arXiv:2409.03945v2 [cs.CV] 8 Nov 2024

TropNNC: Structured Neural Network Compression Using Tropical Geometry

Konstantinos Fotopoulos * KOSTFOTO2001@GMAIL.COM
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, 15773-ECE Building 2.2.19, Greece

Petros Maragos MARAGOS@CENTRAL.NTUA.GR
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, 15773-ECE Building 2.2.19, Greece

Panagiotis Misiakos PMISIAKOS@ETHZ.CH
Department of Computer Science
ETH Zürich
Zürich, Universitätstrasse 6-CAB H 81.2, Switzerland

Abstract

We present TropNNC, a framework for compressing neural networks with linear and convolutional layers and ReLU activations. TropNNC is a structured compression framework based on a geometrical approach to machine/deep learning, using tropical geometry and extending the work of Misiakos et al. (2022). We use the Hausdorff distance of zonotopes in its standard continuous form to achieve a tighter approximation bound for tropical polynomials compared to previous work. This enhancement leads to the development of an effective compression algorithm that achieves superior functional approximations of neural networks. Our method is significantly easier to implement compared to other frameworks, and does not depend on the availability of training data samples. We validate our framework through extensive empirical evaluations on the MNIST, CIFAR, and ImageNet datasets. Our results demonstrate that TropNNC achieves performance on par with state-of-the-art methods like ThiNet (even surpassing it in compressing linear layers) and CUP. To the best of our knowledge, it is the first method that achieves this using tropical geometry.

Keywords: tropical geometry, zonotopes, Hausdorff distance, neural network compression

1 Introduction

In recent years, deep neural networks (DNNs) have significantly advanced the field of computer vision, excelling in tasks such as image classification (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; He et al., 2016; Rawat and Wang, 2017), object detection (Zou et al., 2023), semantic segmentation (Noh et al., 2015; Guo et al., 2018), and image captioning (Jia et al., 2015; Hossain et al., 2019). These networks have outperformed traditional methods reliant on manually crafted visual features. However, despite their success, deploying DNNs on resource-constrained devices, such as mobile phones or embedded systems, remains challenging due to their substantial computational and storage demands. For

∗. Corresponding author

example, the VGG-16 model (Simonyan and Zisserman, 2015), with its approximately 138 million parameters, requires over 500MB of storage and performs about 31 billion FLOPs to classify a single image. Such resource-intensive models exceed the capabilities of small devices and pose significant challenges, including energy limitations recently highlighted by the deployment of large language models (Samsi et al., 2023). This has led to a growing focus on network compression techniques (Blalock et al., 2020) as a critical area of research.

Initial attempts to reduce network complexity date back to the 1990s, when LeCun et al. (1989) demonstrated that removing negligible weights from a network had minimal impact on performance. This approach, known as neural network pruning, has since evolved. Han et al. (2015) proposed a simple yet effective pruning strategy: iteratively removing connections with weights below a certain threshold and fine-tuning the network to recover accuracy, resulting in highly sparse models.

These methods belong to the category of unstructured pruning, which alters the network's structure by eliminating individual weights. While effective in reducing network size, unstructured pruning presents challenges in practical applications. Such pruned networks often require specialized hardware and software for efficient inference due to issues like poor cache locality and jumping memory accesses, which can negate the benefits of sparsity and sometimes even degrade performance (Wen et al., 2016).

To overcome the limitations of unstructured pruning, structured pruning methods, such as channel-level pruning, have been proposed (He and Xiao, 2024). Notably, the ThiNet framework by Luo et al. (2017) prunes entire filters or channels, maintaining the network's original structure. In another work, Wen et al. (2016) proposed the structured sparsity learning approach, which also includes filter- or channel-wise pruning. This structured approach ensures compatibility with existing deep learning libraries and offers several advantages: it significantly reduces memory footprint, facilitates further compression and acceleration through methods like parameter quantization (Gong et al., 2014; Chen et al., 2015; Wu et al., 2016), and accelerates various vision tasks.

Parallel to these advancements, tropical geometry (Maclagan and Sturmfels, 2021) has emerged as a promising mathematical framework with applications in neural networks and machine learning (Maragos et al., 2021). Tropical geometry, rooted in algebraic geometry, operates over the tropical semiring (Cuninghame-Green, 1994), where conventional addition and multiplication are replaced by min and plus operations, respectively. This transformation turns polynomials into piecewise linear functions, making tropical geometry highly relevant to the study of neural networks with piecewise linear activations like ReLU.

Tropical mathematics encompasses a wide range of applications, which include operations research and scheduling (Cuninghame-Green, 1979), control and optimization (Baccelli et al., 2001; Cohen et al., 2004; Butkovič, 2010; Akian et al., 2012), mathematical physics (Litvinov et al., 2001), speech recognition (Mohri et al., 2002), polyhedral geometry (Gaubert and Katz, 2011), tropical geometry (Maclagan and Sturmfels, 2021), dynamical systems on weighted lattices (Maragos, 2017), sparsity theory on weighted lattices (Tsilivis et al., 2022), matrix factorization (Kordonis et al., 2024), finite state transducers (Theodosis and Maragos, 2018), convex regression (Maragos and Theodosis, 2020), and machine learning (Maragos et al., 2021; Gärtner and Jaggi, 2008). Recently, tropical geometry has been applied to the theoretical study of neural networks. For example, Zhang et al. (2018) demonstrated the equivalence of ReLU-activated neural networks with tropical rational

mappings, and used zonotopes to compute a bound on the number of linear regions of networks, equal to the one in (Montufar et al., 2014). Other works, like those of Charisopoulos and Maragos (2018); Alfarra et al. (2023); Smyrnis and Maragos (2020), have used tropical geometry to compute bounds on the number of linear regions in neural networks and to develop methods for pruning and compressing these networks.

In this paper, we explore the application of tropical geometry to neural network compression. We build on the work of Misiakos et al. (2022), who utilized tropical geometry for structured neural network compression by geometrically approximating the zonotopes corresponding to the network layers. Our contributions include:

- Obtaining a tighter bound for the functional approximation of tropical polynomials than that presented in (Misiakos et al., 2022) using the standard, continuous Hausdorff distance.

- Designing an algorithm that leverages tropical geometry and the Hausdorff distance for the structured compression of neural networks. Our algorithm refines the choice of weights in the compressed network from the clustering done in (Misiakos et al., 2022) and we also provide an iterative technique to further optimize them. Our method is simple to implement and does not depend on the availability of training data samples like other frameworks such as ThiNet (Luo et al., 2017). It is also the first method based on the theory of tropical geometry that compresses convolutional layers.

- Evaluating our algorithm through experiments on MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, and ImageNet datasets. Our method outperforms that of Misiakos et al. (2022) and achieves competitive or superior performance compared to ThiNet, especially in compressing linear layers, and superior performance compared to CUP (Duggal et al., 2021), particularly in the VGG architecture.

This work demonstrates the potential of tropical geometry in enhancing neural network compression techniques, providing a theoretical foundation and a practical algorithm that improves efficiency and maintains accuracy.

2 Related Work

Seminal works have showcased that neural networks tend to be hyperparameterized. Denil et al. (2013) demonstrated that neural networks can be reconstructed using a small subset of parameters, which sparked significant interest in network compression techniques. Blalock et al. (2020) reviewed various methods for reducing model size without compromising performance, highlighting the ongoing advancements in this area.

Early methods for network compression focused on unstructured pruning. LeCun et al. (1989) showed that removing unimportant weights leads to minimal performance loss. This approach was further refined by Han et al. (2015), who proposed pruning weights below a certain threshold followed by fine-tuning the network. Despite their effectiveness, these methods require specialized hardware for efficient inference due to their unstructured nature.

Structured pruning techniques maintain the network's architecture, facilitating better compatibility and efficiency. Wen et al. (2016) introduced structured sparsity learning to

compress networks by finding compact representations of filters and channels, preserving the architecture. Luo et al. (2017) proposed ThiNet, a method that greedily prunes filters based on their impact on the next layer, thereby maintaining network structure and efficiency. Yu et al. (2018) presented the Neuron Importance Score Propagation (NISP) algorithm, which extended the ideas of ThiNet by pruning neurons and channels based on their impact on the final layer output. Lin et al. (2020) introduced HRank, which uses feature map information to rank and prune filters. Duggal et al. (2021) proposed Cluster Pruning (CUP), which prunes similar filters by clustering them based on features derived from both incoming and outgoing weight connections. CUP addresses the limitations of prior work by efficiently determining the ideal number of filters to prune in each layer and integrating pruning into the initial training phase, leading to significant savings in training time and maintaining performance. Wang et al. (2021) introduced a novel approach to pruning convolutional neural networks that focuses on reducing structural redundancy rather than merely pruning individual neurons or channels. Their method identifies and eliminates redundant structures within the network architecture, thereby achieving more efficient pruning without compromising the network's performance. Smyrnis and Maragos (2020) proposed a tropical division algorithm for structured network compression, further enhancing the capabilities of structured pruning techniques. Misiakos et al. (2022) developed Neural Path K-means, which uses clustering techniques based on tropical geometry and functional approximation for structured compression.

Beyond pruning, other state-of-the-art strategies for network compression include parameter quantization and low-rank approximation. Parameter quantization, as explored by Zhou et al. (2017), Zhou et al. (2018) and Han et al. (2015), reduces the number of bits per parameter, significantly lowering memory and computational requirements. Behdin et al. (2023) presented an optimization-based framework for efficient post-training quantization of large language models. Polino et al. (2018) presented quantized distillation, a method that combines quantization with teacher-student distillation. Low-rank approximation (LRA) techniques decompose weight matrices into smaller matrices, reducing parameters and computations. Studies by Denton et al. (2014), Jaderberg et al. (2014), Sindhwani et al. (2015), Ioannou et al. (2016), Tai et al. (2016), and Chen et al. (2018) have shown that LRA can achieve significant speedups and compression with minimal accuracy loss, although it typically requires iterative optimization. These methods can be combined with structured pruning approaches like ThiNet to achieve further model compression and efficiency.

Our framework is a structured pruning method that extends the approach of Misiakos et al. (2022). It prunes networks layer-by-layer, aiming to minimize the error introduced to the input of the next layer. In that sense, it is similar to the ThiNet framework. However, our algorithm is simpler to implement and does not require a training dataset. Our framework leverages clustering and can be naturally extended to non-uniform pruning, drawing parallels to the CUP framework. In fact, it improves upon steps 1, 2, and 3 of CUP. For step 1, we use similar clustering vectors (referred to as filter features by CUP authors Duggal et al. (2021)) for linear layers, and introduce a new method for constructing clustering vectors in convolutional layers. For step 2, we introduce two approaches for choosing the distance threshold of the hierarchical clustering. Our main contribution lies in step 3, where we employ tropical geometry to select a better representative for each cluster, rather than simply choosing the neuron or channel with the most energy. While it is tempting to com-

pare our framework with those of NISP (Yu et al., 2018) and HRank (Lin et al., 2020) given that they are both structured pruning methods, it is important to recognize the distinctions between them and ours. NISP and HRank are advanced frameworks specifically designed to address the limitations of layer-by-layer pruning approaches, such as those employed by ThiNet and our own method. In contrast, our framework leverages a rich theoretical foundation and does not rely on training data samples, offering a unique advantage in certain contexts. We believe our results are promising and present significant advantages. Our work highlights the potential of tropical geometrical techniques and paves the way for further research that could enhance their competitiveness with current state-of-the-art methods.

3 Tropical Geometry of Neural Networks

In this section, we introduce the fundamental background of tropical geometry and its applications to neural network compression.

3.1 Background on Tropical Geometry

Tropical algebra is a field of mathematics that studies the tropical semiring. Tropical geometry is the counterpart of algebraic geometry in the tropical setting. The tropical semiring can refer to either the *min-plus semiring* or the *max-plus semiring* (Butkovič, 2010). In this work, we adhere to the convention of using the *max-plus semiring* $(\mathbb{R}_{\max}, \vee, +)$, defined as the set $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$ equipped with two binary operations: $\vee$ (ordinary max) and $+$ (ordinary sum).

3.1.1 Tropical Polynomials and Rational Functions

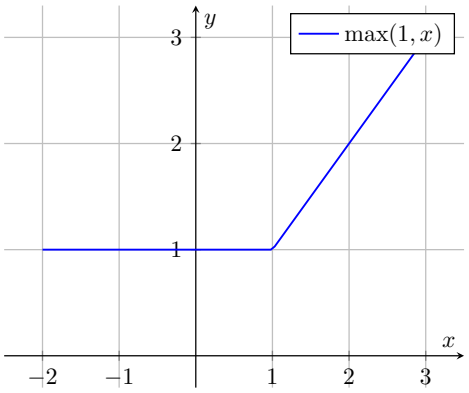
Within the max-plus semiring, we can define polynomials. A *tropical polynomial* f in d variables $\mathbf{x} = (x_1, \ldots, x_d)$ is defined as the function:

$$f(\mathbf{x}) = \bigvee_{i=1}^{n} \left\{ \mathbf{a}_i^T \mathbf{x} + b_i \right\} = \max_{i \in [n]} \left\{ \mathbf{a}_i^T \mathbf{x} + b_i \right\}, \tag{1}$$

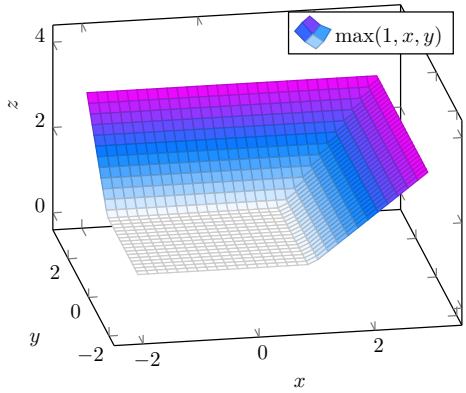
where $[n] := \{1, ..., n\}$. Here, n represents the *rank* of the tropical polynomial. Each monomial term $\{\mathbf{a}_i^T \mathbf{x} + b_i\}$ of the polynomial has an *exponent* or *slope* $\mathbf{a}_i \in \mathbb{R}^d$ and a *coefficient* or *bias* $b_i \in \mathbb{R}$. Each monomial term corresponds to a plane in $\mathbb{R}^{d+1}$. Consequently, tropical polynomials are piecewise linear convex functions. Specifically, every tropical polynomial is a continuous piecewise linear convex function, and every continuous piecewise linear convex function can be expressed (though not uniquely) as a tropical polynomial (Maclagan and Sturmfels, 2021). The set of tropical polynomials in $\mathbf{x}$ defines the semiring $\mathbb{R}_{\max}[\mathbf{x}]$. Figures 1a, 1b illustrate examples of tropical polynomials in one and multiple variables, respectively.

Tropical rational functions extend the concept of rational polynomial functions to tropical algebra. A *tropical rational function* is defined as the tropical multiplication of a tropical polynomial p by the tropical multiplicative inverse q^{-1} of another tropical polynomial q. In conventional arithmetic, this operation corresponds to the difference of two tropical polynomials p and q:

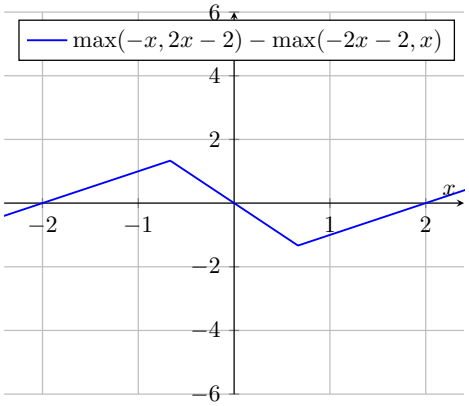
$$r(\mathbf{x}) = p(\mathbf{x}) - q(\mathbf{x})$$



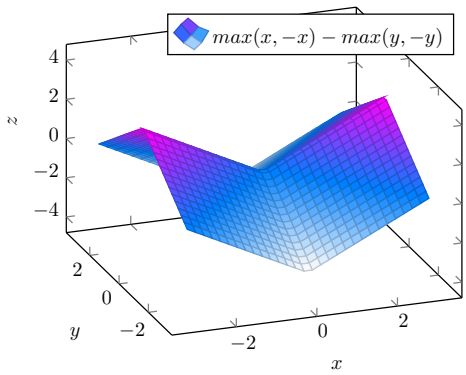
(a) $f(x) = \max\{x, 1\}$



(b) $f(x, y) = \max\{x, y, 1\}$



(c) $f(x) = \max\{2x - 2, -x\} - \max\{-2x - 2, x\}$



(d) $f(x, y) = \max\{x, -x\} - \max\{y, -y\}$

Figure 1: (a) depicts a single-variate tropical polynomial, (b) depicts a multi-variate tropical polynomial, (c) depicts a single-variate tropical rational function, (d) depicts a multi-variate tropical rational function

Tropical rational functions correspond to general piecewise linear functions. Specifically, every tropical rational function is a continuous piecewise linear function, and every continuous piecewise linear function can be expressed (though not uniquely) as a tropical rational function. Figures 1c, 1d provide examples of tropical rational functions.

