation of their Newton Polytopes, in order to minimize networks trained for multiclass classification problems. We make theoretical contributions in this domain, by proposing and analyzing methods which seek to reduce the size of such networks. In addition, we make experimental evaluations on the MNIST and Fashion-MNIST datasets, with our results demonstrating a significant reduction in network size, while retaining adequate performance.

1. Introduction

Minimax algebra (Cuninghame-Green, 1979) and tropical geometry (Maclagan & Sturmfels, 2015) are fields of mathematics with various applications, such as in optimization (Cohen et al., 2004; Akian et al., 2012), dynamical system analysis (Butkovič, 2010; Maragos, 2017) and more. This form of algebra corresponds to the analysis of the max-plus semiring, which is defined as $(\mathbb{R} \cup \{-\infty\}, \max, +)$, meaning that regular addition and multiplication are substituted with maximum and addition, respectively. Other works also consider the dual version of this algebra, that is, min-plus algebra. In what follows, we shall consider the name "tropical algebra" to refer to the max-plus version.

While the above field of mathematics has long been studied, the recent advent of neural networks has provided it with another application of great interest, which is the study of such networks with piecewise linear activations. Neural networks with ReLU activations, which fall under this category, are closely linked with the field of tropical algebra, with recent studies demonstrating that their output can be expressed by the use of polynomials in the above semiring, henceforth referred to as *tropical polynomials* (Charisopoulos & Maragos, 2017; 2018; Zhang et al., 2018). Therefore, further analysis of such polynomials is expected to be useful in the goal of furthering the study of such networks.

To that end, in recent studies (Smyrnis & Maragos, 2019; Smyrnis et al., 2020) we introduced a process called *Approximate Tropical Polynomial Divsion*, which attempts to emulate the normal Euclidean division of regular polynomials, when working with tropical polynomials. In that work, we applied this process to the task of the minimization of neural networks, trained for a binary classification problem. This task is of interest, given that it is often the case that neural networks require several redundant parameters during training, which may however be eliminated afterwards (Luo et al., 2017; Han et al., 2015).

In this work, we further this initial study on the division of multivariate tropical polynomials via a single divisor, by examining how the process we previously introduced can be extended to networks trained for multiclass classification problems, an extension which requires the simultaneous approximation of several Newton polytopes of tropical polynomials. We make theoretical contributions by analyzing such extensions, and we also perform experiments on the MNIST and Fashion-MNIST datasets to validate some of our theoretical approaches, with results demonstrating adequate performance for the methods studied.

The rest of this work is structured as follows. In Section 2 we perform a review of certain previous works related to this paper. In Section 3 we further analyze some key points from our previous work, upon which we build. In Section 4 we present extensions of this minimization process in the case of neural networks trained for multiclass classification problems, and in Section 5 we present an alternative, more stable, minimization method for the single class case, to be used in the aforementioned multiclass extensions. Finally,

[1]School of ECE, National Technical University of Athens, Athens, Attiki, Greece. Also: Robot Perception and Interaction Unit, Athena Research Center, Maroussi, Greece. Correspondence to: Georgios Smyrnis <geosmirnis@gmail.com>, Petros Maragos <maragos@cs.ntua.gr>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

in Section 6 we perform certain experiments in order to demonstrate the results of our method.

2. Related Work

2.1. Tropical Algebra and Neural Networks

The field of tropical algebra has recently been extensively linked with that of neural networks (Charisopoulos & Maragos, 2017; 2018; Zhang et al., 2018). It is known that the functions corresponding to feedforward networks with ReLU activations can be expressed as the difference of two tropical polynomials. These are the result of the substitution of addition with maximum and multiplication with addition, in a regular polynomial. Thus, a tropical polynomial is the following convex, piecewise linear function:

$$p(\boldsymbol{x}) = \max_{i=1}^{k} \left\{ \boldsymbol{a}_i^T \boldsymbol{x} + b_i \right\}, \boldsymbol{x} \in \mathbb{R}^d \quad (1)$$

where $\boldsymbol{a}_i$ are the *tropical degrees* (slopes) of its terms, and b_i the tropical coefficients. As such, they are the maximum (instead of sum) of several linear terms, given that each term $c_i \boldsymbol{u}^{\boldsymbol{a}_i}$ of a regular polynomial (with positive variables and coefficients) is converted to a term $\boldsymbol{a}_i^T \boldsymbol{x} + b_i$, where $b_i = \log c_i,\ x = \log u$, since multiplication becomes addition.

The approximation as a difference of two convex functions also leads to Log-Sum-Exp networks (Calafiore et al., 2019; 2020). These networks, which contain exponential activations in the hidden layer and logarithmic in the output, are universal approximators, since they calculate functions via several convex components. Moreover, their structure is closely related to tropical algebra, given the fact that they contain a temperature factor T which, if small enough, makes them approach networks with ReLU activations.

2.2. Network Minimization

The task of minimizing a given network has long been studied in the literature (LeCun et al., 1990; Hassibi & Stork, 1993; Castellano et al., 1997). More recent studies have introduced various methods for the task of minimizing a given network, either by removing unimportant neurons (He et al., 2017; Luo et al., 2017) or by pruning connections and neurons from the network (Han et al., 2015).

The above studies achieve great decrease in the size of the network, while at the same time retaining high classification accuracy. Their authors agree in the fact that, while a high number of parameters is required to properly train a network, most of them can be removed afterwards without loss in performance, as demonstrated by their experimental results.

