HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

FEDERAL STATE AUTONOMOUS EDUCATIONAL INSITUTION FOR HIGHER PROFESSIONAL EDUCATION NATIONAL RESEARCH UNIVERSITY „HIGHER SCHOOL OF ECONOMICS“

Faculty of Computer Science

Anton Zakharenkov

Representing Tropical Rational Functions via ReLU Neural Networks

Group mFTiAD18

Qualification paper – Master of Science Dissertation
Field of study 01.04.02 Applied Mathematics and Informatics

Program: Financial Technologies and Data Analysis

Supervisor:
PhD. Gleb Pogudin

June 3, 2020

ВЫСШАЯ ШКОЛА ЭКОНОМИКИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
"ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Факультетет Компьютерных Наук

Антон Захаренков

Представление Тропических Рациональных Функций с Помощью ReLU-нейросетей

Группа мФТиАД18

Квалификационная работа – Магистерская Диссертация
Направлению подготовки: 01.04.02 Прикладная Математика и Информатика

Образовательная программа: Финансовые Технологии и Анализ Данных

Научный руководитель:
к. ф.-м. н. Глеб Погудин

3 июня 2020 г.

Abstract

This dissertation continues the study of the relationship between tropical algebraic geometry and neural networks. We will give an overview of our *tropical* python framework [16], which provides a convenient way to describe such relationship from an algebraic, geometric and combinatorial perspective, allowing us to perform simple arithmetic operations in a tropical semiring, construct tropical curves and their dual subdivisions, simplify the structure of tropical polynomials removing redundant monomials, factor some of tropical polynomials and so on. In particular, we make use of the functionality it provides, in order to visualize the decision boundary of a two-layer fully connected ReLU network, trained for a binary classification problem.

Continuing the work of Zhang[17], we use this framework for the implementation of conversion algorithms between certain family of ReLU neural networks and tropical rational functions and vice versa. We also conduct several experiments on such conversion and observe a tropical polynomial "explosion" phenomenon.

Аннотация

Эта диссертация продолжает исследование взаимосвязи между тропической алгебраической геометрией и нейронными сетями. Мы дадим обзор нашей библиотеки *tropical* [16] на языке Python, которая предоставляет удобный способ описания этой взаимосвязи с алгебраической, геометрической и комбинаторной точек зрения, позволяя нам выполнять простые арифметические операции в тропическом полукольце, строить тропические кривые и их двойные подразделения, упрощать структуру тропических полиномов удаляя избыточные мономы и так далее. В частности, мы используем предоставляемые ей функциональные возможности для визуализации границы принятия решения двухслойной нейронной ReLU сети, обученной для бинарной задачи классификации.

Продолжая работу Zhang [17], мы используем эту библиотеку для реализации алгоритмов конвертации определенного семейства нейронных ReLU сетей в тропические рациональные функции и обратно. Мы также проведём несколько экспериментов с этими конвертациями и продемонстрируем явление «разрастания» тропических полиномов.

Contents

Chapter 1

Introduction

1.1 Motivation

Tropical mathematics, a field of mathematics which redefines the rules of arithmetic by replacing addition and multiplication with maximum and addition, respectively, finds its applications in a variety of domains, such as optimization [10], design of auctions [15], and evolutionary biology [12]. One of the key objects of study are polynomials build with these new operations. They define piecewise-linear functions allowing to use combinatorial and geometric methods to study such functions.

Zhang et al.[17] showed a close relationship between tropical polynomials and family of functions represented by feedforward neural networks with rectified linear units (ReLU) and integer weights. This connection helps to study both theoretical capabilities of neural networks, such as their expressiveness [17], and the methods of their pruning and initialization [1]. An important point is the close geometric connection between piece-wise linear surfaces, tropical hypersurfaces, and the decision boundaries of neural networks.