3.1.2 Newton Polytopes

As with algebraic geometry (Hartshorne, 2013), in tropical geometry we can define the Newton polytope (Monical et al., 2019) of a tropical polynomial. Newton polytopes can be used to analyse the behavior of a polynomial. They connect tropical geometry with polytope theory, an extensively studied field of mathematics (Ziegler, 2012). For a tropical polynomial f as defined in (1), we define its *Newton polytope* as the convex hull of the slopes $\mathbf{a}_i$ of f

$$\text{Newt}(f) := \text{conv}\left\{\mathbf{a}_i, i \in [n]\right\}.$$

Additionally, we define the *extended Newton polytope* of a tropical polynomial f as the convex hull of the slopes $\mathbf{a}_i$ of f extended in their last dimension by the coefficient b_i

$$\text{ENewt}(f) := \text{conv}\left\{(\mathbf{a}_i^T, b_i), i \in [n]\right\}.$$

The following proposition allows us to calculate the (extended) Newton polytope of expressions of tropical polynomials.

Proposition 1 (Zhang et al., 2018**)** *Let $f, g \in \mathbb{R}_{\max}[\mathbf{x}]$ be two tropical polynomials in $\mathbf{x}$. For the extended Newton polytope, the following holds:*

$$\text{ENewt}\left(f \vee g\right) = \text{conv}\left\{\text{ENewt}(f) \cup \text{ENewt}(g)\right\}$$
$$\text{ENewt}(f + g) = \text{ENewt}(f) \oplus \text{ENewt}(g).$$

Here, $\vee$ denotes the tropical addition (taking the maximum), and $+$ denotes the tropical multiplication (ordinary addition). The Minkowski sum $\oplus$ of two polytopes (or more generally subsets of $\mathbb{R}^d$) P and Q is defined as:

$$P \oplus Q := \{p + q \mid p \in P, q \in Q\}.$$

Using the principle of induction, Proposition 1 can be generalized to any finite tropical expression of tropical polynomials.

The extended Newton polytope provides a geometrical interpretation for studying tropical polynomials. The following propositions establish the relationship between tropical polynomials and their extended Newton polytopes. The *upper envelope* or *upper hull* $UF(P)$ of an extended Newton polytope is defined as the set of all points $(\mathbf{a}^T, b)$ of the polytope P that are not "shadowed" by any other part of the polytope when viewed from above (last dimension). This means that there is no $b' > b$ such that $(\mathbf{a}^T, b')$ belongs to P. We have the following useful lemma (proof in Appendix A.1).

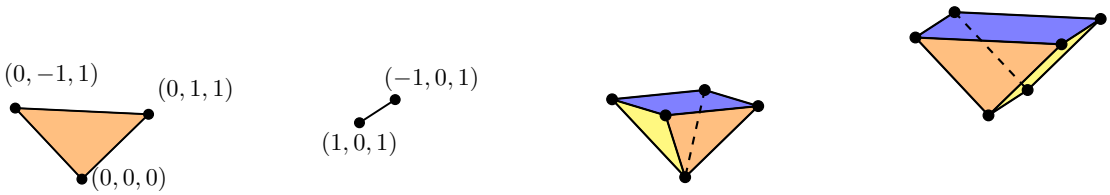
Lemma 2 *Let $p \in \mathbb{R}_{\max}[\mathbf{x}]$ a tropical polynomial in d variables with extended Newton polytope $P = \text{ENewt}(p)$. If $(\mathbf{a}^T, b)$ lies below the upper envelope of P, then $\forall \mathbf{x} \in \mathbb{R}^d, \mathbf{a}^T\mathbf{x} + b \leq p(\mathbf{x})$. The inequality is strict if $(\mathbf{a}^T, b)$ lies strictly below the upper envelope.*

An immediate consequence of the above lemma is the following theorem:

Theorem 3 *Let $f \in \mathbb{R}_{\max}[\mathbf{x}]$ be a tropical polynomial. The values of f are fully determined by the upper envelope $UF(\text{ENewt}(f))$ of its extended Newton polytope.*

Theorem 4 (Charisopoulos and Maragos, 2018**)** *The linear regions of a tropical polynomial $f \in \mathbb{R}_{\max}[\mathbf{x}]$ are in one-to-one correspondence with the vertices of $UF(\text{ENewt}(f))$. In fact, each vertex $(\mathbf{a}_i^T, b_i) \in UF(\text{ENewt}(f))$ gives exactly one linear region D where $f(\mathbf{x}) = \mathbf{a}_i^T\mathbf{x} + b_i$.*

The above theorem highlights the direct relationship between the geometric structure of the extended Newton polytope and the piecewise linear nature of the tropical polynomial. An immideate consequence of Theorems 3 and 4 is the following theorem:



(a) ENewt(f) (b) ENewt(g) (c) ENewt($f \vee g$) (d) ENewt($f + g$)

Figure 2: Operations on tropical polynomials. ENewt($f \vee g$) corresponds to the convex hull of the union of the vertices of the polytopes ENewt(f), ENewt(g). ENewt($f+g$) corresponds to the Minkowski sum of ENewt(f), ENewt(g). For the polytope ENewt($f \vee g$) we illustrate with blue the upper envelope, which consists of a single face. The vertices of the upper envelope are the only non reduntant terms of the polynomial $f \vee g$.

Theorem 5 *For any two tropical polynomials $f, g \in \mathbb{R}_{\max}[\mathbf{x}]$, the following holds:*

$$f = g \Leftrightarrow UF(\text{ENewt}(f)) = UF(\text{ENewt}(g))$$

This implies that two tropical polynomials are functionally identical if and only if their extended Newton polytopes have the same upper envelope.

The above theorems indicate that a tropical polynomial is fully functionally determined by the upper envelope of its extended Newton polytope, as shown by the following example.

Example 1 *Consider the polynomials corresponding to Figure 2.*

$$f(x, y) = \max\{0, -y + 1, y + 1\}$$

$$g(x, y) = \max\{x + 1, -x + 1\}.$$

We have that

$$(f \vee g)(x, y) = \max\{0, x + 1, y + 1, -y + 1, -x + 1\}$$

$$(f + g)(x, y) = \max\{x + 1, -x + 1, x - y + 2, -x - y + 2, x + y + 2, -x + y + 2\}.$$

The extended Newton polytopes of $f, g, f \vee g, f + g$ are shown in Figure 2. The polynomial $f \vee g$ can be reduced as follows:

$$(f \vee g)(x, y) = \max\{x + 1, y + 1, -y + 1, -x + 1\},$$

which corresponds to the vertices of the upper envelope of ENewt($f \vee g$).

3.1.3 Zonotopes

In polytope theory, *zonotopes* are a special class of convex polytopes that can be defined as the Minkowski sum of a finite set of line segments (or edges). Formally, given a set of line segments $g_1, \ldots, g_n$, the zonotope is defined as

$$Z := \bigoplus_{i \in [n]} g_i$$

The line segments g_i are referred to as the zonotope's *generators*. Each generator contributes to the shape and size of the zonotope, which is formed by the space swept out by these line segments.

Alternatively, a zonotope can be expressed equivalently by a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^d$ and a starting point $\mathbf{s} \in \mathbb{R}^d$. By taking the generators to be the segments $[\mathbf{0}, \mathbf{v}_1], \ldots, [\mathbf{0}, \mathbf{v}_n]$ and translating the first segment by $\mathbf{s}$, we obtain the equivalent form:

$$Z = \left\{ \mathbf{s} + \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mid 0 \leq \lambda_i \leq 1 \right\}$$

In this context, the vectors $\mathbf{v}_i$ are sometimes referred to as the zonotope's generators, meaning the segments $[\mathbf{0}, \mathbf{v}_i]$. When the starting point $\mathbf{s}$ is not mentioned, it is assumed to be the origin $\mathbf{0}$.

For a zonotope with a starting point $\mathbf{s} \in \mathbb{R}^d$ and generators $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^d$, a vertex $\mathbf{u} \in V_Z$ corresponds to points where $\lambda_i = 0$ or 1. The vertex $\mathbf{u}$ can be expressed as:

$$\mathbf{u} = \mathbf{s} + \sum_{i \in I} \mathbf{v}_i,$$

where $I \subseteq [n]$. It can be shown that zonotopes have up to exponential many verticesIn fact, for generators in general positions they have exactly $2\sum_{j=0}^{d-1} nCr(n-1, j)$ vertices (Gritzmann and Sturmfels, 1993).

Zonotopes exhibit several interesting properties. Notably, they are centrally symmetric. This symmetry, along with their convex structure, make zonotopes particularly useful in various fields, including optimization (Bern and Eppstein, 2001) and computational geometry (Ziegler, 2012).

3.2 Neural Networks with Piecewise Linear Activations

Tropical geometry provides a mathematical framework for analyzing neural networks with piecewise linear activation functions. Leveraging the properties of tropical polynomials and Newton polytopes, it is used to model the behavior of various activation functions commonly used in neural networks. In this work, we focus on ReLU-activated networks. In this section, we outline the basic tropical geometrical properties of ReLU-activated networks and establish our notations.

ReLU Activations. Consider a network which consists of an input layer $\mathbf{x} = (x_1, \ldots, x_d)$, a hidden layer $\mathbf{f} = (f_1, \ldots, f_n)$ of ReLU units, and an output layer $\mathbf{v} = (v_1, \ldots, v_m)$. The input, hidden, and output layers are connected through linear transformations represented by matrices $\mathbf{A}$ and $\mathbf{C}$. Each neuron i has input weights and bias given by $\mathbf{A}_{i,:} = (\mathbf{a}_i^T, b_i)$ and output weights $\mathbf{C}_{:,i}^T = (c_{1i}, \ldots, c_{mi})$. We assume the output layer has no bias. Such a network is depicted in Figure 3.

The output of the ReLU unit i is given by:

$$f_i(\mathbf{x}) = \mathrm{ReLU}(\mathbf{a}_i^T \mathbf{x} + b_i) = \max\{\mathbf{a}_i^T \mathbf{x} + b_i, 0\}.$$

This expression represents a tropical polynomial of rank 2, with one term being the constant 0. The extended Newton polytope $\mathrm{ENewt}(f_i)$ of f_i is an edge with one endpoint at the origin

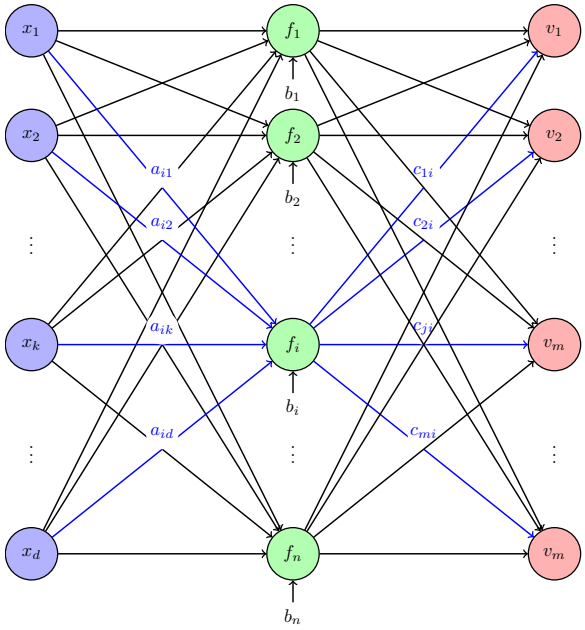


Figure 3: Neural network with one hidden ReLU layer. The first linear layer has weights $\{\mathbf{a}_i^T\}$ with bias $\{b_i\}$ corresponding to i-th node $\forall i \in [n]$ and the second has weights $\{c_{ji}\}$, $\forall j \in [m], i \in [n]$

$\mathbf{0}$ and the other endpoint at $(\mathbf{a}_i^T, b_i)$. The j-th component of the output layer v_j can be computed as follows:

$$v_j = \sum_{i\in[n]} c_{ji} f_i = \sum_{i:c_{ji}>0} |c_{ji}| f_i - \sum_{i:c_{ji}<0} |c_{ji}| f_i = p_j - q_j.$$

In the above expression, $|c_{ji}|f_i$ are tropical polynomials. Thus, p_j and q_j are tropical polynomials formed by the addition of tropical polynomials. Consequently, v_j is a tropical rational function. We call p_j the *positive polynomial* and q_j the *negative polynomial* of v_j. This result can be extended to deeper networks, as suggested by the following proposition:

Theorem 6 (Zhang et al., 2018) *A ReLU-activated deep neural network* $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^m$ *is a tropical rational mapping (vector whose elements are tropical rational functions).*

The extended Newton polytope of $|c_{ji}|f_i$ is an edge with one endpoint at the origin $\mathbf{0}$ and the other at $|c_{ji}|(\mathbf{a}_i^T, b_i)$. The extended Newton polytope P_j of p_j is the Minkowski sum of the *positive* generators $\{|c_{ji}|(\mathbf{a}_i^T, b_i) : c_{ji} > 0\}$, and the polytope Q_j of q_j is the Minkowski sum of the *negative* generators $\{|c_{ji}|(\mathbf{a}_i^T, b_i) : c_{ji} < 0\}$. Thus, P_j, Q_j are zonotopes. We refer to P_j as the *positive zonotope* and Q_j as the *negative zonotope* of v_j.

4 Approximation based on Hausdorff distance

In this section, we present our improved theorem, which uses the Hausdorff distance in its standard continuous form to bound the error between the original and approximate neural

networks. Misiakos et al. (2022) derived a bound for the error of approximating the tropical polynomials that represent a neural network. The motivation for geometrically bounding the error of these polynomials stems from Theorem 5. This theorem indicates that two polynomials with the same extended Newton polytopes are functionally equivalent. Consequently, it is expected that two tropical polynomials with approximately equal extended Newton polytopes will attain similar values.

The metric used by Misiakos et al. (2022) to define the distance between extended Newton polytopes is a discrete form of the Hausdorff distance. In contrast, we extend their result by employing the standard continuous Hausdorff distance between two polytopes to obtain a tighter bound. As a convention we consider the definition of polytopes as convex and including their interior. Moreover, the distance between two points $\mathbf{u}$ and $\mathbf{v}$ is denoted as $\text{dist}(\mathbf{u}, \mathbf{v}) := \|\mathbf{u} - \mathbf{v}\|$, where $\|\cdot\|$ denotes the standard L^2 Euclidean norm. The distance between a point $\mathbf{u}$ and a set V is defined as $\text{dist}(\mathbf{u}, V) = \text{dist}(V, \mathbf{u}) := \inf_{\mathbf{v}\in V} \|\mathbf{u} - \mathbf{v}\|$. We proceed by providing the definition of the Hausdorff distance.

Definition 7 (Hausdorff distance) *Let $S, \tilde{S}$ be two subsets of $\mathbb{R}^d$. The Hausdorff distance $H(S, \tilde{S})$ of the two sets is defined as*

$$H(S, \tilde{S}) := \max\left\{\sup_{\mathbf{u}\in S} \text{dist}(\mathbf{u}, \tilde{S}), \sup_{\mathbf{v}\in \tilde{S}} \text{dist}(S, \mathbf{v})\right\}$$

In the case of polytopes, the following lemma (proof in Appendix A.2) reassures us that the suprema in the above expression are attained, and in fact by points in the vertex sets $V_P, V_{\tilde{P}}$ of the polytopes.