2.3. Tropical Polynomial Division

When working with polynomials over fields, it is possible to define Euclidean division of polynomials. More precisely, assuming that $p(\boldsymbol{x})$ and $d(\boldsymbol{x})$ are two polynomials over a regular field, Euclidean division seeks to identify two polynomials $q(\boldsymbol{x})$ and $r(\boldsymbol{x})$, such that:

$$p(\boldsymbol{x}) = q(\boldsymbol{x})d(\boldsymbol{x}) + r(\boldsymbol{x}) \quad (2)$$

with the leading term of d not dividing any term of r (with one variable, this is equivalent to $\deg(d) > \deg(p)$). It is possible to expand the above in the case of multiple divisor polynomials $d_i(\boldsymbol{x})$. The result of this process may vary based on the ordering of $d_i(\boldsymbol{x})$. However, in the case where these polynomials constitute a Groebner basis, the remainder of the result is always zero, if $p(\boldsymbol{x})$ can be precisely factored by these polynomials (Buchberger, 1985).

Nevertheless, it is always possible to satisfy (2), since during the process of division, the cancellation of terms is permitted. However, in the case of tropical polynomials, the corresponding property is:

$$p(\boldsymbol{x}) = \max \left\{ q(\boldsymbol{x}) + d(\boldsymbol{x}), r(\boldsymbol{x}) \right\} \quad (3)$$

In the one dimensional case, factorization of tropical polynomials is feasible, as demonstrated by Speyer and Sturmfels (2009). In other cases however, (3) might be impossible to fully satisfy, since there is no possibility of cancellation of terms when performing operations on a semiring (in our case, there is no way to properly invert the $\max$ operator). As such, division of tropical polynomials may not always be possible, as stated by Maclagan and Sturmfels (2015).

In this context, some works attempt to identify and analyze cases where it is possible to completely satisfy (3) (Crowell, 2019). Another approach is to attempt an approximation of the above condition, using a process similar to residuation, where the best $q(\boldsymbol{x})$ and $r(\boldsymbol{x})$ are found, in order for (3) to hold as tightly as possible. This is the approach we adopted in our previous work, due to its more general nature.

3. Approximate Tropical Polynomial Division

3.1. Tropical Polynomial Division Basics

Let $p(\boldsymbol{x}) = \max_{i=1}^{k} \left\{ \boldsymbol{a}_i^T \boldsymbol{x} + b_i \right\}, \boldsymbol{x} \in \mathbb{R}^d$ be a tropical polynomial that we want to divide by another tropical polynomial $d(\boldsymbol{x}) = \max_{i=1}^{k} \left\{ \tilde{\boldsymbol{a}}_i^T \boldsymbol{x} + \tilde{b}_i \right\}$. These shall henceforth be referred to as the dividend and the divisor, respectively, as is the case in normal polynomial division. In previous works we proposed a method to approximately divide such polynomials (Smyrnis & Maragos, 2019; Smyrnis et al., 2020). The method used in that work performs this division, in the sense that a quotient tropical polynomial $q(\boldsymbol{x})$ and a remainder $r(\boldsymbol{x})$ are found, such that:

$$p(\boldsymbol{x}) \geq \max \left\{ q(\boldsymbol{x}) + d(\boldsymbol{x}), r(\boldsymbol{x}) \right\} \quad (4)$$

To perform this form of division, the *Newton Polytope* $\mathrm{Newt}(p)$ and the *Extended Newton Polytope* $\mathrm{ENewt}(p)$ of the tropical polynomial $p(\boldsymbol{x})$ are approximated.

Definition 1. *The Newton Polytope of the tropical polynomial $p(\boldsymbol{x})$ is the convex hull of the set:*

$$\{(a_{i1}, \ldots, a_{id})\}, i = 1, \ldots, k \tag{5}$$

while its Extended Newton Polytope is the convex hull of:

$$\{(a_{i1}, \ldots, a_{id}, b_i)\}, i = 1, \ldots, k \tag{6}$$

where $a_{i1}, \ldots, a_{id}$ are the individual elements of the tropical degrees $\boldsymbol{a}_i$.

Theorem 1. *The value of $p(\boldsymbol{x})$ depends solely on the vertices which correspond to the upper faces (with respect to the final dimension) of* $\mathrm{ENewt}(p)$.

We refer to Charisopoulos and Maragos (2018) for a detailed proof of this important property.

With a goal of approximating the polytope of the dividend, the method we used in our previous work (Smyrnis et al., 2020) to approximately divide $p(\boldsymbol{x})$ by $d(\boldsymbol{x})$ is as follows:

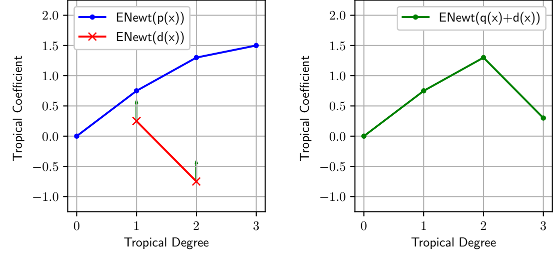
1. Shift $d(\boldsymbol{x})$ so that its Newton polytope lies completely in the interior of that of $p(\boldsymbol{x})$, and mark the necessary shift in degrees as $\boldsymbol{c} \in C = \left\{\boldsymbol{c} \in \mathbb{Z}^d : \mathrm{Newt}\left(\boldsymbol{c}^T\boldsymbol{x} + d(\boldsymbol{x})\right) \subseteq \mathrm{Newt}(p)\right\}$.
2. Raise $d(\boldsymbol{x})$ as much as possible, so that $\mathrm{ENewt}(d)$ stays lower than $\mathrm{ENewt}(p)$. This increase is $q_{\boldsymbol{c}} = \max\left\{q \in R : p(\boldsymbol{x}) \geq q + \boldsymbol{c}^T\boldsymbol{x} + d(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^d\right\}$.
3. Set $q_{\boldsymbol{c}} = \max_{\boldsymbol{c} \in C}\left(q_{\boldsymbol{c}} + \boldsymbol{c}^T\boldsymbol{x}\right)$. The terms which are not covered by any possible shift of $\mathrm{Newt}(d)$ make up the remainder polynomial $r(\boldsymbol{x})$.

An example of this procedure can be seen in Figure 1. Note that this approximation is maximal (Smyrnis & Maragos, 2019; Smyrnis et al., 2020).