1.2 Outline

The rest of this work is structured as follows. In Chapter 2, we recall basic notions of tropical geometry and neural networks and survey related research. In Chapter 3, we describe our implementation of tropical algebra in Python and some basic algorithms for working with tropical polynomials. In Chapter 4, we discuss the algorithms for converting a tropical polynomial to a neural network and backwards from [17] and our implementation of these algorithms. We analyze these algorithms and demonstrate that their outputs can be further substantially improved, and which is one of the starting point for the rest of the work. In Chapter 5, we consider tropical representation of the simple neural network and its connection to decision boundaries. In Chapter 6, we summarize the work and describe future work.

Chapter 2

Background

In Section 2.1 we introduce the basics tropical arithmetic and tropical linear algebra. Then in Section 2.2 we provide an overview of tropical algebraic geometry of relevance to us. For further details, we refer to [11] (see also [13, 17]).

2.1 Tropical Arithmetic and Tropical Linear Algebra

Definition 2.1. The tropical semiring $\mathbb{T}$ is the triplet $\{\mathbb{R}\cup\{-\infty\},\oplus,\odot\}$, where $x\oplus y := \max\{x,y\}$ and $x\odot y := x+y$ are *tropical addition* and *tropical multiplication*, respectively. Also we define *tropical quotient* of x and y as $x \oslash y := x - y$ and *tropical power* as $x^{\odot a} := a \cdot x$ for $a \in \mathbb{R}$ (will be used for the case $a \in \mathbb{Z}$).

The distinguished element $-\infty$ has the property that it is smaller than any element of $\mathbb{R}$. This means, for instance, that $-\infty \oplus 2 = 2$, and $-\infty \odot 2 = -\infty$.

$\mathbb{T}$ *almost* has the structure of an associative, commutative, and distributive ring with unity, where $-\infty$ is the additive identity and 0 is the multiplicative identity. Every $x \in \mathbb{R}$ has a tropical multiplicative inverse $x^{\odot(-1)} := -x$. However, here we do not have additive inverses, so the triplet $\{\mathbb{R}\cup\{-\infty\},\oplus,\odot\}$ is a semiring.

Definition 2.2. Let $x_1,\ldots,x_d$ be variables representing elements of $\mathbb{T}$. A *tropical monomial* is a finite product of any of these variables, with repetition allowed. Any tropical monomial can be written as:

$$x_1^{\odot\alpha_1} \odot \cdots \odot x_d^{\odot\alpha_d}.$$

Definition 2.3. A *tropical polynomial* is a finite sum of tropical monomials:

$$c_1 x_1^{\alpha} \oplus \cdots \oplus c_n x^{\alpha_n}$$

with $c_i \in \mathbb{T}$ and $\boldsymbol{\alpha}_i := (\alpha_{i1}, \ldots, \alpha_{id}) \in \mathbb{Z}^d$ and a monomial of a given multiindex appears at most once in the sum, i.e., $\alpha_i \neq \alpha_j$ for any $i \neq j$.

Evaluating in classical algebra, a tropical polynomial is the maximum of a finite number of linear functions:

$$p(\boldsymbol{x}) = \max\{c_1 + \alpha_{11}x_1 + \cdots + \alpha_{1d}x_d, \ldots, c_n + \alpha_{n1}x_1 + \cdots + \alpha_{nd}x_d\}$$

Remark. For two polynomials f and g, we write $f = g$ if they are equal coefficient-wise. We can also view of a tropical polynomial as a function. Two polynomials f and g are functionally equivalent if, for each $x \in R$, $f(x) = g(x)$. Functional equivalence does not imply equality. For example, the polynomials $x_1 \odot x_2 \oplus x_1^2 \oplus x_2^2$ and $x_1^2 \oplus x_2^2$ are functionally equivalent, but not equal as polynomials. This follows from the fact that in the first polynomial monomial $x_1 \odot x_2$ is less than or equal to x_1^2 for $x_2 \leq x_1$ and less than or equal to x_2^2 for $x_1 \leq x_2$, which implies that its value at any point coincides with $x_1^2 \oplus x_2^2$. We will be mostly interested in functional equivalence although it is harder to test 3.3.

Tropical polynomials are continuous, piecewise-linear with a finite number of pieces, and convex as taking max and sum of convex functions preserve convexity.

Definition 2.4. A tropical quotient of two tropical polynomials $f(\boldsymbol{x}) \oslash g(\boldsymbol{x})$ is a tropical rational function.