Lemma 8 *Due to the convexity and compactness of polytopes, we have that*

$$H(P, \tilde{P}) = \max\left\{\max_{\mathbf{u}\in V_P} \text{dist}(\mathbf{u}, \tilde{P}), \max_{\mathbf{v}\in V_{\tilde{P}}} \text{dist}(P, \mathbf{v})\right\}$$

In their work, Misiakos et al. (2022) used the discrete form of the Hausdorff distance, defined as the Hausdorff distance of the vertex sets of the two polytopes ($DH(P, \tilde{P}) := H(V_P, V_{\tilde{P}})$), to obtain the following result.

Proposition 9 (Misiakos et al., 2022**)** *Let $p, \tilde{p} \in \mathbb{R}_{\max}[\mathbf{x}]$ be two tropical polynomials with extended Newton polytopes $P = \text{ENewt}(p)$ and $\tilde{P} = \text{ENewt}(\tilde{p})$. Then,*

$$\frac{1}{\rho} \max_{\mathbf{x}\in B} |p(\mathbf{x}) - \tilde{p}(\mathbf{x})| \leq DH(P, \tilde{P})$$

where $B = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| \leq r\}$ and $\rho = \sqrt{r^2 + 1}$.

We extend the result of Misiakos et al. (2022). The proof is provided in the appendix.

Proposition 10 *Let $p, \tilde{p} \in \mathbb{R}_{\max}[\mathbf{x}]$ be two tropical polynomials with extended Newton polytopes $P = \text{ENewt}(p)$ and $\tilde{P} = \text{ENewt}(\tilde{p})$. Then,*

$$\frac{1}{\rho} \max_{\mathbf{x}\in B} |p(\mathbf{x}) - \tilde{p}(\mathbf{x})| \leq H(P, \tilde{P})$$

where $B = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| \leq r\}$ and $\rho = \sqrt{r^2 + 1}$.

Since $V_P \subseteq P, V_{\tilde{P}} \subseteq \tilde{P}$, we have that $\text{dist}(\mathbf{u}, \tilde{P}) \leq \text{dist}(\mathbf{u}, V_{\tilde{P}}), \text{dist}(P, \mathbf{v}) \leq \text{dist}(V_P, \mathbf{v})$. The equality in the aforementioned inequalities is achieved only in special cases and thus the bound $H(P, \tilde{P})$ we provide is in general tighter than $DH(P, \tilde{P})$ from Misiakos et al. (2022). We demonstrate one example where the two formulations give equal distances and one where the standard Hausdorff distance is strictly smaller than the discrete one.

Example 2 *Let P be the zonotope formed by taking the generators $(1, 0)$ and $(\cos 60^\circ = \frac{1}{2}, \sin 60^\circ = \frac{\sqrt{3}}{2})$. Form zonotope $\tilde{P}$ by taking as a generator the mean of the two generators of P, which is the vector $(\frac{3}{4}, \frac{2+\sqrt{3}}{4})$. Refer to Figure 4a for a visual representation.*
The discrete Hausdorff distance between the two polytopes is given by:

$$DH(P, \tilde{P}) = \max\left\{\frac{\sqrt{3}}{2}, \frac{1}{2}\right\} = \frac{\sqrt{3}}{2}.$$

In this case, the Hausdorff distance is equal to the discrete one:

$$H(P, \tilde{P}) = \max\left\{\frac{\sqrt{3}}{2}, \sin 30^\circ \cdot |\textit{generator of } \tilde{P}|\right\}$$

$$= \max\left\{\frac{\sqrt{3}}{2}, \frac{1}{2} \cdot \cos 30^\circ = \frac{\sqrt{3}}{4}\right\} = \frac{\sqrt{3}}{2}.$$

Example 3 *Let zonotope P be formed by taking the generators $(1, 0)$ and $(\cos 60^\circ = \frac{1}{2}, \sin 60^\circ = \frac{\sqrt{3}}{2})$. Form zonotope $\tilde{P}$ by taking as a generator the sum of the two generators, which results in the vector $(\frac{3}{2}, \frac{2+\sqrt{3}}{2})$. Refer to Figure 4b for a visual representation.*
The discrete Hausdorff distance between the two polytopes is calculated as:

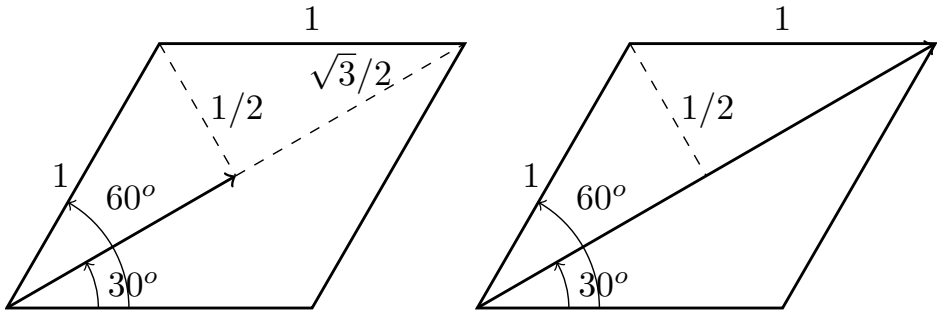
$$DH(P, \tilde{P}) = \max\{0, 1\} = 1.$$

Here, the Hausdorff distance is strictly smaller and equal to:

$$H(P, \tilde{P}) = \max\left\{0, \frac{1}{2}\right\} = \frac{1}{2}.$$

Notice that if our goal were to approximate the polytope P by the polytope $\tilde{P}$, then according to the discrete Hausdorff distance, taking the mean is the optimal strategy and gives distance $\sqrt{3}/2$ (example 2), better than 1 (example 3). On the other hand, according to the Hausdorff distance in its standard form, taking the sum is the optimal strategy and gives distance $1/2$ (example 2), better than $\sqrt{3}/2$ (example 3). This implies that an algorithm based on the Hausdorff distance would imply a better approximation of P in the above example.

We derive the following result, which differs from (Misiakos et al., 2022) by using the standard form of the Hausdorff distance and thus providing a tighter bound. For the proof we apply the triangle inequality and then repeatedly Proposition 10 to each output of the network. The full proofs of Proposition 10 and Theorem 11 are provided in Appendix A.3.



(a) Example 2 (b) Example 3

Figure 4: Polytopes P, $\tilde{P}$ of examples 2, 3

Theorem 11 *Let v and $\tilde{v}$ be the outputs of two neural networks as in Figure 3. Then, the following inequality holds:*

$$\frac{1}{\rho}\max_{\mathbf{x}\in B}\|v(\mathbf{x})-\tilde{v}(\mathbf{x})\|_1 \leq \sum_{j=1}^{m}\Big(H(P_j,\tilde{P}_j)+H(Q_j,\tilde{Q}_j)\Big),$$

where $\|\cdot\|_1$ denotes the L^1 norm.

5 Compression Algorithm

In this section we present our algorithm **TropNNC** for neural network compression. Our method, based on Theorem 11, is a structured neural network compression method that reduces the number of neurons in the hidden layers and produces a compressed network that is a functional approximation of the original. We differentiate between single-output and multi-output networks and introduce novel approaches for both scenarios. The main achievement of our method is to preserve the neural network as a function, achieving performance comparable to state-of-the-art frameworks, all without relying on training data samples.

5.1 Single Output

First, we focus on single output networks, as the one depicted in Figure 3, with $m = 1$. The network has weight matrices $\mathbf{A} \in \mathbb{R}^{n\times(d+1)}$, $\mathbf{C} \in \mathbb{R}^{1\times n}$, and a single output v with corresponding positive and negative zonotopes P and Q. We aim to approximate this network with a smaller one that has fewer neurons in the hidden layer, say $K < n$, resulting in a new hidden layer $\tilde{\mathbf{f}} = (\tilde{f}_1, \ldots, \tilde{f}_K)$ with new weight matrices $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{C}}$. Our goal is to ensure that $\tilde{v}(\mathbf{x}) \approx v(\mathbf{x})$ for all $\mathbf{x} \in B$. According to Theorem 11, it is sufficient to choose the new weight matrices $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{C}}$ such that $\widetilde{P} \approx P$ and $\widetilde{Q} \approx Q$, where the approximation relation is in terms of the Hausdorff distance between the zonotopes.

In the original network with n hidden neurons, each neuron contributes a generator $|c_i|(\mathbf{a}_i^T, b_i)$, which can be either positive or negative. In the approximating network with K hidden neurons, each neuron contributes a generator $|\tilde{c}_i|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$. We aim to pick generators

$|\tilde{c}_i|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$ such that the generated positive and negative zonotopes $\widetilde{P}$ and $\widetilde{Q}$ approximate the positive and negative zonotopes P and Q, respectively.

In essence, we have translated the neural network approximation problem into a *zonotope approximation problem*—specifically, the task of approximating a zonotope with another zonotope that has fewer generators. This problem is known in the literature as *zonotope order reduction* (Yang and Scott, 2018). In our case, the approximation must happen in terms of the Hausdorff distance.

5.1.1 Algorithm for single output network

Misiakos et al. (2022) proposed the algorithm "Zonotope K-means" for compressing single-output neural networks. Their approach uses K-means clustering on the set of positive and negative generators, and replaces the generators of each formed cluster by a single representative, the center of the cluster (i.e. the mean of the generators of the cluster).

In contrast, our approach also uses K-means to cluster the generators, but instead of taking the representative to be the center of the cluster, we take the representative to be the sum of the generators of the cluster. We show later that this compression produces a tighter approximation of the neural network compared to (Misiakos et al., 2022). We demonstrate this with Example 4 and prove this formally with Proposition 12 and Corollary 13. Our Algorithm 1 is depicted below.

Algorithm 1 TropNNC, Single output

1: Split the generators $|c_i|(\mathbf{a}_i^T, b_i)$ into positive and negative generators:

$$\{|c_i|(\mathbf{a}_i^T, b_i) : c_i > 0\},$$
$$\{|c_i|(\mathbf{a}_i^T, b_i) : c_i < 0\}.$$

2: Execute K-means with $K/2$ centers on the positive generators $\{|c_i|(\mathbf{a}_i^T, b_i) : c_i > 0\}$, and $K/2$ centers on the negative generators $\{|c_i|(\mathbf{a}_i^T, b_i) : c_i < 0\}$.

3: Obtain positive and negative cluster representatives:

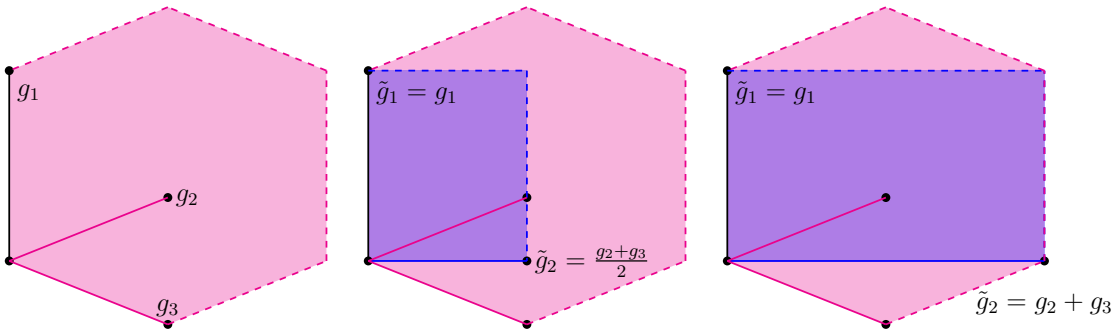
$$\{|\tilde{c}_i|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i) : i \in C_+\},$$
$$\{|\tilde{c}_i|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i) : i \in C_-\},$$

where $C_+ \sqcup C_- = [K]$ and $|\tilde{c}_i|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$ is the **sum of the generators** of cluster i.

4: For each $i \in [K]$ construct a (hidden layer) neuron with input weights and bias $|\tilde{c}_i|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$.

5: For each constructed neuron i set the output weight to 1 if the neuron corresponds to a representative of a positive cluster ($i \in C_+$), otherwise set it to -1.

Example 4 *Suppose we have a positive zonotope consisting of 3 positive generators g_1, g_2, g_3 like the pink zonotope (hexagon) in Figure 5a, and suppose we have $K/2 = 2$. The Zonotope K-means algorithm of Misiakos et al. (2022) will form 2 clusters of generators, the black cluster which consists of a single generator g_1, and the pink cluster which consists of the*



(a) Original Zonotope (b) Zonotope K-means (c) TropNNC

Figure 5: Example execution of Zonotope K-means and TropNNC

pink generators g_2, g_3. Zonotope K-means will take as cluster representatives the centers of the clusters. In our example, this will be the black generator $\tilde{g}_1 = g_1$ for the black cluster, and the blue generator $\tilde{g}_2 = \frac{g_2+g_3}{2}$ for the pink cluster, as in Figure 5b. The blue zonotope is the zonotope generated by the representatives of the clusters (black and blue generators $\tilde{g}_1, \tilde{g}_2$ respectively), and it is the approximation that Zonotope K-means uses to approximate the pink zonotope.

In contrast, TropNNC will also form the same 2 clusters of generators, but instead of taking the center, it will take the sum of the generators of each cluster as the cluster representative. In our example, these will be the black generator $\tilde{g}_1 = g_1$ and the blue generator $\tilde{g}_2 = g_2 + g_3$ of Figure 5c, which form a different blue zonotope which is used as the approximation of the pink zonotope.

In this example, one can visually verify that according to the Hausdorff distance, the approximation of TropNNC is better than the approximation of Zonotope K-Means.

The result of Example 4 can be generalized. Indeed, the following bound and its corollary hold (proofs in Appendix A.4 and A.5).

Proposition 12 *Suppose the clusters K are enough so that for every cluster, no 2 generators of the cluster form an obtuse angle. Then, single-output TropNNC produces a neural network with output $\tilde{v}$ satisfying:*

$$\frac{1}{\rho} \max_{\mathbf{x} \in B} |u(\mathbf{x}) - \tilde{v}(\mathbf{x})| \leq \sum_{i \in I} \min\{||c_i(\mathbf{a}_i^T, b_i)||, \delta_{max}\},$$

where δ_{max} is the largest distance from a point to its corresponding cluster center.

Corollary 13 *The above bound is tighter than the bound of Zonotope K-means of Misiakos et al. (2022).*

5.2 Multi-output

Now we consider the case of multi-output networks as in Figure 3, for general $m \in \mathbb{N}$. We consider a network with output $\mathbf{v} = (v_1, \ldots, v_m)$ and corresponding positive and negative

zonotopes P_j and Q_j for each output node v_j. An interesting property of these polytopes is that they share the directions $(\mathbf{a}_i^T, b_i)$ of their generators. For instance, output v_1 might have a positive generator $|c_{1i}|(\mathbf{a}_i^T, b_i)$ of zonotope P_1, while output v_2 might have a negative generator $|c_{2i}|(\mathbf{a}_i^T, b_i)$ of zonotope Q_2. These generators are parallel to each other, with common direction $(\mathbf{a}_i^T, b_i)$.

As before, we aim to approximate this network with a smaller network that has fewer neurons in the hidden layer, say $K < n$, resulting in a new hidden layer $\tilde{\mathbf{f}} = (\tilde{f}_1, \ldots, \tilde{f}_K)$ with new weight matrices $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{C}}$. Our goal is to ensure that $\tilde{\mathbf{v}}(\mathbf{x}) \approx \mathbf{v}(\mathbf{x})$ for all $\mathbf{x} \in B$. As we have seen, this can be translated to having $\widetilde{P}_j \approx P_j$ and $\widetilde{Q}_j \approx Q_j$, where the approximation relation is in terms of the Hausdorff distance between the zonotopes.