3.2. Application to Network Minimization for Binary Classification Problems

The ideas discussed in the above can be applied to the minimization of networks trained for binary classification problems. With a single hidden layer and one output neuron, if $\boldsymbol{W}^1, \boldsymbol{b}^1$ and $\boldsymbol{w}^2, b^2$ are the corresponding weights, then the output is:

$$\begin{aligned} o = &\sum_{i=1}^{n_1} w^2_{+,i} \max\left(\sum_{j=1}^{d} w^1_{ij}x_j + b^1_i, 0\right) \\ &- \sum_{i=1}^{n_1} w^2_{-,i} \max\left(\sum_{j=1}^{d} w^1_{ij}x_j + b^1_i, 0\right) + b^2 \end{aligned} \tag{7}$$



(a) Dividend and Divisor. (b) Result of division.

Figure 1. Division of $p(x) = \max(3x+1.5, 2x+1.3, x+0.75, 0)$ by $d(x) = \max(x-1, 0)$. Quotient is $q(x) = \max(2x+1.3, x+0.75, 0)$.

where $\boldsymbol{w}^2 = \boldsymbol{w}^2_+ - \boldsymbol{w}^2_-$ the splitting of the output vector into the parts with positive and negative weights. Each tropical polynomial has its own Newton polytope, and by approximating both it is possible to create a neural network which, while smaller than the original, is still capable of performing adequately in the classification task for which it was trained. The algorithm which we previously used for this task, and upon which we shall build, is Algorithm 1. Algorithm 1 attempts to create a divisor polynomial (for each of the two polynomials of the network), whose polytope contains the most activated vertices of the original. This is due to the fact that, as we will see later on, the vertices of the network polytopes can be expressed as the sum of a subset of its neurons. Thus, by ensuring that the chosen vertices are one such sum, we provide a heuristic via which we attempt to retain the vertices of the original polytope. Moreover, the final step attempts to better approximate the missing vertices, by altering the bias of the output layer in an optimal fashion, in order to minimize the mean squared error of the output (this forms the quotient, if (4) is lifted and instead the difference in outputs is minimized with the given divisor).

3.3. Example of Algorithm 1

Here we can see the application of Algorithm 1 for the minimization of a given neural network. Let us assume that we have a neural network with the following weights:

$$\boldsymbol{W}^1 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{b}^1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

$$\boldsymbol{w}^2 = \begin{bmatrix} 1 & 1 & 2 & 2 \end{bmatrix} \qquad b^2 = 0$$

and ReLU activation in the hidden layer, as well as the following dataset:

$$D = \left\{ \begin{matrix} [1,1]^T, & [1,2]^T, & [2,1]^T, \\ [1,-0.5]^T, & [1,-0.75]^T, & [-0.5,1]^T \end{matrix} \right\} \tag{9}$$

Algorithm 1 Heuristic Minimization Algorithm (Smyrnis et al., 2020)

1. Select the neurons of the hidden layer (with their weights and bias), corresponding to positive weights in the output layer.
2. For each set of these neurons (vertex of the Newton Polytope), calculate how many samples activate it.
3. Sort the above vertices in descending order.
4. Set the first vertex $\boldsymbol{v}_1$ as the neuron $\boldsymbol{w}_1$, and for each $\boldsymbol{v}_k$, set $\boldsymbol{w}_k = \boldsymbol{v}_k - \boldsymbol{w}_j$, where $\boldsymbol{w}_j$ a random previously added neuron (so that their sum is included in the new polytope).
5. Repeat for the negative weights of the output layer, and set output weights equal to ± 1 (for positive and negative part, respectively).
6. Calculate the mean difference of outputs (disregarding the output activation and bias), and add this difference to the bias of the output layer (plus any original output bias).

Since the weights of the output neuron are positive, we can integrate them in the hidden layer. Thus, the above network has a polytope whose vertices are subset sums of the vectors containing the weights and biases of the hidden layer:

$$\begin{aligned} \boldsymbol{p}_1 = [1,2,1]^T, \quad \boldsymbol{p}_2 = [2,1,1]^T, \\ \boldsymbol{p}_3 = [2,0,0]^T, \quad \boldsymbol{p}_4 = [0,2,0]^T \end{aligned} \tag{10}$$

By calculating the outputs of the elements of D, we see that the first three samples activate all neurons, the next two samples activate the first three neurons, and the final sample all the neurons but the third. These correspond to three activations of the vertex $\boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_3 + \boldsymbol{p}_4 = [5,5,2]^T$, two of the vertex $\boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_3 = [5,3,2]^T$ and one of $\boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_4 = [3,5,2]^T$. Thus, performing the algorithm, the first neuron added is the first vertex, so $\boldsymbol{w}_1 = [5,5,2]^T$. The second neuron is the difference of the second vertex with $\boldsymbol{w}_1$, so $\boldsymbol{w}_2 = [5,3,2]^T - [5,5,2]^T = [0,-2,0]^T$. For the third neuron, we subtract the first one (randomly chosen) from the third vertex, so $\boldsymbol{w}_3 = [3,5,2]^T - [5,5,2]^T = [-2,0,0]^T$. Thus, the new hidden layer is:

$$\boldsymbol{W'}^1 = \begin{bmatrix} 5 & 5 \\ 0 & -2 \\ -2 & 0 \end{bmatrix}, \ \boldsymbol{b'}^1 = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} \tag{11}$$

In this case, the outputs of the network for D can be found to be the same, before and after the minimization, so there is no output bias. Thus, we have constructed a smaller network, which has the same output as the original for the dataset.