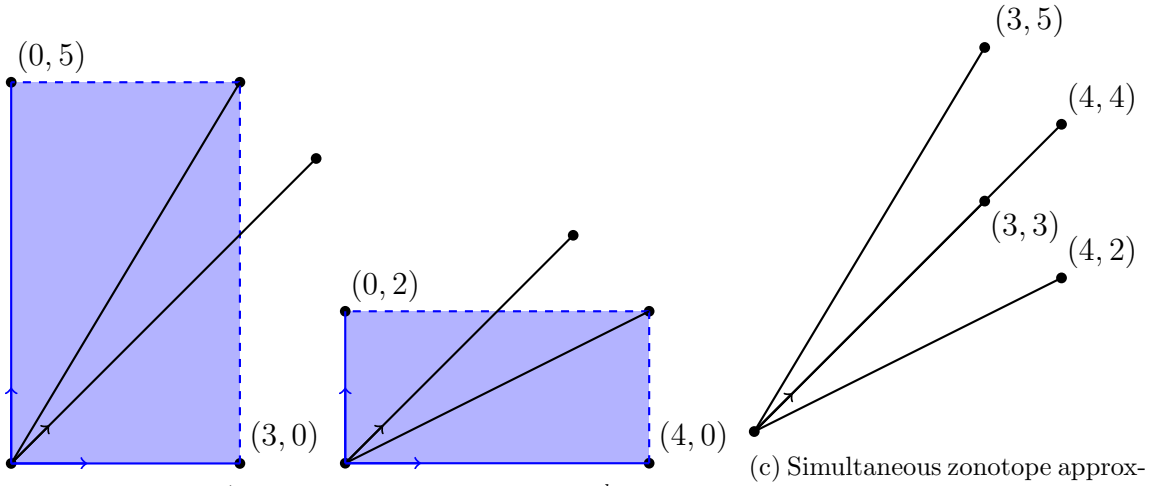
Notice that for each new neuron $\tilde{f}_i$, its input weight-bias $(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$ defines a generator direction for every output $\tilde{v}_j$, where $j \in [m]$. For this generator direction, the magnitudes of the generators change across different outputs by adjusting the output weight $\tilde{c}_{ji}$ for each $j \in [m]$, resulting in parallel generators for each output: $|\tilde{c}_{1i}|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i), \ldots, |\tilde{c}_{mj}|(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$. These generators can be either positive or negative according to the sign of $\tilde{c}_{ji}$.

Thus, we start with original positive and negative zonotopes, with parallel generators, and aim to approximate them with new positive and negative zonotopes, also with parallel generators. We refer to this complex approximation problem as *simultaneous zonotope approximation*.

Example 5 *Suppose we have a neural network with a single hidden layer, as in Figure 3, with dimensions* $d + 1 = n = m = 2$. *Consider input weights* $(\mathbf{a}_1^T, b_1) = (1, 0), (\mathbf{a}_2^T, b_2) = (0, 1)$ *and output weights* $c_{11} = 3, c_{12} = 5, c_{21} = 4, c_{22} = 2$. *In this example, for simplicity we took all output weights to be positive so that we only deal with positive zonotopes. The zonotopes of the two outputs will be two parallelograms with parallel edges, as illustrated in the Figure 6. The zonotope of the first output is generated by* $c_{11}(\mathbf{a}_1^T, b_1) = 3(1, 0) = (3, 0)$ *and* $c_{12}(\mathbf{a}_2^T, b_2) = 5(0, 1) = (0, 5)$, *and of the second output by* $c_{21}(\mathbf{a}_1^T, b_1) = 4(1, 0) = (4, 0)$ *and* $c_{22}(\mathbf{a}_2^T, b_2) = 2(0, 1) = (0, 2)$. *Say we want to reduce the hidden neurons to* $K = 1, \tilde{\mathbf{f}} = (\tilde{f}_1)$. *If we could approximate each output's zonotope separately, we could simply apply the single output algorithm and approximate each parallelogram by its diagonal. However, these diagonals are not parallel to each other, and thus can not occur by a single hidden neuron* $\tilde{f}_1$ *with input weights* $(\tilde{\mathbf{a}}_1^T, \tilde{b}_1)$. *Instead, we have to choose a single common direction* $(\tilde{\mathbf{a}}_1^T, \tilde{b}_1)$ *for both output zonotopes. We can however choose a different magnitude for each output along this common direction. As will be presented in the algorithm below, for the common direction we choose the vector* $(\tilde{\mathbf{a}}_i^T, \tilde{b}_i) = \frac{(\mathbf{a}_1^T, b_1) + (\mathbf{a}_2^T, b_2)}{2} = (0.5, 0.5)$. *For each output* j, $\tilde{c}_{j1}$ *is chosen so that the edge is as close to the diagonal as possible. Specifically, we choose* $\tilde{c}_{11} = 3 + 5 = 8$ *and* $\tilde{c}_{21} = 2 + 4 = 6$. *The approximation procedure can be seen in Figure 6c.*

5.2.1 Non-iterative Algorithm for multi-output network

To tackle this problem of simultaneous zonotope approximation, Misiakos et al. (2022) proposed the "Neural Path K-means" algorithm. Their approach to the problem is to use K-means clustering on the set of vectors $\left\{(\mathbf{a}_i^T, b_i, \mathbf{C}_{:,i}^T), i \in [n]\right\}$, and replace the vectors of each formed cluster by a single representative, the center of the cluster (i.e. the mean of the vectors of the cluster). Our approach also uses K-means to cluster the same vectors, but



(a) Zonotope of 1^{st} output (b) Zonotope of 2^{nd} output (c) Simultaneous zonotope approximation

Figure 6: Example of simultaneous zonotope approximation for a network with 2 outputs and 2 hidden neurons

instead of taking the representative to be the center of the cluster, we form the representative using the following heuristic:

Suppose you have 2 neurons with input weights $(\mathbf{a}_1^T, b_1), (\mathbf{a}_2^T, b_2)$ and output weights $\mathbf{C}_{:,1}^T, \mathbf{C}_{:,2}^T$, that get clustered together. If $(\mathbf{a}_1^T, b_1) \approx (\mathbf{a}_2^T, b_2) \approx \frac{(\mathbf{a}_1^T, b_1)+(\mathbf{a}_2^T, b_2)}{2}$, then for every output j the following holds:

$$C_{j,1}(\mathbf{a}_1^T, b_1) + C_{j,2}(\mathbf{a}_2^T, b_2) \approx (C_{j,1} + C_{j,2})\frac{(\mathbf{a}_1^T, b_1) + (\mathbf{a}_2^T, b_2)}{2}.$$

Thus, in order to approximate the 2 neurons, it suffices to take the mean of their input weights and the sum of their output weights. This heuristic can also be motivated by viewing at the network: it corresponds to taking the mean in terms of the input activations of the two neurons, and taking the sum as the joint output activations of the two neurons.

We generalize the above to clusters with more neurons. For every cluster $k \in [K]$ with clustered neuron indexes I_k and vectors $\left\{(\mathbf{a}_i^T, b_i, \mathbf{C}_{:,i}^T), i \in I_k\right\}$ take $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ to be the mean of $\left\{(\mathbf{a}_i^T, b_i), i \in I_k\right\}$, take $\widetilde{\mathbf{C}}_{:,k}^T$ to be the sum of $\left\{\mathbf{C}_{:,i}^T, i \in I_k\right\}$, and form the representative of the cluster $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k, \widetilde{\mathbf{C}}_{:,k}^T)$. This leads to Algorithm 2.

There are several variations of Algorithm 2. For instance, one could use the sum instead of the mean for the input weights. However, in our experiments, this approach yielded worse results. Another variation involves clustering neurons based on cosine similarity rather than the L^2 distance of the vectors, recognizing that neurons with parallel input weights can be clustered together regardless of their output weights. Additionally, one could normalize the vectors based on their input weights by shifting the norm to the output weights before applying K-means. Despite testing these methods, none provided significant advantages. Therefore, we simply recommend Algorithm 2.

Algorithm 2 (Non-iterative) TropNNC for Multi-output networks

1: Execute KMeans with K centers on the vectors $(\mathbf{a}_i^T, b_i, \mathbf{C}_{:,i}^T)$ for $i \in [n]$, forming K clusters $\{(\mathbf{a}_i^T, b_i, \mathbf{C}_{:,i}^T) \mid i \in I_k\}$ for $k \in [K]$.

2: **For each** $k \in [K]$**, form the cluster representative** $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k, \widetilde{\mathbf{C}}_{:,k}^T)$ **as follows**:

(i) Compute $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ as the **mean of the input weights and biases** of the vectors in the cluster, i.e., the mean of the set $\{(\mathbf{a}_i^T, b_i) \mid i \in I_k\}$.

(ii) Compute $\widetilde{\mathbf{C}}_{:,k}^T$ as the **sum of the output weights** of the vectors in the cluster, i.e., the sum of the set $\{\mathbf{C}_{:,i}^T \mid i \in I_k\}$.

3: Construct the new hidden layer:

(i) For the input weights, set the k-th row of the weight-bias matrix to $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$.

(ii) For the output weights, set the k-th column to $\widetilde{\mathbf{C}}_{:,k}$.

5.2.2 Iterative Algorithm for multi-output network

To improve the approximation of Algorithm 2, we again make use of tropical geometry. Specifically, we formulate an optimization problem that takes the output of Algorithm 2 and with an iterative process produces weights that achieve a better simultaneous zonotope approximation.

Motivated by Algorithm 1, assuming the number of null neurons are few (see Misiakos et al. (2022) and Appendix A.6), we wish in terms of every output j the cluster representative $\widetilde{C}_{j,k}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ to be as close to the cluster sum $\sum_{i \in I_k} C_{j,i}(\mathbf{a}_i^T, b_i)$ as possible. Thus, for every cluster k, we have unknowns $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k, \widetilde{\mathbf{C}}_{:,k}^T) = (\tilde{a}_{k1}, \ldots, \tilde{a}_{kd}, \tilde{b}_k, \widetilde{C}_{1k}, \ldots)$, and we wish to find a solution which minimizes the following criterion:

$$\sum_{j=1}^{m} \left\| \tilde{C}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) - \sum_{i \in I_k} C_{ji}(\mathbf{a}_i^T, b_i) \right\|^2,$$

where m is the number of outputs, and I_k is the set of neurons of cluster k.

The above optimization problem can be solved by means of iterative alternating minimization. Specifically:

1. Holding the input weights $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ constant and minimizing with respect to $\widetilde{\mathbf{C}}_{:,k}$, we notice that the terms of the sum are independent and thus can be minimized separately. For each term of the sum, the minimization occurs if we project the sum $\sum_{i \in I_k} C_{ji}(\mathbf{a}_i^T, b_i)$ onto $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$. We have:

$$\widetilde{C}_{jk} = \frac{\left\langle \sum_{i \in I_k} C_{ji}(\mathbf{a}_i^T, b_i), (\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\rangle}{\left\| (\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\|^2}$$

2. Holding the output weights $\widetilde{\mathbf{C}}_{:,k}$ constant, and minimizing with respect to the input weights $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$, we take the derivative of the criterion with respect to $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ and set it to zero. We have:

$$2\sum_{j=1}^{m}\left[\widetilde{C}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) - \sum_{i\in I_k} C_{ji}(\mathbf{a}_i^T, b_i)\right]\widetilde{C}_{jk} = 0 \Leftrightarrow$$

$$\Leftrightarrow (\tilde{\mathbf{a}}_k^T, \tilde{b}_k)\sum_{j=1}^{m}\widetilde{C}_{jk}^2 = \sum_{j=1}^{m}\sum_{i\in I_k} C_{ji}(\mathbf{a}_i^T, b_i)\widetilde{C}_{jk} \Leftrightarrow$$

$$(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) = \frac{\sum_{j=1}^{m}\sum_{i\in I_k} C_{ji}(\mathbf{a}_i^T, b_i)\widetilde{C}_{jk}}{\sum_{j=1}^{m}\widetilde{C}_{jk}^2}$$

We iterate over the above alternating steps for num_iter iterations. Additionally, we can initialize the iteration with the representative obtained from Algorithm 2, which should provide a good starting point. This initialization is expected to reduce the number of required epochs and speed up the compression algorithm. The resulting procedure is detailed in Algorithm 3. We should note that, in practise, the number of null generators is not negligible, and thus our optimization criterion also constitutes a heuristic method. For this reason, the number of iterations should not be excessive. We provide the following bound for the approximation error of our algorithm (proof in Appendix A.6).

Proposition 14 *Suppose the clusters K are enough so that for every cluster, no two $(\mathbf{a}_i^T, b_i), (\mathbf{a}_{i'}^T, b_{i'})$ of the cluster form an obtuse angle. Then, a variant (see Appendix) of Iterative TropNNC produces a neural network with output $\tilde{v}$ satisfying:*

$$\frac{1}{\rho}\max_{\mathbf{x}\in B}\|v(\mathbf{x}) - \tilde{v}(\mathbf{x})\|_1 \leq \sqrt{m}\sum_{i=1}^{n}\min\left\{\|\mathbf{C}_{:,i}\|\|(\mathbf{a}_i^T, b_i)\|, \frac{l_{k(i)}}{N_{min}} + \|\epsilon_{:,i}\|_F\right\} + \sum_{j=1}^{m}\sum_{i\in N_j}|c_{ji}|\|(\mathbf{a}_i^T, b_i)\|$$

where:

- N_j *is the set of null neurons with respect to output* j*.*
- $k(i)$ *is the cluster of neuron* i*.*
- N_{min} *is the minimum cardinality of the non-null generators of a cluster.*
- l_k *is the objective value of the optimization criterion for cluster* k*.*
- $\epsilon_{j,i}$ *is the difference/error between* $c_{ji}(\mathbf{a}_i^T, b_i)$ *and the cluster mean* $\frac{\sum_{i\in I_{jk}} c_{ji}(\mathbf{a}_i^T, b_i)}{|I_{jk}|}$*.*
- $\epsilon_{:,i} = [\epsilon_{1,i}, \ldots, \epsilon_{m,i}]$*.*
- $\|\cdot\|_F$ *is the Frobenius norm.*

Algorithm 3 Iterative TropNNC for Multi-output networks

Execute K-means with K centers on the vectors $(\mathbf{a}_i^T, b_i, \mathbf{C}_{:,i}^T), i \in [n]$, and form K clusters $\{(\mathbf{a}_i^T, b_i, \mathbf{C}_{:,i}^T), i \in I_k\}, k \in [K]$
for $k \in [K]$ **do**
  Form the initial cluster representative $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k, \widetilde{\mathbf{C}}_{:,k}^T)$ as follows:
    2i. Take $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ to be **the mean of the input weights + bias of the vectors of the cluster**, i.e., the mean of the set of vectors $\{(\mathbf{a}_i^T, b_i), i \in I_k\}$
    2ii. Take $\widetilde{\mathbf{C}}_{:,k}^T$ to be **the sum of the output weights of the vectors of the cluster**, i.e., the sum of the set of vectors $\{\mathbf{C}_{:,i}^T, i \in I_k\}$
end for
for iter = 1 to *num_iter* **do**
  for $k \in [K]$ **do**
    for $j \in [m]$ **do**
      $\widetilde{C}_{jk} \leftarrow \frac{\langle \sum_{i \in I_k} C_{ji}(\mathbf{a}_i^T, b_i), (\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \rangle}{\|(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)\|^2}$
    end for
    $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \leftarrow \frac{\sum_{j=1}^m \sum_{i \in I_k} C_{ji}(\mathbf{a}_i^T, b_i)\widetilde{C}_{jk}}{\sum_{j=1}^m \widetilde{C}_{jk}^2}$
  end for
end for
Construct the new linear layer:
  For the input weights, the k-th row of weight-bias matrix is $(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$
  For the output weights, the k-th column is $\widetilde{\mathbf{C}}_{:,k}^T$

6 Extensions, Limitations, and Applications of the Method

Before proceeding to our experiments, we would like to discuss the applicability of our algorithm, highlight some limitations, and explore potential strategies to overcome these challenges.

Deep and Convolutional Networks. Our methods were analyzed previously for the case of networks with a single hidden linear layer with ReLU activations. However, they can be generalized both for deep and convolutional neural networks. In the first case, where we have a deep network with multiple layers, we can view each layer as an instance for our algorithm, by having an input from the previous layer and providing an output to the next layer. Therefore, we can recursively apply our multi-output algorithm to compress each hidden layer of the network.

In the case of convolutional networks, our methods can be extended for convolutional layers with ReLU and maxout or average pooling layers in between them. Here, instead of neurons, we have channels, and instead of weights that connect the neurons, we have kernels that connect the channels. For each hidden layer, the input and output weight matrices $\mathbf{A}$ and $\mathbf{C}$ are 4-dimensional, where each element of the row $(\mathbf{a}_i^T, b_i)$ and the column $\mathbf{C}_{:,i}$ is a kernel. Compressing a convolutional layer in a structured manner involves clustering channels. To apply our multi-output algorithm, we first flatten the matrices $\mathbf{A}$ and $\mathbf{C}$ so that they become 2-dimensional, by unravelling the kernels row-wise for $\mathbf{A}$ and column-wise

for $\mathbf{C}$. We then apply our multi-output algorithm on the flattened matrices. Finally, after obtaining the new $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{C}}$, we reshape the matrices to restore their original dimensions as convolutional layer weight matrices. We emphasize that we are the first to apply a tropical geometrical method to convolutional layers.

Batch normalization and other limitations. Our algorithm can not be applied in the case of networks with batch normalization layers in between the convolutional and linear layers. The reason for this is that batch normalization layers have a non-negligible effect on the zonotopes of the network we approximate. Here, we explain two methodologies that can be used to adapt networks with batch normalization to be applicable for our algorithm. These methodologies were used in the experiments to convert the VGG and ResNet models to equivalent models that our algorithm can be applied to. We propose two methodologies that allow most networks with batch normalization layers to be expressed equivalently as networks without batch normalization. As a first idea, we can fuse together a pair *(convolutional (or linear) layer, batch normalization layer)* into a single convolutional layer as follows: We take the pair we wish to fuse and form a network with just these two layers. We form a second network with a single, new convolutional layer. We then train the second network to give the same output as the first network on random noise input. This should produce a new convolutional layer that approximates the pair *(convolutional (or linear) layer, batch normalization layer)* with very little accuracy loss. Finally, we replace each such pair with its approximate convolutional (or linear) layer, which gives us an approximately equivalent network without batch normalization. An alternative second method is to shift the computations of the normalization layer directly to the convolutional or linear layer. Specifically, we can view the batch normalization layer being applied as a multiplication with corrective terms and addition with an extra bias term. We can incorporate these computations into the weights and bias of the convolutional or linear layer.

Using these ideas, we can fuse a batch normalization layer with its preceding convolutional or linear layer. This enables two approaches for applying our algorithm to networks with batch normalization. The first approach involves fusing all layers of the network beforehand, resulting in a fully fused network without batch normalization. The second approach performs fusion during the compression process, applying it layer-by-layer as each layer is compressed. For the second approach, we made a notable observation during our experiments, although the reasoning behind it remains unclear: using the pre-fusion weights/kernels of the layers for clustering vectors consistently led to significantly better results.

As for other limitations, we should note that our algorithm is designed to work strictly on layers as described above. Namely, convolutional or linear layers with ReLU and maxout or average pooling layers in between. Layers that apply computations, like batch normalization layers or the loops of residual blocks of ResNet networks fundamentally change the theory for the analysis of the network. For example, similarly to ThiNet, our methods can not be applied for the compression of the terminal convolutional layer of residual blocks.

Non-uniform compression. The techniques we have presented apply a uniform pruning ratio to all layers of the network. However, they can easily be extended to non-uniform pruning if instead of K-means we use hierarchical clustering with a global threshold parameter, like in the CUP framework (Duggal et al., 2021). In fact, when used with hierarchical clustering, TropNNC differs from CUP in step 1, where we choose a different

approach to build the clustering vectors (or filter features as Duggal et al. (2021) call them) of a convolutional layer, and more importantly in step 3, where we choose a different cluster representative based on tropical geometry. We also propose a modification to step 2 of CUP, specifically the hierarchical clustering step using a global distance threshold. Since the clustering vectors of different layers have varying dimensions, and vectors in higher-dimensional spaces tend to be more spread out, we introduce two variants for selecting the distance threshold for each layer:

- Variant 1: For each layer, take the distance threshold to be some global constant times the square root of the dimension of the clustering vectors of the layer.
- Variant 2: For each layer, take the distance threshold to be some global constant times the mean of the norms of the clustering vectors of the layer.

As demonstrated in our experiments, the first variant excels when minimizing the number of parameters, while the second variant is more effective when our objective is to minimize floating-point operation demands.

7 Experiments

We conduct experiments targeting the compression of the linear and convolutional layers of both convolutional and deep neural networks. Our method **TropNNC** performs layer-by-layer compression of both linear and convolutional layers and is suitable for regression and classification tasks. The empirical results from these experiments demonstrate that our algorithm achieves the expected performance as claimed. Throughout our experiments, we reserve the use of the iterative variant of TropNNC for cases where its slower performance is justified by the expectation of significantly better results compared to the non-iterative variant.

Baselines. We compare our framework with state-of-the-art methods that conduct structured pruning without requiring re-training (or fine-tuning). Specifically, we compare it with the algorithms proposed by Misiakos et al. (2022), ThiNet (Luo et al., 2017), CUP (Duggal et al., 2021) and the simple baselines random and L1 structured pruning. The algorithms presented in (Misiakos et al., 2022) are originally designed for linear layers only. To enable a fair comparison in compressing convolutional layers, we extended their approach using our proposed technique, making them applicable to convolutional layers as well. ThiNet employs a greedy criterion to remove neurons and channels that contribute least to the subsequent layer's input. For the non-uniform pruning variant of our framework, we compare it with CUP. CUP assigns to each neuron/channel a feature, performs hierarchical clustering based on these features, and replaces each neuron/channel in each cluster by a cluster representative according to a maximum norm criterion. As explained, our proposed algorithm enhances all three steps of CUP. Through an ablation study, we found that each improvement is essential for achieving a competitive advantage over CUP, indicating that these enhancements complement each other. In the presented experiments, we compare CUP exclusively with the fully enhanced version of TropNNC, incorporating all improvements. Furthermore, the trivial baseline of random structured pruning discards neurons or channels based on a uniform probability distribution, while L1 structured pruning targets those with the smallest L1 norm of their weights or kernels.

Percentage of Remaining Neurons	Zonotope K-means	TropNNC, single output	Neural Path K-means	TropNNC
100.0	99.42 ± 0.05	99.42 ± 0.05	99.44 ± 0.06	99.44 ± 0.06
10.0	99.38 ± 0.04	99.43 ± 0.04	99.41 ± 0.06	99.34 ± 0.16
5.0	99.37 ± 0.06	99.42 ± 0.06	99.37 ± 0.04	99.31 ± 0.16
1.0	99.33 ± 0.05	99.41 ± 0.04	99.32 ± 0.06	99.33 ± 0.05
0.5	99.12 ± 0.28	99.43 ± 0.04	99.40 ± 0.08	99.37 ± 0.04
0.3	95.98 ± 3.16	99.43 ± 0.04	98.39 ± 0.99	99.39 ± 0.10

Table 1: Comparison of methods on MNIST-3_5

Percentage of Remaining Neurons	Zonotope K-means	TropNNC, single output	Neural Path K-means	TropNNC
100.0	99.55 ± 0.04	99.55 ± 0.04	99.56 ± 0.04	99.56 ± 0.04
10.0	99.60 ± 0.07	99.54 ± 0.04	99.52 ± 0.04	99.50 ± 0.05
5.0	99.56 ± 0.04	99.54 ± 0.04	99.52 ± 0.04	99.50 ± 0.03
1.0	99.35 ± 0.29	99.54 ± 0.04	99.49 ± 0.14	99.47 ± 0.11
0.5	98.03 ± 0.71	99.54 ± 0.04	99.01 ± 0.50	99.48 ± 0.12
0.3	86.38 ± 7.59	99.54 ± 0.04	94.25 ± 4.57	99.47 ± 0.09

Table 2: Comparison of methods on MNIST-4_9

Datasets and networks. We evaluate our framework on the MNIST and CIFAR datasets, testing it across various models including simple multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012), and VGG (Simonyan and Zisserman, 2015). The non-uniform variant of our algorithm is tested on the CIFAR and ImageNet datasets, with its performance evaluated across models such as VGG, ResNet56, and ResNet18 (He et al., 2016).

7.1 MNIST Dataset, Pairs 3-5 and 4-9

The first experiment is performed on the binary classification tasks of pairs 3/5 and 4/9 of the MNIST dataset and so we can utilize both single-output and multi-output versions of our algorithm for the compression of the final hidden linear layer. In Tables 1 and 2, we compare single-output TropNNC and TropNNC with the corresponding single-output and multi-output algorithms from Misiakos et al. (2022). We use the same network, a simple CNN network with two fully connected layers and a final hidden linear layer of size 1000, and the same pruning ratios as in the corresponding experiment of Misiakos et al. (2022). According to Tables 1 and 2, our algorithm performs better than Misiakos et al. (2022) with particular difference in performance when using 0.3% of the nodes of the hidden layer.

7.2 MNIST and Fashion-MNIST Datasets

The second experiment is performed on the MNIST and FASHION-MNIST datasets. This classification task is multiclass, and thus only our multi-output algorithm may be used.

Percentage of Remaining Neurons	MNIST		FASHION-MNIST	
	Neural Path K-means	TropNNC	Neural Path K-means	TropNNC
100.0	98.54 ± 0.16	98.54 ± 0.16	89.16 ± 0.21	89.16 ± 0.21
50.0	97.85 ± 0.39	98.49 ± 0.14	88.17 ± 0.46	89.00 ± 0.25
25.0	96.69 ± 1.06	98.36 ± 0.14	86.33 ± 0.87	88.70 ± 0.22
10.0	96.25 ± 1.39	97.96 ± 0.35	84.91 ± 1.28	88.24 ± 0.40
5.0	95.17 ± 2.36	97.06 ± 0.73	81.48 ± 3.90	87.42 ± 0.46

Table 3: Comparison of Neural Path K-means and TropNNC on MNIST and FASHION-MNIST

Table 3 compares TropNNC with Neural Path K-means of Misiakos et al. (2022) for the same CNN network as above, and for the same pruning ratios as in (Misiakos et al., 2022). We again compress the final linear layer. As shown in the results, for both datasets, our algorithm outperforms Neural Path K-means.

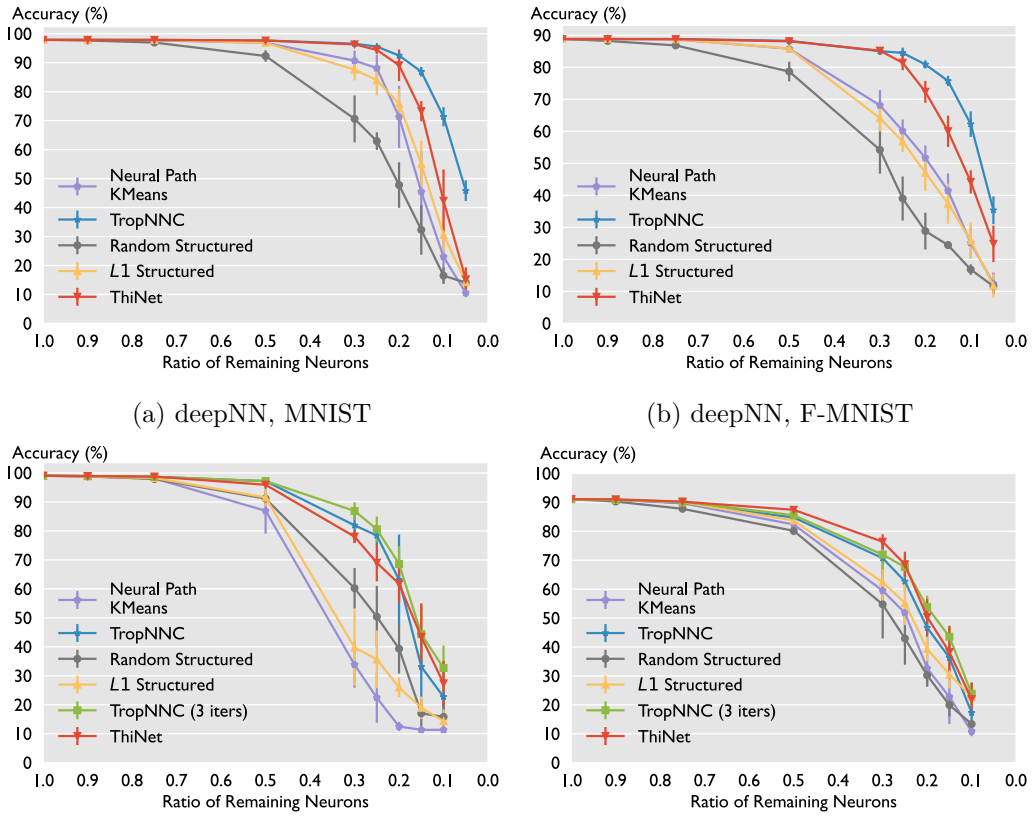
To evaluate the performance of our algorithm in compressing linear layers of deeper networks, we applied TropNNC to "deepNN", a fully connected neural network with layer sizes $28 \times 28, 512, 256, 128$, and 10. The performance plots are provided in Figures 7a and 7b. As illustrated, TropNNC outperforms both Neural Path K-means from (Misiakos et al., 2022) and ThiNet from (Luo et al., 2017).

To assess the performance of our algorithm in compressing convolutional layers, we applied TropNNC to "deepCNN2D", a LeNet-type convolutional neural network with ReLU activations. The performance plots are provided in Figures 7c and 7d. The results demonstrate that TropNNC outperforms Neural Path K-means from (Misiakos et al., 2022) and matches, or even surpasses, ThiNet from (Luo et al., 2017).

7.3 CIFAR Datasets

In this experiment, we compress AlexNet and VGG trained on the CIFAR-10 and CIFAR-100 datasets to assess the performance of each compression method. Figures 8a and 8b illustrate the compression of the linear layers of AlexNet on CIFAR-10 and CIFAR-100, respectively. Additionally, Figures 8c and 8d show the compression of VGG's convolutional layers for these datasets. For VGG, we applied compression to the fused networks without batch normalization, as we found that ThiNet's performance on the original networks was practically identical to its performance on the fused networks.

The results indicate that for larger datasets, TropNNC consistently outperforms Neural Path K-means from (Misiakos et al., 2022). Furthermore, it at least matches or even surpasses the performance of ThiNet from (Luo et al., 2017) for the compression of linear layers. These findings highlight the effectiveness of TropNNC in handling more complex and larger-scale data scenarios. For the compression of convolutional layers of VGG, TropNNC matches ThiNet. For the compression of convolutional layers of VGG, we do not present results for iterative TropNNC because it had almost identical performance to the non-iterative algorithm.



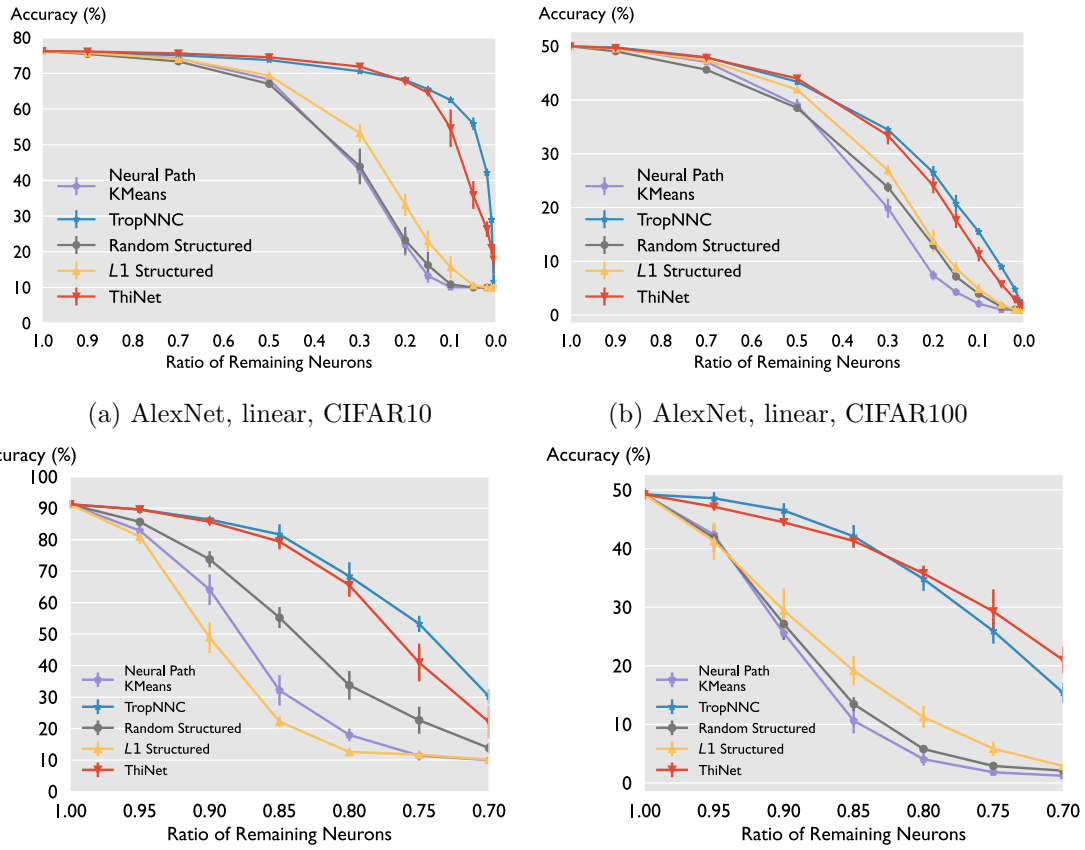
(a) deepNN, MNIST

(b) deepNN, F-MNIST

(c) deepCNN2D, MNIST

(d) deepCNN2D, F-MNIST

Figure 7: Compression of linear and convolutional layers of ReLU neural networks on MNIST datasets.