4. Multiclass Minimization

In the previous section, we saw how tropical polynomial division, and the corresponding approximation of Newton polytopes, can be used to minimize a network with a single output neuron, hinting to their use in more complicated, multiclass settings. However, in the case of multiclass problems, the number of output neurons, and thus that of tropical rational functions corresponding to the network, increases. The main difficulty of this extension is the fact that we want to approximate several, interconnected, tropical polynomials and tropical Newton polytopes. As such, minimizing the network becomes more difficult, since this approximation must be simultaneous, and the network does not offer, by itself, enough degrees of freedom to do so immediately. Note that the multiclass extension is a necessary step, in order to move on to deep architectures (possibly via sequential examination of each layer).

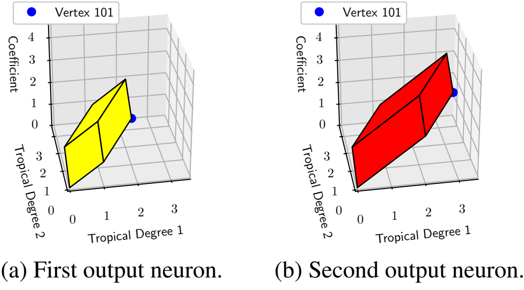
In this section, we shall examine how this minimization task can be performed. The methods we provide shall attempt to approximate several polytopes at once, corresponding to the output neurons of the network. We shall study the case of two-layer feedforward networks with ReLU activations, since their simple structure serves to easily introduce our methods. In this case, it is possible to see the following points, regarding the structure of the polytopes of the network in this simple case:

- Each vertex of the polytope corresponds to the sum of a combination of neurons. The samples which activate this combination correspond to this vertex.
- Each vertex also has a natural binary labeling, via the combination of neurons from which it is constructed.

To illustrate these points, let us think of an example, where:

$$\boldsymbol{W}^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{b}^1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \boldsymbol{W}^2 = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \tag{12}$$

and there is no bias in the output layer. In this case, we have a single Newton polytope per output neuron (due to all the weights being positive). In this example, assuming i.e. an input $\boldsymbol{x} = [1, -1.5]^T$, the hidden layer corresponds to $\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1 = [2, -0.5, 0.5]^T$. Given that only the first and the third neurons of the hidden layer are activated, we can state that this sample activates the output vertex of the polytope with index 101, corresponding to this combination of neurons. Note that this index is shared among the polytopes of the two output neurons, since it depends solely on the common hidden layer. This is also shown in Figure 2, where the polytopes of both output neurons are visible.



(a) First output neuron. (b) Second output neuron.

Figure 2. Upper hulls of polytopes of output neurons from the example in Section 4.1. The common vertex is also shown.

4.1. Minimization via Vertex Transformation

In order to address the problem of multiclass minimization via tropical Newton polytope approximation in a simple fashion, we can attempt to approximate a single Newton polytope, and then modify the result in order to approximate all polytopes corresponding to the network at once. Let:

$$\boldsymbol{W}^1 = \left[w_{ij}^1\right], \boldsymbol{W}^2 = \left[w_{ij}^2\right] \tag{13}$$

be the $n_1 \times d$, $n_2 \times n_1$ matrices, corresponding to the neuron weights in each layer, with n_1, n_2 being the number of neurons in the hidden and the output layer, respectively. For the following, we shall assume that the matrix $\boldsymbol{W}^2$ contains only positive entries (we shall afterwards describe a way to get around this limitation). This way, we want to approximate one polytope per output neuron. Let us focus on a single output neuron, namely the one with weights $\boldsymbol{w}_l^2$. Its polytope is constructed by the following vectors:

$$w_{lj}^2 \left[w_{j1}^1, w_{j2}^1, \ldots, w_{jd}^1, b_j^1\right]^T, j = 1, \ldots, n_1 \tag{14}$$

each corresponding to a neuron of the hidden layer. At this point, we can make use of the aforementioned binary indices, which stem from the hidden layer of the network. Algorithm 1 implicitly takes into account these indices, when calculating the corresponding vertex of the output polytope. Indeed, given the aforementioned observations regarding the vertices of the polytope of the original networks, any of them, let $\boldsymbol{v}$, can be described via the vertices $\boldsymbol{v}_j = \left[w_{j1}^1, w_{j2}^1, \ldots, w_{jd}^1, b_j^1\right]^T$ of the hidden layer neurons:

$$\boldsymbol{v} = \sum_{j \in I} w_{lj}^2 \boldsymbol{v}_j = \left(\bar{\boldsymbol{W}}^1\right)^T \mathrm{diag}\left(\boldsymbol{w}_l^2\right) \mathbb{1}_I \tag{15}$$

where:

- I is the set of activated vertices of the hidden layer, corresponding to the index of this vertex,
- $\mathbb{1}_I$ is the binary column vector with 1 in the rows corresponding to the neurons of this set,
- $\bar{\boldsymbol{W}}^1$ is an extended form of the weight matrix of the hidden layer, whose final column contains the biases of the neurons.

This representation is the same across the operations performed during Algorithm 1, with the only difference being that the binary vector is instead replaced with an arbitrary integer vector $\boldsymbol{t}$ (due to the differences calculated by the algorithm), as seen in the following:

$$\boldsymbol{v} = \left(\bar{\boldsymbol{W}}^1\right)^T \mathrm{diag}\left(\boldsymbol{w}_l^2\right) \boldsymbol{t} \tag{16}$$

Let us assume that first, the above algorithm is applied to our network, albeit considering only one output neuron, with all of its weights equal to 1, and that the result is a hidden layer with a corresponding matrix $\bar{\boldsymbol{W}}'^1$. We want to approximate the above vertices, using those of $\bar{\boldsymbol{W}}'^1$, as well as the weights of the output neurons. For the lth output neuron, we want to solve:

$$\boldsymbol{v}'_j w_{lj}^2 = \left(\bar{\boldsymbol{W}}^1\right)^T \mathrm{diag}\left(\boldsymbol{w}_l^2\right) \boldsymbol{t}_j,\ j = 1, \ldots, n'_1 \tag{17}$$

where $\boldsymbol{v}'_j$ is the jth column of $\bar{\boldsymbol{W}}'^1$. An approximate solution can easily be found, using the pseudoinverse of v'_j:

$$w_{lj}^2 = \frac{1}{\|\boldsymbol{v}_j\|_2^2} \boldsymbol{v}'^T_j \left(\bar{\boldsymbol{W}}^1\right)^T \mathrm{diag}\left(\boldsymbol{w}_l^2\right) \boldsymbol{t}_j,\ j = 1, \ldots, n'_1 \tag{18}$$

Using the above idea, we can approximate multiple outputs from all neurons at once, using the following method:

- First perform the algorithm to approximate a neural network with only a single output neuron, assuming that all of its weights are equal to 1. This approximation results in a weight matrix $\bar{\boldsymbol{W}}'^1$, for the hidden layer of the network. While doing so, also keep note of the vectors $\boldsymbol{t}_j$, corresponding to each of these neurons.
- For each output neuron l of the network, calculate its weights w_{lj}^2, using (18).

This way, we first create a baseline Newton polytope. We then transform its vertices, in an attempt to approximate their representation in the space of the original polytope (that is, the points which would have resulted by adding the vertices in the same sequence in each separate polytope).

In the above, we made the assumption that the weights of the output layer are strictly positive. This assumption can be lifted, by splitting the weight matrix of the output layer (and thus, the output layer itself) into two parts, each with positive weights, as follows:

$$\boldsymbol{W}^2 = \boldsymbol{W}_+^2 - \boldsymbol{W}_-^2 \tag{19}$$

Note that this is always possible, regardless of the actual weights of the output layer. As such, we can treat each part as a separate output layer (with a common hidden layer), and recombine them after we approximate them individually, by setting the final weight matrix as their difference.

4.2. Minimization as a One-Versus-All Problem

The above method to approximate the vertices of the polytope via a transformation allows us to perform an arbitrary minimization on the hidden layer of the network. However, when calculating the importance of each vertex, it treats the samples the same way, regardless of their class. This might not be ideal, given that, intuitively, the ability to handle samples belonging to separate classes differently appears to be of great importance. As such, we elect to regard the multiclass minimization problem as a one-versus-all variation, rather than a complete, single problem.

More precisely, let us examine a network with one hidden layer, with weights and biases $\boldsymbol{W}^1, \boldsymbol{b}^1$ and $\boldsymbol{W}^2, \boldsymbol{b}^2$ for the first and the second layer, respectively. It is possible to create an equivalent network, by adding more neurons in the hidden layer, so that the weight matrix of the output layer becomes block diagonal. In particular, we can use the following matrices for the two layers:

$$\begin{aligned} \tilde{\boldsymbol{W}}^1 &= \begin{bmatrix} \boldsymbol{W}^1 \\ \vdots \\ \boldsymbol{W}^1 \end{bmatrix} & \tilde{\boldsymbol{b}}^1 &= \begin{bmatrix} \boldsymbol{b}^1 \\ \vdots \\ \boldsymbol{b}^1 \end{bmatrix} \\ \tilde{\boldsymbol{W}}^2 &= \begin{bmatrix} \boldsymbol{w}_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \boldsymbol{w}_C^2 \end{bmatrix} & \tilde{\boldsymbol{b}}^2 &= \boldsymbol{b}^2 \end{aligned} \tag{20}$$

where $\boldsymbol{w}_i^2$ is the ith row of $\boldsymbol{W}^2$ and C the number of classes to be distinguished, which is also equal to the number of output neurons of the network. The resulting network has C times the neurons of the original in its hidden layer, but corresponds to the same function as the original.

The process used for the one-versus-all minimization can be seen in Algorithm 2. The resulting network contains the desired number of neurons, given that C copies of the hidden layer are created, for each output neuron to be independent of the others. This way, we approximate the polytopes of each output neuron as separate entities from the rest.

Of note is the second step of Algorithm 2. Assuming that the original dataset is balanced, if the one-versus-all procedure is applied as is, then for each output neuron there will be $C - 1$ negative samples (other classes) for each positive sample (class of this output neuron). Dynamically weighting the appearances of each sample allows us to artificially rebalance this dataset, in order for both the positive and negative samples of each class to influence the result.

Algorithm 2 One-Versus-All Multiclass Minimization

Goal: Create a new network with an arbitrary percentage f of the original hidden layer.

1. Perform the single output minimization algorithm, separately for each output neuron (considering only its corresponding copy of the matrix $\boldsymbol{W}^1$), while keeping only f/C neurons for each part.
2. During the previous step, when minimizing the polytope corresponding to the ith output neuron, add $C - 1$ appearances for each sample with label i, and only 1 for each other sample.
3. Accumulate the results for both layers, and create a single matrix for all classes (either by stacking them vertically for the hidden layer, as before, or by creating a block diagonal matrix for the output layer).
4. Set the bias for the output layer via a weighted average difference of activations (calculated as in Algorithm 1), with samples belonging to the same class as the output neuron having a weight of $C - 1$ as before.

It is also important to notice that, given the nature of Algorithm 1, at least one neuron is given as output for the positive and the negative polytope corresponding to the network. This provides a lower limit to the number of neurons in the hidden layer of the network constructed by our method, given that there will be, at the very least, 2 neurons for each of the C classes. This limit is supported by the theoretical fact that, with only one neuron with a ReLU activation, we cannot approximate a general, non-convex function.

5. Stable Single Output Minimization Algorithm

5.1. Algorithm Definition

Due to the increased complexity in the multiclass minimization tasks we examine in this work, Algorithm 1 might be inadequate in this case, since it adds vertices outside of the original polytope might be added in the new network, which might lead to high difference in outputs. In this context, we propose an alternative method by which to select a divisor polynomial for each of the classes. This alternative method is also applicable in the case of binary classification networks, however we expect its value to be demonstrated in our, more complicated, task.