(a) AlexNet, linear, CIFAR10

(b) AlexNet, linear, CIFAR100

(c) VGG, conv., CIFAR10

(d) VGG, conv., CIFAR100

Figure 8: Compression of linear layers of AlexNet and convolutional layers of VGG on CIFAR datasets.

7.4 Non-ReLU Activation

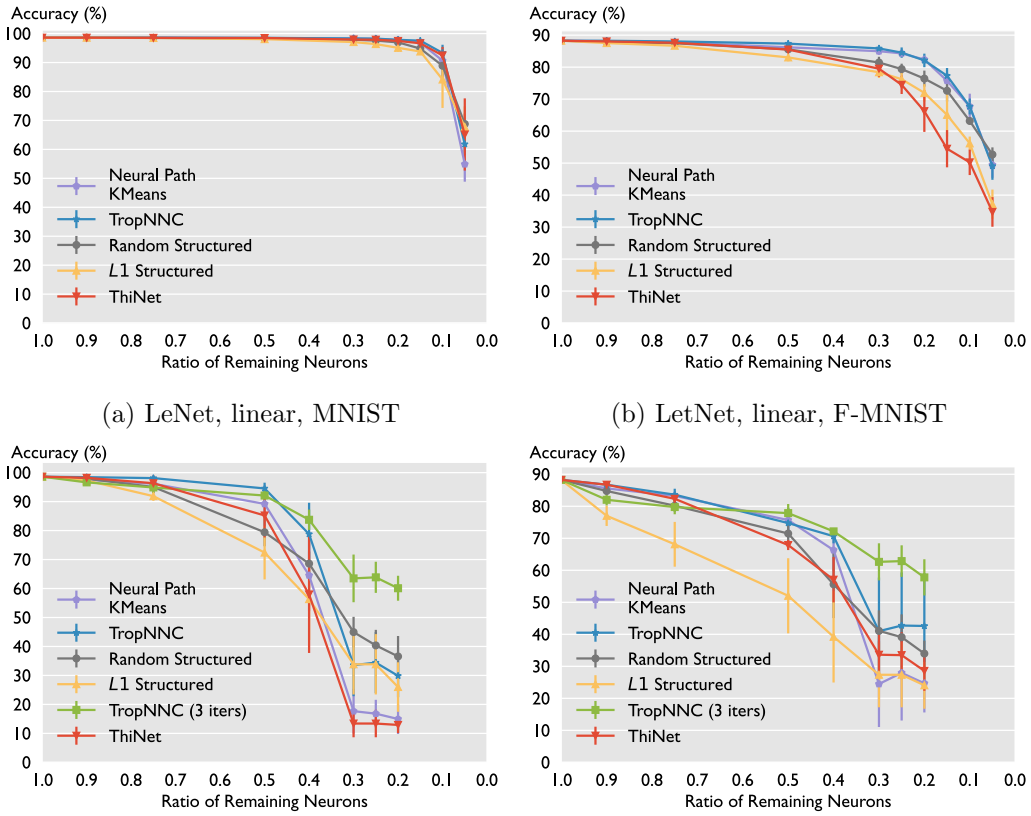
In this experiment, we evaluate the performance of our algorithm in compressing networks that do not use ReLU activations, such as LeNet (LeCun et al., 1998). We conduct experiments on the MNIST and FASHION-MNIST datasets, focusing on compressing the linear and convolutional layers of LeNet. The results are presented in Figures 9a, 9b, 9c and 9d. Interestingly, our algorithm demonstrates robust performance even on networks with non-ReLU activations.

7.5 Non-uniform Pruning

In this section, we evaluate the effectiveness of the non-uniform variant of TropNNC. Our experiments here compress various models, such as VGG on CIFAR-10, ResNet56 on CIFAR-10, and ResNet18 on ImageNet. The results are summarized in Tables 4, 5, 6, and 7. We should emphasize that we did not apply any form of fine-tuning or re-training. In Table 4 we present the compression of fused models, i.e. models whose batch normalization has been fused into its preceding convolutional or linear layer, as explained in a previous section. We noticed that the performance of CUP was better on the original networks with batch normalization. In Tables 5, 6 and 7 we compare our performance with the performance of CUP compressing the original networks. For our method only, we applied batch normalization fusion following the second approach described earlier—fusing the batch normalization layer with the preceding layer on a layer-by-layer basis, followed by compression of the fused layer before moving on with the next layer. We observed that our method performed significantly better when the clustering vectors were generated using the pre-fusion weights/kernels of the layers. While the reason for this improvement remains unclear, it consistently led to better performance in our experiments.

Our findings indicate that our method demonstrates a clear advantage. Notably, our approach shows a significant performance improvement for the VGG model, while the benefits are comparatively modest for ResNet models. The substantial advantage observed with VGG remains somewhat unexplained, and we have yet to determine why this effect is less pronounced for ResNet architectures. Further investigation may be required to fully understand these discrepancies and to optimize our approach across different model types.

We also found that the first variant of non-uniform TropNNC excelled at reducing the overall network size but was less effective at minimizing inference operations. In contrast, the second variant performed well in both tasks, outperforming CUP. Upon further analysis, we concluded that the first variant tends to focus more aggressively on the final layers, where parameter count is high due to the increased number of channels, but the number of operations is lower because of smaller image sizes. Meanwhile, the second variant, like CUP, also targets the initial layers, where fewer parameters are present, but a greater number of operations is required.



(a) LeNet, linear, MNIST

(b) LetNet, linear, F-MNIST

(c) LeNet, conv., MNIST

(d) LetNet, conv., F-MNIST

Figure 9: Compression of LeNet on MNIST Datasets.

Model	**Method**	**Threshold**	**#params ↓**	**FLOPS ↓**	**Acc. ↑**
VGG	Original	-	14.72M	0.63G	93.64
CIFAR10	CUP	0.15	4.70M	0.40G	56.22
	TropNNC (v2)	1.12	4.70M	0.33G	90.82
ResNet56	Original	-	0.85M	0.25G	93.67
CIFAR10	CUP	0.55	0.79M	0.18G	45.89
	TropNNC (v2)	1.2	0.65M	0.18G	76.50
ResNet18	Original	-	11.68M	3.63G	69.10
ImageNet	CUP	0.6	11.62M	3.50G	58.20
	TropNNC (v2)	1	11.60M	3.47G	62.60

Table 4: Comparison of CUP and TropNNC (variant 2) accuracy across different pruning thresholds for **fused** models without batch normalization.

Method	**Threshold**	**#params ↓**	**FLOPS ↓**	**Acc. ↑**
Original	-	14.7M	0.63G	93.64
CUP	0.15	6.24M	0.46G	92.41
TropNNC (v1)	0.006	5.70M	0.44G	93.58
CUP	0.20	4.62M	0.42G	67.91
TropNNC (v1)	0.0085	4.32M	0.41G	93.46
CUP	0.25	3.61M	0.39G	16.02
TropNNC (v1)	0.011	3.53M	0.39G	93.21

Table 5: Comparison of CUP and TropNNC (variant 1) accuracy across different pruning thresholds on CIFAR10, VGG.

Method	**Threshold**	**#params ↓**	**FLOPS ↓**	**Acc. ↑**
Original	-	0.85M	0.25G	93.67
CUP	0.55	0.72M	0.20G	87.32
TropNNC (v2)	1.17	0.71M	0.19G	90.53
CUP	0.6	0.63M	0.17G	78.83
TropNNC (v2)	1.23	0.63M	0.17G	87.04
CUP	0.65	0.54M	0.15G	71.48
TropNNC (v2)	1.3	0.54M	0.15G	73.03
TropNNC (v1)	0.067	0.49M	0.19G	82.70

Table 6: Comparison of CUP and TropNNC accuracy across different pruning thresholds on CIFAR10, ResNet56.

Method	**Threshold**	**#params ↓**	**FLOPS ↓**	**Acc. ↑**
Original	-	11.69M	3.64G	70.10
CUP	0.5	11.66M	3.58G	66.60
TropNNC (v2)	1.1	11.64M	3.44G	68.40
CUP	0.6	11.49M	3.38G	53.20
TropNNC (v2)	1.2	11.27M	3.07G	57.90
CUP	0.65	11.17M	3.15G	29.20
TropNNC (v2)	1.25	10.71M	2.75G	41.20
TropNNC (v1)	0.021	9.87M	3.46G	58.10

Table 7: Comparison of CUP and TropNNC accuracy across different pruning thresholds on ImageNet, ResNet18.

8 Conclusions and Future work

We improved upon the theoretical results of Misiakos et al. (2022) by using the (usual) Hausdorff distance instead of its discrete counterpart, to obtain a tighter bound for the approximation of tropical polynomials. Leveraging this enhancement we refined the choice of compressed node's weights that correspond to the K-means clusters from (Misiakos et al., 2022). Our novel algorithm, TropNNC, applied to linear and convolutional layers of deep neural networks outperforms Neural Path K-means of Misiakos et al. (2022), and manages to match or even surpass the performance of ThiNet, particularly for the compression of linear layers. In non-uniform pruning it performs better than CUP, with significant improvement in the case of the VGG network architecture. These achievements underscore the potential of tropical geometry in the realm of neural network compression. One promising direction for further investigation could involve optimizing the approximation techniques by considering the upper hulls of the network's tropical polynomials.

Appendix A. Proofs of Propositions and Lemmas

A.1 Proof of Lemma 2

Proof Since $(\mathbf{a}^T, b)$ lies below the upper envelope of P, there exists a point

$$(\mathbf{a}^T, b') = \sum_{i=1}^{k} \lambda_i \mathbf{v}_i, \quad \sum_{i=1}^{k} \lambda_i = 1$$

on a face of the upper envelope of P defined by the points $\mathbf{v}_1, \ldots, \mathbf{v}_k \in V_{UF(P)}$ such that $b' \geq b$. Therefore, we have:

$$\mathbf{a}^T\mathbf{x} + b \leq \mathbf{a}^T\mathbf{x} + b' = \langle (\mathbf{a}^T, b'), (\mathbf{x}, 1) \rangle$$

$$= \sum_{i=1}^{k} \lambda_i \langle \mathbf{v}_i, (\mathbf{x}, 1) \rangle \leq \max_{i=1,\ldots,k} \langle \mathbf{v}_i, (\mathbf{x}, 1) \rangle \leq p(\mathbf{x}).$$

If $(\mathbf{a}_i^T, b_i)$ lies strictly below the upper envelope of P, then $b' > b$, and the inequality is strict. ■

A.2 Proof of Lemma 8

Proof We will prove that

$$\sup_{\mathbf{u} \in P} \operatorname{dist}(\mathbf{u}, \tilde{P}) = \max_{\mathbf{u} \in V_P} \operatorname{dist}(\mathbf{u}, \tilde{P}),$$

i.e., the supremum is attained at some vertex of P.

The polytope P, the domain of the supremum, is convex and compact. Thus, it suffices to prove that the function

$$f(\mathbf{u}) = \operatorname{dist}(\mathbf{u}, \tilde{P}) = \inf_{\tilde{\mathbf{u}} \in \tilde{P}} \|\mathbf{u} - \tilde{\mathbf{u}}\|$$

is convex in terms of $\mathbf{u}$.

Let $\mathbf{u}_1, \mathbf{u}_2 \in P$. By the compactness of $\tilde{P}$, there exist points $\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2 \in \tilde{P}$ such that $f(\mathbf{u}_1) = \operatorname{dist}(\mathbf{u}_1, \tilde{P}) = \|\mathbf{u}_1 - \tilde{\mathbf{u}}_1\|$ and $f(\mathbf{u}_2) = \operatorname{dist}(\mathbf{u}_2, \tilde{P}) = \|\mathbf{u}_2 - \tilde{\mathbf{u}}_2\|$.

For every $\lambda \in [0, 1]$, we have that

$$\begin{aligned}\lambda f(\mathbf{u}_1) + (1-\lambda) f(\mathbf{u}_2) &= \lambda\|\mathbf{u}_1 - \tilde{\mathbf{u}}_1\| + (1-\lambda)\|\mathbf{u}_2 - \tilde{\mathbf{u}}_2\| \\ &\geq \|\lambda(\mathbf{u}_1 - \tilde{\mathbf{u}}_1) + (1-\lambda)(\mathbf{u}_2 - \tilde{\mathbf{u}}_2)\| \\ &= \|\lambda\mathbf{u}_1 + (1-\lambda)\mathbf{u}_2 - \lambda\tilde{\mathbf{u}}_1 - (1-\lambda)\tilde{\mathbf{u}}_2\|.\end{aligned}$$

By the convexity of $\tilde{P}$, $\tilde{\mathbf{u}} = \lambda\tilde{\mathbf{u}}_1 + (1-\lambda)\tilde{\mathbf{u}}_2 \in \tilde{P}$. Hence,

$$\lambda f(\mathbf{u}_1) + (1-\lambda) f(\mathbf{u}_2) \geq \|\lambda\mathbf{u}_1 + (1-\lambda)\mathbf{u}_2 - \tilde{\mathbf{u}}\| \geq f(\lambda\mathbf{u}_1 + (1-\lambda)\mathbf{u}_2),$$

which concludes the proof. ■

A.3 Proof of Proposition 10 and Theorem 11

Proof Consider a point $\mathbf{x} \in B$ and assume that $p(\mathbf{x}) = \mathbf{a}^T\mathbf{x} + b$ and $\tilde{p}(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + d$. Take an arbitrary $(\mathbf{u}^T, v) \in \tilde{P}$. This point lies below the upper envelope of $\tilde{P}$. Thus, by Lemma 2, we have that $\tilde{p}(\mathbf{x}) \geq \mathbf{u}^T\mathbf{x} + v$. Choose $(\mathbf{u}^T, v)$ to be the closest point to $(\mathbf{a}^T, b)$. Then,

$$\begin{aligned}p(\mathbf{x}) - \tilde{p}(\mathbf{x}) &\leq p(\mathbf{x}) - (\mathbf{u}^T, v)\begin{pmatrix}\mathbf{x}\\1\end{pmatrix} \\ &= ((\mathbf{a}^T, b) - (\mathbf{u}^T, v))\begin{pmatrix}\mathbf{x}\\1\end{pmatrix} \\ &\leq \|(\mathbf{a}^T, b) - (\mathbf{u}^T, v)\| \left\|\begin{pmatrix}\mathbf{x}\\1\end{pmatrix}\right\| \\ &\leq \operatorname{dist}((\mathbf{a}^T, b), \tilde{P}) \cdot \rho \\ &\leq \max_{(\mathbf{a}^T, b) \in V_P} \operatorname{dist}((\mathbf{a}^T, b), \tilde{P}) \cdot \rho,\end{aligned}$$

where the second inequality is due to the Cauchy-Schwarz inequality.