Let us first assume that we analyze a single output neuron of the network, and the vertices of the polytope are assigned weights and sorted as previously described. We require our result to contain neurons which can be constructed via com-

Algorithm 3 Stable Divisor Picking Algorithm

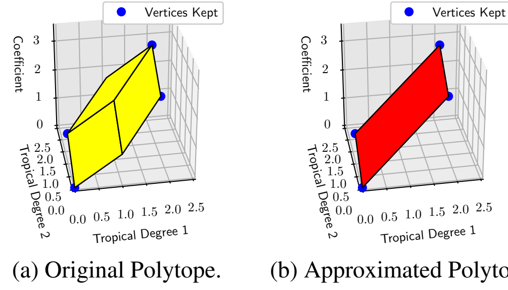
1. Pick the part of the network corresponding to positive (or negative) output weights, and rank the importance of vertices (weights and bias of hidden layer neurons, multiplied by output weight) as before, converting them to their binary representation.
2. Add the first vertex $\boldsymbol{v}_1$ as a neuron, and initialize an accumulator $\boldsymbol{a} = \boldsymbol{v}_1$. Let $k = 1$ be the number of neurons already inserted.
3. For each new vertex $\boldsymbol{v}_i$:
   - If it has no common neurons with any of the previous ones ($\boldsymbol{a} \wedge \boldsymbol{v}_1 = 0 \ldots 0$), set $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{v}_i$, $\boldsymbol{a} \leftarrow \boldsymbol{a} \vee \boldsymbol{v}_i$.
   - Otherwise, $\boldsymbol{c} = \boldsymbol{a} \wedge \boldsymbol{v}_i$. For each $\boldsymbol{w}_j$ inserted, split it into two neurons:
   $$\boldsymbol{w}_j \leftarrow \boldsymbol{w}_j \wedge \neg \boldsymbol{c},\ \boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_j \wedge \boldsymbol{c}$$
   if both are nonzero. After checking all neurons, add one more, $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{v}_i \wedge \neg \boldsymbol{c}$, if nonzero.

   In any case, increment k as neurons are added, and set $\boldsymbol{a} \leftarrow \boldsymbol{a} \vee \boldsymbol{v}_i$.
4. Repeat until a given k, and replace all new neurons with their actual weights (sums of the original neurons which construct them).

binations of neurons in the original network, which leads to the polytope of the divisor being inside of that of the original network. What this means is that we want to choose distinct subsets of the neurons, each with a score based on the number of corresponding samples. This problem is similar to *Set Packing* (Karp, 1972), but here the score of a subset is defined by all the sets that we choose (since, for example, picking any two vertices causes us to implicitly pick their sum). In order to approximately maximize the number of samples covered by the chosen divisor, we propose an alternative algorithm to picking a divisor, namely Algorithm 3. This algorithm is, once again, performed separately for the positive and negative parts of the network with output bias being added as before.

Our algorithm adds neurons so that unions of some of their subsets gives us the binary labels of the chosen vertices. These are matched exactly, as seen in Figure 3. Of note is the splitting process in step 3, where, for example, to add vertices with labels $1110, 0111$, we add the neurons $1000, 0110, 0001$, to retain the vertices as subset sums.

Theorem 2. *The resulting polytope of the divisor contains only vertices inside of the original network polytope.*



(a) Original Polytope. (b) Approximated Polytope.

Figure 3. Creation of divisor by the use of Algorithm 3. Left is the upper hull of the polytope of a network with three neurons. On the right is the approximated network with two neurons, using the marked vertices.

Proof. This is due to the fact that, in each step, the previous neurons are modified, setting them as $\boldsymbol{w} \wedge \neg \boldsymbol{c}$, removing bits from their binary labels so that each neuron of the original hidden layer is contained at most once. Since the vertices of the divisor polytope are subset sums of the new neurons, all of them will also correspond to sums of the original neurons, that is, points of the original polytope. As such, the Newton polytope of the divisor is inside the original one. □

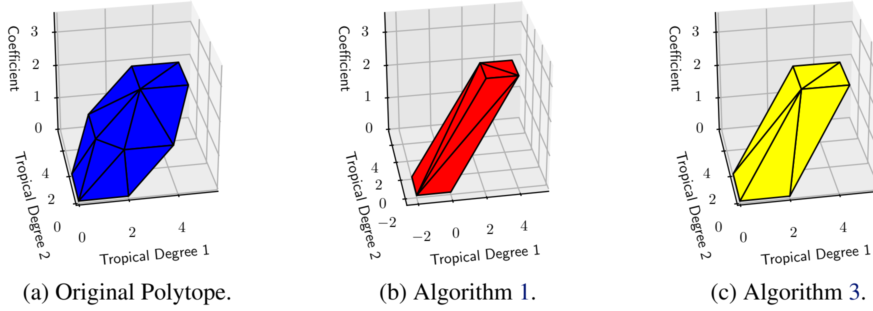
Theorem 3. *The samples corresponding to the chosen vertices have the same output in the new divisor, as they did in the original network (if the output bias is ignored).*

Proof. This is due to the entire polytope of the divisor being contained inside of the original, and the chosen vertices matching exactly. Thus, the corresponding samples need to retain their output (there are no points outside of the original polytope, to influence the result). □

Theorem 4. *If n_1 is the number of neurons in the original hidden layer (corresponding to either positive or negative output weights), N the total number of samples, and n' the number of desired neurons, then the samples which certainly correspond to the chosen vertices are at least:*

$$\frac{N}{\sum_{j=0}^{d} \binom{n_1}{j}} O\left(\log n'\right) \tag{21}$$

Proof. The maximum length of the list of counts of activated vertices is equal to $\sum_{j=0}^{d} \binom{n_1}{j}$, the number of linear regions in the polynomial of the network (Zhang et al., 2018). Since the ordering is descending, the sum of the counts of the chosen vertices is at least $NK \Big/ \sum_{j=0}^{d} \binom{n_1}{j}$, if the N samples are evenly distributed across the list. Finally, K, the number of polytope vertices which are certainly picked, is at least $O\left(\log n'\right)$, since if all previous



(a) Original Polytope. (b) Algorithm 1. (c) Algorithm 3.