In a similar manner, take an arbitrary $(\mathbf{r}^T, s) \in P$. This point lies below the upper envelope of P. Thus, by Lemma 2, we have that $p(\mathbf{x}) \geq \mathbf{r}^T\mathbf{x} + s$. Choose $(\mathbf{r}^T, s)$ to be the closest point to $(\mathbf{c}^T, d)$. Then,

$$\begin{aligned}p(\mathbf{x}) - \tilde{p}(\mathbf{x}) &\geq (\mathbf{r}^T, s)\begin{pmatrix}\mathbf{x}\\1\end{pmatrix} - \tilde{p}(\mathbf{x}) \\ &= ((\mathbf{r}^T, s) - (\mathbf{c}^T, d))\begin{pmatrix}\mathbf{x}\\1\end{pmatrix} \\ &\geq -\|(\mathbf{r}^T, s) - (\mathbf{c}^T, d)\| \left\|\begin{pmatrix}\mathbf{x}\\1\end{pmatrix}\right\| \\ &\geq -\operatorname{dist}((\mathbf{r}^T, s), \tilde{P}) \cdot \rho \\ &\geq -\max_{(\mathbf{c}^T, d) \in V_{\tilde{P}}} \operatorname{dist}((\mathbf{c}^T, d), P) \cdot \rho.\end{aligned}$$

Finally, we obtain that

$$-\max_{(\mathbf{c}^T,d)\in V_{\tilde{P}}} \operatorname{dist}((\mathbf{c}^T,d),P)\cdot\rho \le p(\mathbf{x})-\tilde{p}(\mathbf{x}) \le \max_{(\mathbf{a}^T,b)\in V_P} \operatorname{dist}((\mathbf{a}^T,b),\tilde{P})\cdot\rho,$$

which implies

$$\frac{1}{\rho}|p(\mathbf{x})-\tilde{p}(\mathbf{x})| \le \max\left\{\max_{(\mathbf{a}^T,b)\in V_P} \operatorname{dist}((\mathbf{a}^T,b),\tilde{P}), \max_{(\mathbf{c}^T,d)\in V_{\tilde{P}}} \operatorname{dist}((\mathbf{c}^T,d),P)\right\}, \quad \forall \mathbf{x}\in B.$$

Therefore, by Lemma 8, we have,

$$\frac{1}{\rho}\max_{\mathbf{x}\in B}|p(\mathbf{x})-\tilde{p}(\mathbf{x})| \le H(P,\tilde{P}),$$

which concludes the proof of Proposition 10.
Finally, notice that we may write

$$\begin{aligned}
\|\mathbf{v}(\mathbf{x})-\tilde{\mathbf{v}}(\mathbf{x})\|_1 &= \sum_{j=1}^{m}|\mathbf{v}_j(\mathbf{x})-\tilde{\mathbf{v}}_j(\mathbf{x})| \\
&= \sum_{j=1}^{m}|(p_j(\mathbf{x})-q_j(\mathbf{x}))-(\tilde{p}_j(\mathbf{x})-\tilde{q}_j(\mathbf{x}))| \\
&= \sum_{j=1}^{m}|(p_j(\mathbf{x})-\tilde{p}_j(\mathbf{x}))-(q_j(\mathbf{x})-\tilde{q}_j(\mathbf{x}))| \\
&\le \sum_{j=1}^{m}\left(|p_j(\mathbf{x})-\tilde{p}_j(\mathbf{x})|+|q_j(\mathbf{x})-\tilde{q}_j(\mathbf{x})|\right).
\end{aligned}$$

By Proposition 10 we derive

$$\frac{1}{\rho}\max_{\mathbf{x}\in B}\|\mathbf{v}(\mathbf{x})-\tilde{\mathbf{v}}(\mathbf{x})\|_1 \le \sum_{j=1}^{m}\left(H(P_j,\tilde{P}_j)+H(Q_j,\tilde{Q}_j)\right).$$

■

A.4 Proof of Proposition 12

Proof For the output function, it holds that

$$v(\mathbf{x})=p(\mathbf{x})-q(\mathbf{x}), \quad \tilde{v}(\mathbf{x})=\tilde{p}(\mathbf{x})-\tilde{q}(\mathbf{x}).$$

From the triangle inequality, we deduce

$$\frac{1}{\rho}\,|v(\mathbf{x})-\tilde{v}(\mathbf{x})| \le H(P,\tilde{P})+H(Q,\tilde{Q}).$$

Thus, it suffices to get a bound on $H(P,\tilde{P})$ and $H(Q,\tilde{Q})$.

Let $I_+ \subseteq [n]$ and $I_- \subseteq [n]$ be the sets of positive and negative generators, respectively. First, we deal with $H(P, \tilde{P})$. Notice that $\forall \mathbf{v} \in V_{\tilde{P}}, \mathbf{v}$ is the sum of generators of some clusters of P. Thus $\forall \mathbf{v} \in V_{\tilde{P}}, \mathbf{v}$ is a vertex of P. Hence,

$$\operatorname{dist}(P, \mathbf{v}) = 0, \quad \forall \mathbf{v} \in V_{\tilde{P}}.$$

Let $I_k \subseteq [n]$ be the set of generators that belong to cluster k. Let $C_+ \subseteq [K]$ be the set of clusters of positive generators, and $C_- \subseteq [K]$ the set of clusters of negative generators.

Consider any vertex $\mathbf{u}$ of P. This vertex can be written as the sum of generators $c_i(\mathbf{a}_i^T, b_i)$, for some subset $I'_+ \subseteq I_+$. Thus,

$$\mathbf{u} = \sum_{i \in I'_+} c_i(\mathbf{a}_i^T, b_i).$$

For every positive generator $c_i(\mathbf{a}_i^T, b_i)$ that belongs to cluster k, define

$$x_i = \arg\min_x \left\| c_i(\mathbf{a}_i^T, b_i) - x\tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\|,$$

i.e., project the generator onto its cluster representative. Since every inner-cluster pair of generators forms an acute angle, any generator and its cluster representative (sum of generators of the cluster) will also form an acute angle, and thus $x_i \geq 0$.

For every cluster $k \in C_+$ define $I'_k = I_k \cap I'_+$ and

$$\tilde{x}_k = \sum_{i \in I'_k} x_i.$$

Since the cluster representative is the sum of the generators of the cluster, we have

$$\sum_{i \in I_k} x_i = 1 \Rightarrow \tilde{x}_k = \sum_{i \in I'_k} x_i \leq 1$$

Thus, for every cluster $k \in C_+$, the point $\tilde{x}_k \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ lies inside the segment $[\mathbf{0}, \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)]$ and thus belongs to $\tilde{P}$.

For the vertex $\mathbf{u}$, we choose to compare it with the point

$$\sum_{k \in C_+} \tilde{x}_k \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \in \tilde{P}.$$

Thus, we have that

$$\begin{aligned}
\operatorname{dist}(\mathbf{u}, \tilde{P}) &\leq \left\| \sum_{i \in I'_+} c_i(\mathbf{a}_i^T, b_i) - \sum_{k \in C_+} \tilde{x}_k \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\| \\
&\leq \sum_{k \in C_+} \left\| \sum_{i \in I'_k} c_i(\mathbf{a}_i^T, b_i) - \tilde{x}_k \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\| \\
&\leq \sum_{k \in C_+} \left\| \sum_{i \in I'_k} \left[c_i(\mathbf{a}_i^T, b_i) - x_i \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right] \right\| \\
&\leq \sum_{k \in C_+} \sum_{i \in I'_k} \left\| c_i(\mathbf{a}_i^T, b_i) - x_i \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\| \\
&= \sum_{k \in C_+} \sum_{i \in I'_k} \min_x \left\| c_i(\mathbf{a}_i^T, b_i) - x \tilde{c}_k(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\| \\
&= \sum_{k \in C_+} \sum_{i \in I'_k} \min_x \left\| c_i(\mathbf{a}_i^T, b_i) - x |I_k| (c_i(\mathbf{a}_i^T, b_i) + \epsilon_i) \right\| \\
&\leq \sum_{k \in C_+} \sum_{i \in I'_k} \min_x \left\{ |1 - x|I_k|| \cdot \left\| c_i(\mathbf{a}_i^T, b_i) \right\| + |x|I_k|| \cdot \|\epsilon_i\| \right\} \\
&= \sum_{k \in C_+} \sum_{i \in I'_k} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \|\epsilon_i\| \right\} \\
&\leq \sum_{k \in C_+} \sum_{i \in I'_k} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\} \\
&= \sum_{i \in I'_+} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\},
\end{aligned}$$

where ϵ_i is the error between generator i and the cluster center/mean of K-means.

The maximum value of the upper bound occurs when $I'_+ = I_+$. Thus,

$$\max_{\mathbf{u} \in V_P} \operatorname{dist}(\mathbf{u}, \tilde{P}) \leq \sum_{i \in I_+} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\}.$$

Finally, we have

$$\begin{aligned}
H(P, \tilde{P}) &= \max \left\{ \max_{\mathbf{u} \in V_P} \operatorname{dist}(\mathbf{u}, \tilde{P}), \max_{\mathbf{v} \in V_{\tilde{P}}} \operatorname{dist}(P, \mathbf{v}) \right\} \\
&\leq \max \left\{ \sum_{i \in I_+} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\}, 0 \right\} \\
&= \sum_{i \in I_+} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\}.
\end{aligned}$$

Similarly, for $H(Q, \tilde{Q})$, we have

$$H(Q, \tilde{Q}) \leq \sum_{i \in I_-} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\}.$$

Combining, we get

$$\frac{1}{\rho} |v(\mathbf{x}) - \tilde{v}(\mathbf{x})| \leq \sum_{i \in I} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\},$$

which concludes the proof of Proposition 12. ■

A.5 Proof of Corollary 13

Proof The bound of Misiakos et al. (2022) is the following:

$$\frac{1}{\rho} \max_{\mathbf{x} \in B} |v(\mathbf{x}) - \tilde{v}(\mathbf{x})| \leq K\delta_{\max} + \left(1 - \frac{1}{N_{\max}}\right) \sum_{i=1}^{n} |c_i| \|(\mathbf{a}_i^T, b_i)\|.$$

We will show that

$$K\delta_{\max} + \left(1 - \frac{1}{N_{\max}}\right) \sum_{i=1}^{n} |c_i| \|(\mathbf{a}_i^T, b_i)\| \geq \sum_{i \in I} \min \left\{ \left\| c_i(\mathbf{a}_i^T, b_i) \right\|, \delta_{\max} \right\}.$$

This can be rewritten as

$$K\delta_{\max} + \left(1 - \frac{1}{N_{\max}}\right) \sum_{i=1}^{n} |c_i| \|(\mathbf{a}_i^T, b_i)\| \geq \sum_{i \in I} \left\| c_i(\mathbf{a}_i^T, b_i) \right\| + \sum_{i \in I} \min \left\{ 0, \delta_{\max} - \left\| c_i(\mathbf{a}_i^T, b_i) \right\| \right\}.$$

Further simplifying, we get

$$K\delta_{\max} \geq \frac{1}{N_{\max}} \sum_{i=1}^{n} |c_i| \|(\mathbf{a}_i^T, b_i)\| + \sum_{i=1}^{n} \min \left\{ 0, \delta_{\max} - \left\| c_i(\mathbf{a}_i^T, b_i) \right\| \right\}.$$

It suffices to show that for every cluster k, we have:

$$\delta_{\max} \geq \frac{1}{|I_k|} \sum_{i \in I_k} |c_i| \|(\mathbf{a}_i^T, b_i)\| + \sum_{i \in I_k} \min \left\{ 0, \delta_{\max} - |c_i| \|(\mathbf{a}_i^T, b_i)\| \right\}.$$

However, it holds that

$$\sum_{i \in I_k} \min \left\{ 0, \delta_{\max} - |c_i| \|(\mathbf{a}_i^T, b_i)\| \right\} \leq \delta_{\max} - \max_{i \in I_k} |c_i| \|(\mathbf{a}_i^T, b_i)\|,$$

and

$$\max_{i \in I_k} |c_i| \|(\mathbf{a}_i^T, b_i)\| \geq \frac{1}{|I_k|} \sum_{i \in I_k} |c_i| \|(\mathbf{a}_i^T, b_i)\|.$$

Hence, we have

$$\sum_{i\in I_k} \min\left\{0, \delta_{\max} - |c_i|\|(\mathbf{a}_i^T, b_i)\|\right\} + \frac{1}{|I_k|}\sum_{i\in I_k} |c_i|\|(\mathbf{a}_i^T, b_i)\| \\ \leq \delta_{\max} - \max_{i\in I_k} |c_i|\|(\mathbf{a}_i^T, b_i)\| + \frac{1}{|I_k|}\sum_{i\in I_k} |c_i|\|(\mathbf{a}_i^T, b_i)\| \leq \delta_{\max},$$

which concludes the proof. ■

A.6 Proof of Proposition 14

Before we proceed with the proof of Proposition 14, we first give the definition of null neurons and generators.

Definition 15 (Null neuron/generator) *A neuron/generator $i \in [n]$ that belongs to cluster $k \in [K]$ is a null neuron/generator with respect to output $j \in [m]$ if $\tilde{c}_{jk}c_{ji} \leq 0$. N_j is the set of all null neurons with respect to output j.*

Proof Assume the algorithm iterative scheme has reached a stationary point (otherwise, assume the last step is an output weight update and the proof works fine). First, we focus on a single output, say j-th output. We will bound $H(P_j, \tilde{P}_j), H(Q_j, \tilde{Q}_j)$ for all $j \in [m]$.

Let I_{j+}, I_{j-} be the sets of positive and negative generators of output j. Let I_k be the set of neurons that belong to cluster k. Let C_{j+} be the set of positive clusters for output j (i.e. clusters for which $\tilde{c}_{jk} > 0$), and C_{j-} be the set of negative clusters for output j. Let $I_{jk} = I_k \cap I_{j+}$ if k is a positive cluster, else $I_{jk} = I_k \cap I_{j-}$

Consider any vertex $\mathbf{u}$ of P_j. This vertex can be written as the sum of generators $c_{ji}(\mathbf{a}_i^T, b_i)$, for some subset $I'_{j+} \subseteq I_{j+}$. Thus,

$$\mathbf{u} = \sum_{i\in I'_{j+}} c_{ji}(\mathbf{a}_i^T, b_i).$$

For every generator $c_{ji}(\mathbf{a}_i^T, b_i), i \in I_{j+}$ that belongs to positive cluster $k \in C_{j+}$, define

$$x_{ji} = \arg\min_x \left\| c_{ji}(\mathbf{a}_i^T, b_i) - x\tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\|,$$

i.e., project the generator onto $\tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ of its cluster k.

Assume a variant of the algorithm, where the optimization criterion is the following:

$$\sum_{j=1}^{m} \left\| |\tilde{C}_{jk}|(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) - \sum_{i\in I_{jk}} |C_{ji}|(\mathbf{a}_i^T, b_i) \right\|^2.$$

The sign of $\tilde{C}_{jk}$ never changes and gets fixed based on the initial solution. The set I_{jk} of the non-null generators of cluster k in terms of output j depends on the sign of $\tilde{C}_{jk}$, and it is determined by the initial solution. The output weight update rule changes: We update the absolute value of the weight $|\tilde{C}_{jk}|$. The update is performed as normal if the result is positive, otherwise we set $|\tilde{C}_{jk}| = 0$. We deduce that throughout the execution of the algorithm, after every output weight update step the following holds:

- If $|\tilde{C}_{jk}| = 0$ then, every generator of cluster k is a null generator by definition.

- If $|\tilde{C}_{jk}| > 0$ then the following argument holds.

By hypothesis, for every cluster k, the vectors of the set $\{(\mathbf{a}_i^T, b_i)|i \in I_k\}$ form pair-wise acute angles. Every vector of the set of vectors $\{\sum_{i\in I_{jk}} |c_{ji}|(\mathbf{a}_i^T, b_i)|j \in [m]\}$ lies inside the cone of the set of vectors $\{(\mathbf{a}_i^T, b_i)|i \in I_k\}$. It is easy to verify that the representative $|\tilde{c}_{jk}|(\tilde{\mathbf{a}}_k, \tilde{b}_k)$ that the iterative algorithm produces lies inside the cone of the set of vectors $\{\sum_{i\in I_{jk}} |c_{ji}|(\mathbf{a}_i^T, b_i)|j \in [m]\}$, which is a subset of the cone of the set of vectors $\{(\mathbf{a}_i^T, b_i)|i \in I_k\}$. Thus, the representative forms an acute angle with every vector of the set $\{(\mathbf{a}_i^T, b_i)|i \in I_k\}$, and thus $x_{ji} \geq 0$.

For every cluster $k \in C_{j+}$ define $I'_{jk} = I_k \cap I'_{j+}$ and

$$\tilde{x}_{jk} = \sum_{i\in I'_{jk}} x_{ji}.$$

Since $x_{ji} \geq 0$, and by the definition of the update step for the output weights, we have

$$\sum_{i\in I_{jk}} x_{ji} = 1 \Rightarrow \tilde{x}_{jk} = \sum_{i\in I'_{jk}} x_{ji} \leq 1$$

Thus, for every cluster $k \in C_{j+}$, the point $\tilde{x}_{jk}\tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)$ lies inside the segment $[\mathbf{0}, \tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k)]$, and thus belongs to $\tilde{P}_j$.