Figure 4. Network approximation example, the polytope created by Algorithm 3 is closer to the original.

neurons change when adding a vertex, then their number is doubled. All of the above lead to the desired limit (the limit holds separately for the positive and negative parts). □

The above lower bound might be significantly lower than the actual value, since it does not consider the additional vertices covered by the extra neurons added in each step, whose number depends on the actual neurons.

Note that Algorithm 3, while rigorous, is quadratic to the number of neurons n' in the final hidden layer. As such, Algorithm 1 is faster, but also likely less stable when used in the multiclass case. Thus, we shall make use of both in the one-versus-all format discussed in the previous section, in order to minimize a network with multiple output neurons.

5.2. Example for Algorithm 3

To compare this new algorithm with Algorithm 1, we can examine the same example presented in Section 3.3. If we perform Algorithm 3 this time, the activated vertices are labeled as 1111, 1110 and 1101. First we set $\boldsymbol{w}_1 = [5, 5, 2]^T$ as before, but to add vertex 1110 we split it:

$$\boldsymbol{w}_1 = \boldsymbol{v}_{1111 \wedge \neg 1110} = \boldsymbol{v}_{0001} = \boldsymbol{p}_4 = [0, 2, 0]^T$$
$$\boldsymbol{w}_2 = \boldsymbol{v}_{1111 \wedge 1110} = \boldsymbol{v}_{1110} = \boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_3 = [5, 3, 2]^T \tag{22}$$

Similarly, we then add vertex 1101. Note that in this case, the first neuron, currently 0001, would give us an empty neuron: $0001 \wedge \neg 1101 = 0000$, so we do not split it. For the second neuron, we set:

$$\begin{aligned} w_2 = \boldsymbol{v}_{1110 \wedge \neg 1101} = \boldsymbol{v}_{0010} = \boldsymbol{p}_3 = [2, 0, 0]^T \\ w_3 = \boldsymbol{v}_{1110 \wedge 1101} = \boldsymbol{v}_{1100} = \boldsymbol{p}_1 + \boldsymbol{p}_2 = [3, 3, 2]^T \end{aligned} \tag{23}$$

As such, the new hidden layer is:

$$\boldsymbol{W'}^1 = \begin{bmatrix} 0 & 2 \\ 2 & 0 \\ 3 & 3 \end{bmatrix}, \ \boldsymbol{b'}^1 = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \tag{24}$$

It can be seen that, in this case as well, the outputs of the network over the dataset are the same in the new network. However, in Figure 4, we can see the polytopes constructed by the two algorithms. Algorithm 3 creates a polytope which is closer to the one of the original, as was expected.

6. Experiments

We now perform certain experiments for the methods discussed above. In particular, we shall test the one-versus-all formulation of multiclass minimization, both with the simple minimization procedure described in Algorithm 1, as well as that proposed in Algorithm 3, which we shall refer to as *Heuristic* and *Stable Minimization*, respectively. For our experiments we made use of Keras for writing our code (Chollet et al., 2015). The experiments were performed on a Core i5-7200U CPU clocked at 2.5GHz, with 8GB of RAM and 8GB of swap space. The code used is provided as supplementary material, and can also be found here: `https://github.com/GeorgiosSmyrnis/multiclass_minimization_icml2020`.

We made use of the MNIST dataset of handwritten digits (Lecun et al., 1998), as well as Fashion-MNIST (Xiao et al., 2017), to evaluate our methods (available at `http://yann.lecun.com/exdb/mnist/` and `https://github.com/zalandoresearch/fashion-mnist`, respectively). Five runs were performed in total, each beginning by training a simple network over the dataset. In particular, for the MNIST dataset, the network consisted of two convolutional layers, each with 16 units, 5×5 kernels, ReLU activations and maxpooling with a factor of 3. The output of these layers was then fed to a two-layer feedforward network, with ReLU activations and 500 neurons in the hidden layer. For the Fashion-MNIST dataset, the structure was similar, except the convolutional layers contained 32 units and the hidden layer in the end 1000 neurons. These hyperparameters where not optimally chosen via cross-validation, but serve well in demonstrating the results of the minimization task at hand.

Table 1. MNIST Accuracy, Heuristic Minimization (Algorithm 1).

Percentage of Neurons Kept	Mean Acc.	St. Dev.
100% (Original)	98.604	0.027
90%	95.714	1.342
75%	95.048	1.552
50%	95.522	3.003
25%	91.040	5.882
10%	92.790	3.530
5%	92.928	2.589

Table 2. MNIST Accuracy, Stable Minimization (Algorithm 3).

Percentage of Neurons Kept	Mean Acc.	St. Dev.
100% (Original)	98.604	0.027
90%	96.604	1.297
75%	96.560	1.245
50%	96.392	1.177
25%	95.154	2.356
10%	93.478	2.572
5%	92.928	2.589

After training the network for each run, the hidden layer of the feedforward part was minimized. Results are shown in the relevant tables, which record the mean accuracy over the test set and its standard deviation over the five runs. Note that, since our one-versus-all algorithm assigns the same number of neurons for each class, ratios which do not lead to an even distribution of neurons round their number down, so that the final network stays within limits.

6.1. One-Versus-All, Heuristic Minimization (MNIST)

In our first experiment, we evaluate the results of Heuristic Minimization, over the MNIST dataset. The experiments were performed as described previously, with results appearing in Table 1. We see that our method appears to preserve a significant amount of information across all classes. As such, the average accuracy achieved in the test set is close enough to the original. However, there is indeed a high amount of variance from this method, which is to be expected given its random nature.

6.2. One-Versus-All, Stable Minimization (MNIST)

In the next experiment, we test the one-versus-all version of our problem, where instead we use the Stable Minimization, outlined in Section 5. This method makes use of Algorithm 3 in order to find a better divisor polynomial, with the rest of the process being the same as before. The results are summarized in Table 2.

We see that the average accuracy of this method is, in general, higher than that of the previous one. This is likely due to the more accurate approximation of the polytope, in comparison to Heuristic Minimization. Moreover, the variation induced by our method is, in general, comparable or smaller than that of Table 1. This is as expected, especially in the case of a smaller amount of neurons kept, where the random nature of the first method likely induces more variation.

Table 3. Fashion-MNIST Accuracy, Stable Minimization.

Percentage of Neurons Kept	Mean Acc.	St. Dev.
100% (Original)	88.658	0.538
90%	83.634	2.894
75%	83.556	2.885
50%	83.300	2.799
25%	82.224	2.845
10%	80.430	3.267

Note that the final line of both of these tables is the same. This is due to the fact that, as previously described, this level of minimization approaches a lower limit (two neurons per class). Both of these methods keep the same first neuron, and here they only keep one each time they are applied, so their behavior is the same.

6.3. One-Versus-All, Fashion-MNIST

Finally, we test our method in the Fashion-MNIST dataset, using the network previously described. More specifically, we examine the Stable Minimization method, with results in Table 3. We see that the accuracy drop in this case is higher, however the final network is still capable of adequate results in the dataset. This decrease is likely due to the increased difficulty of the dataset, when compared to MNIST. Nevertheless, this result demonstrates the ability of our method to minimize networks trained for similar, multiclass problems.

7. Conclusions and Future Work

In this work, we presented methods to approximate the Newton polytopes of tropical polynomials, in order to minimize networks trained for multiclass classification problems, by expanding and proposing alternatives to our previous work. We also performed experimental evaluations, with results demonstrating an ability to retain high levels of accuracy, despite the lower number of neurons.

In order to advance these subjects, it is important to apply such methods for the approximation of tropical Newton polytopes in more complicated architectures, such as deep and convolutional neural networks. In doing so, it will be possible to compare their results with more traditional network pruning techniques, on well-known architectures and more realistic datasets.

Acknowledgements

The authors would like to thank George Retsinas for the very useful discussions on the topics of this paper. The authors also thank the anonymous reviewers for their very helpful and constructive comments on this paper.

References

Akian, M., Gaubert, S., and Guterman, A. Tropical Polyhedra Are Equivalent To Mean Payoff Games. *Int'l J. Algebra and Computation*, 22(1), 2012.

Buchberger, B. *Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory*, pp. 184–232. D. Reidel Publishing Company, 1985.

Butkovič, P. *Max-linear Systems: Theory and Algorithms*. Springer, 2010.

Calafiore, G. C., Gaubert, S., and Possieri, C. Log-Sum-Exp Neural Networks and Posynomial Models for Convex and Log-Log-Convex Data. *IEEE Trans. Neural Networks and Learning Systems*, 30(5):1–12, May 2019.

Calafiore, G. C., Gaubert, S., and Possieri, C. A Universal Approximation Result for Difference of Log-Sum-Exp Neural Networks. *IEEE Trans. Neural Networks and Learning Systems*, 2020.

Castellano, G., Fanelli, A. M., and Pelillo, M. An Iterative Pruning Algorithm for Feedforward Neural Networks. *IEEE Trans. Neural Networks*, 8(3):519–531, 1997.

Charisopoulos, V. and Maragos, P. Morphological Perceptrons: Geometry and Training Algorithms. In *Proc. Int'l Symp. Mathematical Morphology (ISMM)*, volume 10225 of *LNCS*, pp. 3–15. Springer, Cham, 2017.

Charisopoulos, V. and Maragos, P. A Tropical Approach to Neural Networks with Piecewise Linear Activations. *arXiv preprint arXiv:1805.08749*, 2018.

Chollet, F. et al. Keras. https://keras.io, 2015.

Cohen, G., Gaubert, S., and Quadrat, J. Duality and Separation Theorems in Idempotent Semimodules. *Linear Alegbra and its Applications*, 379:395–422, 2004.

Crowell, R. A. The Tropical Division Problem and the Minkowski Factorization of Generalized Permutahedra. *arXiv preprint arXiv:1908.00241*, 2019.

Cuninghame-Green, R. *Minimax Algebra*. Springer-Verlag, 1979.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems*, pp. 164–171, 1993.

He, Y., Zhang, X., and Sun, J. Channel Pruning For Accelerating Very Deep Neural Networks. In *Proc. Int'l Conf. on Computer Vision*, pp. 1389–1397, 2017.

Karp, R. M. Reducibility among Combinatorial Problems. In *Complexity of computer computations*, pp. 85–103. Springer, 1972.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*, pp. 598–605, 1990.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

Luo, J.-H., Wu, J., and Lin, W. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In *Proc. Int'l Conf. on Computer Vision*, Oct 2017.

Maclagan, D. and Sturmfels, B. *Introduction to Tropical Geometry*. Amer. Math. Soc., 2015.

Maragos, P. Dynamical Systems on Weighted Lattices: General Theory. *Math. Control Signals Syst.*, 29(21), 2017.

Smyrnis, G. and Maragos, P. Tropical Polynomial Division and Neural Networks. *arXiv preprint arXiv:1911.12922*, 2019.

Smyrnis, G., Maragos, P., and Retsinas, G. Maxpolynomial Division with Application To Neural Network Simplification. In *Proc. IEEE ICASSP*, pp. 4192–4196. IEEE, 2020.

Speyer, D. and Sturmfels, B. Tropical Mathematics. *Mathematics Magazine*, 82(3):163–173, 2009.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zhang, L., Naitzat, G., and Lim, L.-H. Tropical Geometry of Deep Neural Networks. In *Proc. Int'l Conf. on Machine Learning*, volume 80, pp. 5824–5832. PMLR, 2018.