For the vertex $\mathbf{u}$, we choose to compare it with the point

$$\sum_{k\in C_{j+}} \tilde{x}_{jk}\tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \in \tilde{P}_j.$$

We have that

$$\tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) = \sum_{i\in I_{jk}} c_{ji}(\mathbf{a}_i^T, b_i) + l_{jk} = |I_{jk}|\frac{\sum_{i\in I_{jk}} c_{ji}(\mathbf{a}_t^T, b_i)}{|I_{jk}|} + l_{jk} = |I_{jk}|(c_{ji}(\mathbf{a}_i^T, b_i) + \epsilon_{ji}) + l_{jk},$$

where $\sum_{j=1}^m \|l_{jk}\|^2 = l_k^2$ the optimization criterion loss, and ϵ_{ji} is the error between $c_{ji}(\mathbf{a}_i^T, b_i)$ and the mean $\frac{\sum_{i\in I_{jk}} c_{ji}(\mathbf{a}_t^T, b_i)}{|I_{jk}|}$. It is easy to verify that ϵ_{ji} tends to zero as δ_{max} tends to 0.

Thus, we have that

$$\begin{aligned}
\operatorname{dist}(\mathbf{u}, \tilde{P}_j) &\leq \left\| \sum_{i \in I'_{j+}} c_{ji}(\mathbf{a}_i^T, b_i) - \sum_{k \in C_{j+}} \tilde{x}_{jk} \tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\| \\
&\leq \sum_{k \in C_{j+}} \left\| \sum_{i \in I'_{jk}} c_{ji}(\mathbf{a}_i^T, b_i) - \tilde{x}_{jk} \tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\| + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\| \\
&\leq \sum_{k \in C_{j+}} \left\| \sum_{i \in I'_{jk}} \left[c_{ji}(\mathbf{a}_i^T, b_i) - x_{ji} \tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right] \right\| + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\| \\
&\leq \sum_{k \in C_{j+}} \sum_{i \in I'_{jk}} \left\| c_{ji}(\mathbf{a}_i^T, b_i) - x_{ji} \tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \right\| + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\| \\
&\leq \sum_{k \in C_{j+}} \sum_{i \in I'_{jk}} \min_x \left\| c_{ji}(\mathbf{a}_i^T, b_i) - x \left(|I_{jk}| (c_{ji}(\mathbf{a}_i^T, b_i) + \epsilon_{ji}) + l_{jk} \right) \right\| + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\| \\
&\leq \sum_{k \in C_{j+}} \sum_{i \in I'_{jk}} \min_x \left\| (1 - x|I_{jk}|) c_{ji}(\mathbf{a}_i^T, b_i) - x l_{jk} - x|I_{jk}| \epsilon_{ji} \right\| + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\| \\
&\leq \sum_{k \in C_{j+}} \sum_{i \in I'_{jk}} \min_x \left\{ |1 - x|I_{jk}|| \|c_{ji}(\mathbf{a}_i^T, b_i)\| + |x|I_{jk}|| \left\| \frac{l_{jk}}{|I_{jk}|} + \epsilon_{ji} \right\| \right\} + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\| \\
&\leq \sum_{k \in C_{j+}} \sum_{i \in I'_{jk}} \min \left\{ \|c_{ji}(\mathbf{a}_i^T, b_i)\|, \left\| \frac{l_{jk}}{|I_{jk}|} + \epsilon_{ji} \right\| \right\} + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\| \\
&\leq \sum_{k \in C_{j+}} \sum_{i \in I'_{jk}} \min \left\{ \|c_{ji}(\mathbf{a}_i^T, b_i)\|, \frac{\|l_{jk}\|}{|I_{jk}|} + \|\epsilon_{ji}\| \right\} + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\|.
\end{aligned}$$

The maximum value of the upper bound occurs when $I'_{j+} = I_{j+}$. Thus, we have

$$\max_{\mathbf{u} \in V_{P_j}} \operatorname{dist}(\mathbf{u}, \tilde{P}_j) \leq \sum_{k \in C_{j+}} \sum_{i \in I_{jk}} \min \left\{ \|c_{ji}(\mathbf{a}_i^T, b_i)\|, \frac{\|l_{jk}\|}{|I_{jk}|} + \|\epsilon_{ji}\| \right\} + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\|.$$

To obtain a bound for $\max_{\mathbf{v} \in V_{\tilde{P}_j}} \operatorname{dist}(P_j, \mathbf{v})$, we write $\mathbf{v} = \sum_{k \in C'_{j+}} \tilde{c}_{jk}(\tilde{\mathbf{a}}_k^T, \tilde{b}_k) \in \tilde{P}_j$ and choose vertex $\mathbf{u} = \sum_{i \in I'_{j+}} c_{ji}(\mathbf{a}_i^T, b_i)$, with $I'_{j+} = \{i \in I_{j+} | i \in I_k, k \in C'_{j+}\}$. For this set we have $\tilde{x}_{jk} = 1$, and thus this distance has already been taken into account in the calculation of $\max_{\mathbf{u} \in V_{P_j}} \operatorname{dist}(\mathbf{u}, \tilde{P}_j)$.

At last, we have

$$H(P_j, \tilde{P}_j) \leq \sum_{k \in C_{j+}} \sum_{i \in I_{jk}} \min \left\{ \|c_{ji}(\mathbf{a}_i^T, b_i)\|, \frac{\|l_{jk}\|}{|I_{jk}|} + \|\epsilon_{ji}\| \right\} + \sum_{i \in N_{j+}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\|.$$

Similarly, for $H(Q_j, \tilde{Q}_j)$ we have

$$H(Q_j, \tilde{Q}_j) \leq \sum_{k \in C_{j-}} \sum_{i \in I_{jk}} \min \left\{ \|c_{ji}(\mathbf{a}_i^T, b_i)\|, \frac{\|l_{jk}\|}{|I_{jk}|} + \|\epsilon_{ji}\| \right\} + \sum_{i \in N_{j-}} |c_{ji}| \|(\mathbf{a}_i^T, b_i)\|.$$

Combining, we obtain

$$\frac{1}{\rho}\max_{\mathbf{x}\in B}|v_j(\mathbf{x})-\tilde{v}_j(\mathbf{x})| \leq \sum_{k=1}^{m}\sum_{i\in I_{jk}}\min\left\{\|c_{ji}(\mathbf{a}_i^T,b_i)\|, \frac{\|l_{jk}\|}{|I_{jk}|}+\|\epsilon_{ji}\|\right\}+\sum_{i\in N_j}|c_{ji}|\|(\mathbf{a}_i^T,b_i)\|.$$

Using the fact that $I_{jk}\subseteq I_k$ and $N_{min}\leq |I_{jk}|, \forall j,k$ we have

$$\frac{1}{\rho}\max_{\mathbf{x}\in B}|v_j(\mathbf{x})-\tilde{v}_j(\mathbf{x})| \leq \sum_{k=1}^{m}\sum_{i\in I_k}\min\left\{\|c_{ji}(\mathbf{a}_i^T,b_i)\|, \frac{\|l_{jk}\|}{N_{min}}+\|\epsilon_{ji}\|\right\}+\sum_{i\in N_j}|c_{ji}|\|(\mathbf{a}_i^T,b_i)\|.$$

We make use of the following inequality, which is a direct consequence of Cauchy-Schwartz Inequality

$$\sum_{j=1}^{m}|u_j|\leq\sqrt{m}\sqrt{\sum_{j=1}^{m}|u_j|^2}=\sqrt{m}\|(u_1,\ldots,u_m)\|.$$

We have

$$\sum_{j=1}^{m}|c_{ji}|\leq\sqrt{m}\|\mathbf{C}_{:,i}\|,$$

$$\sum_{j=1}^{m}\|l_{jk}\|\leq\sqrt{m}\sqrt{\sum_{j=1}^{m}\|l_{jk}\|^2}=\sqrt{m}\cdot l_k,$$

$$\sum_{j=1}^{m}\|\epsilon_{ji}\|\leq\sqrt{m}\sqrt{\sum_{j=1}^{m}\|\epsilon_{ji}\|^2}=\sqrt{m}\|\epsilon_{:,i}\|_F.$$

Using the above inequalities, the fact that $\sum\min\leq\min\sum$, and the fact that $\max\sum\leq\sum\max$ we get

$$\begin{aligned}\frac{1}{\rho}\max_{\mathbf{x}\in B}\|v(\mathbf{x})-\tilde{v}(\mathbf{x})\|_1 &\leq \sum_{k=1}^{m}\sum_{i\in I_k}\min\left\{\sum_{j=1}^{m}|c_{ji}|\|(\mathbf{a}_i^T,b_i)\|, \frac{\sum_{j=1}^{m}\|l_{jk}\|}{N_{min}}+\sum_{j=1}^{m}\|\epsilon_{ji}\|\right\}\\ &\quad+\sum_{j=1}^{m}\sum_{i\in N_j}|c_{ji}|\|(\mathbf{a}_i^T,b_i)\|\\ &\leq\sqrt{m}\sum_{i=1}^{n}\min\left\{\|\mathbf{C}_{:,i}\|\|(\mathbf{a}_i^T,b_i)\|, \frac{l_{k(i)}}{N_{min}}+\|\epsilon_{:,i}\|_F\right\}\\ &\quad+\sum_{j=1}^{m}\sum_{i\in N_j}|c_{ji}|\|(\mathbf{a}_i^T,b_i)\|,\end{aligned}$$

as desired. ■

References

M. Akian, S. Gaubert, and A. Guterman. Tropical Polyhedra Are Equivalent To Mean Payoff Games. *International Journal of Algebra and Computation*, 22(1):1250001, 2012.

Motasem Alfarra, Adel Bibi, Hasan Hammoud, Mohamed Gaafar, and Bernard Ghanem. On the decision boundaries of neural networks: A tropical geometry perspective. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):5027–5037, 2023. doi: 10.1109/TPAMI.2022.3201490.

F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. J. Wiley & Sons, 2001.

Kayhan Behdin, Ayan Acharya, Aman Gupta, Sathiya Keerthi, and Rahul Mazumder. Quantease: Optimization-based quantization for language models–an efficient and intuitive algorithm. *arXiv preprint arXiv:2309.01885*, 2023.

Marshall Bern and David Eppstein. Optimization over zonotopes and training support vector machines. In Frank Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures*, pages 111–121, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44634-7.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 129–146, 2020. URL https://proceedings.mlsys.org/paper_files/paper/2020/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf.

Peter Butkovič. *Max-linear systems: theory and algorithms*. Springer, 2010.

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

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/a2b8a85a29b2d64ad6f47275bf1360c6-Paper.pdf.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2285–2294, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/chenc15.html.

G. Cohen, S. Gaubert, and J.P. Quadrat. Duality and separation theorems in idempotent semimodules. *Linear Algebra and its Applications*, 379:395–422, 2004.

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

Ray A Cuninghame-Green. Minimax algebra and applications. In *Advances in imaging and electron physics*, volume 90, pages 1–121. Elsevier, 1994.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/7fec306d1e665bc9c748b5d2b99a6e97-Paper.pdf.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/2afe4567e1bf64d32a5527244d104cea-Paper.pdf.

Rahul Duggal, Cao Xiao, Richard Vuduc, Duen Horng Chau, and Jimeng Sun. Cup: Cluster pruning for compressing deep neural networks. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5102–5106, 2021. doi: 10.1109/BigData52589.2021.9671980.

S. Gaubert and R. D. Katz. Minimal half-spaces and external representation of tropical polyhedra. *Journal of Algebraic Combinatorics*, 33:325–348, 2011.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Peter Gritzmann and Bernd Sturmfels. Minkowski addition of polytopes: computational complexity and applications to gröbner bases. *SIAM Journal on Discrete Mathematics*, 6(2):246–269, 1993.

Yanming Guo, Yu Liu, Theodoros Georgiou, and Michael S Lew. A review of semantic segmentation using deep neural networks. *International journal of multimedia information retrieval*, 7:87–93, 2018.

Bernd Gärtner and Martin Jaggi. Tropical support vector machines. Technical Report ACS-TR-362502-01, ACS, 2008.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf.

Robin Hartshorne. *Algebraic geometry*, volume 52. Springer, 2013.

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)*, June 2016.

Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):2900–2919, 2024. doi: 10.1109/TPAMI.2023.3334614.

MD Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CsUR)*, 51(6):1–36, 2019.

Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. In *International Conference on Learning Representations*, 2016.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Xu Jia, Efstratios Gavves, Basura Fernando, and Tinne Tuytelaars. Guiding the long-short term memory model for image caption generation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2407–2415, 2015. doi: 10.1109/ICCV.2015.277.

Ioannis Kordonis, Emmanouil Theodosis, George Retsinas, and Petros Maragos. Matrix factorization in tropical and mixed tropical-linear algebras. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6090–6094, 2024. doi: 10.1109/ICASSP48485.2024.10446164.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL `https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf`.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

G. L. Litvinov, V. P. Maslov, and G. B. Shpiz. Idempotent functional analysis: An algebraic approach. *Mathematical Notes*, 69(5):696–729, 2001.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Diane Maclagan and Bernd Sturmfels. *Introduction to tropical geometry*, volume 161. American Mathematical Society, 2021.

Petros Maragos. Dynamical systems on weighted lattices: General theory. *Mathematics of Control, Signals, and Systems*, 29:1–49, 2017.

Petros Maragos and Emmanouil Theodosis. Multivariate tropical regression and piecewise-linear surface fitting. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3822–3826, 2020. doi: 10.1109/ICASSP40776.2020.9054058.

Petros Maragos, Vasileios Charisopoulos, and Emmanouil Theodosis. Tropical geometry and machine learning. *Proceedings of the IEEE*, 109(5):728–755, 2021. doi: 10.1109/JPROC.2021.3065238.

Panagiotis Misiakos, Georgios Smyrnis, George Retsinas, and Petros Maragos. Neural network approximation based on hausdorff distance of tropical zonotopes. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=oiZJwC_fyS`.

M. Mohri, F. Pereira, and M. Ripley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech and Language*, 16:69–88, 2002.

Cara Monical, Neriman Tokcan, and Alexander Yong. Newton polytopes in algebraic combinatorics. *Selecta Mathematica*, 25(66):37, 2019.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper_files/paper/2014/file/109d2dd3608f669ca17920c511c2a41e-Paper.pdf`.

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528, 2015. doi: 10.1109/ICCV.2015.178.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S1XolQbRW`.

Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2023. doi: 10.1109/HPEC58863.2023.10363447.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/851300ee84c2b80ed40f51ed26d866fc-Paper.pdf.

Georgios Smyrnis and Petros Maragos. Multiclass neural network minimization via tropical Newton polytope approximation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9068–9077. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/smyrnis20a.html.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. In *International Conference on Learning Representations*, 2016.

Emmanouil Theodosis and Petros Maragos. Analysis of the viterbi algorithm using tropical algebra and geometry. In *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5, 2018. doi: 10.1109/SPAWC.2018.8445777.

N. Tsilivis, A. Tsiamis, and P. Maragos. Toward a sparsity theory on weighted lattices. *Journal of Mathematical Imaging and Vision*, 64(7):705–717, 2022. doi: 10.1007/s10851-022-01075-1. URL http://robotics.ntua.gr/wp-content/uploads/sites/2/2022_TsilivisTsiamisMaragos_SparsityTheoryOnWeightedLattices_JMIV.pdf.

Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14913–14922, June 2021.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/41bfd20a38bb1b0bec75acf0845530a7-Paper.pdf.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Xuejiao Yang and Joseph K. Scott. A comparison of zonotope order reduction techniques. *Automatica*, 95:378–384, 2018. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2018.06.006. URL https://www.sciencedirect.com/science/article/pii/S000510981830298X.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5824–5832. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/zhang18i.html`.

Shu-Chang Zhou, Yu-Zhi Wang, He Wen, Qin-Yao He, and Yu-Heng Zou. Balanced quantization: An effective and efficient approach to quantized neural networks. *Journal of Computer Science and Technology*, 32:667–682, 2017.

Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Günter M Ziegler. *Lectures on polytopes*, volume 152. Springer, 2012.

Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 111(3):257–276, 2023. doi: 10.1109/JPROC.2023.3238524.