Definition 2.5. A map $F : \mathbb{R}^d \to \mathbb{R}^p : \boldsymbol{x} \mapsto (f_1(\boldsymbol{x}), \ldots, f_p(\boldsymbol{x}))$ is a *tropical polynomial map* if each $f_i(\boldsymbol{x})$ can be represented by a tropical polynomial and a *tropical rational map* if each $f_i(\boldsymbol{x})$ can be represented by a tropical rational function.

2.2 Tropical Algebraic Geometry

To every tropical polynomial, we associate a tropical hypersurface. Although this correspondence looks different from the classical algebraic geometry case, many crucial properties of hypersurfaces (and, more generally, zero sets) are preserved.

Definition 2.6. For a tropical polynomial $f(x) = c_1 x^{\alpha_1} \oplus \cdots \oplus c_n x^{\alpha_n}$, we say it *vanishes* at $x \in \mathbb{R}^d$ if the value $f(x)$ is attained by at least two monomials of f. We define the *tropical hypersurface* of f as the tropical zero locus:

$$\begin{aligned}\mathcal{T}(f) &:= \{x \in \mathbb{R}^d : c_i x^{\alpha_i} = c_j x^{\alpha_j} = f(x) \text{ for some } i \neq j\} \\ &= \{x \in \mathbb{R}^d : f(x) \text{ is attained by by two or more monomials in } f\}\end{aligned}$$

In two dimensions (i.e., in $\mathbb{R}^2$) a we call the tropical hypersurface a *tropical curve.*

The tropical hypersurface of a polynomial f in d variables divides $\mathbb{R}^d$ into cells, on each of which f is linear. These cells are convex polytopes with rational slopes and thus can be defined by $\{x \in \mathbb{R}^d : \mathrm{A}x \leq b\}$ for $A \in \mathbb{Z}^{m\times d}$, $b \in \mathbb{R}^m$. The tropical hypersurface of f is the union of boundaries of these cells.

As in classical algebraic geometry, we can define the Newton polytope of a polynomial.

Definition 2.7. For a tropical polynomial $f(x) = c_1 x^{\alpha_1} \oplus \cdots \oplus c_n x^{\alpha_n}$, we define the *Newton polytope* of f as the convex hull of $\alpha_1, \ldots, \alpha_n$ in $\mathbb{R}^d$,

$$\Delta(f) := \mathrm{Conv}\{\alpha_i : c_i \neq -\infty, i = 1...n\}.$$

and the *extended Newton polytope* of f as

$$\mathcal{P}(f) := \mathrm{Conv}\{(c_i, \alpha_i) \in \mathbb{R}^{d+1} : \alpha_i \neq -\infty\}$$

The faces of extended Newton polytope $\mathcal{P}(f)$ that are visible from above with respect to the first coordinate form the upper convex hull and called UF($\mathcal{P}(f)$)

Definition 2.8. Letting $\pi : \mathbb{R}^{d+1} \to \mathbb{R}^d$ be the projection which drops the first coordinate, the *dual subdivision* of f is then the projection of the upper faces of $\mathcal{P}(f)$, i.e.

$$\delta(f) := \pi(\mathrm{UF}(\mathcal{P}(f))).$$

$\delta(f)$ forms a polyhedral complex with support $\Delta(f)$.

Definition 2.9. A polyhedral complex is a collection Σ of polyhedra satisfying two conditions:

- if P is in Σ , then so is any face of P
- if P and Q lie in Σ, then $P \cap Q$ is either empty or a face of both P and Q

The polyhedra in a polyhedral complex E are called the cells of E. Cells of E that are not faces of any larger cell are facets of the complex. Their facets are called ridges of the complex. The support $|\Sigma|$ of Σ is the set $\{x \in R^n : x \in P \text{ for some } P \in \Sigma\}$.

The subdivision of the Newton polygon induced by the coefficients of the tropical polynomial gives us almost all the information regarding how to draw the tropical curve in the plane. Although this result holds in greater generality, we state it in the case of two variables.

Theorem 2.10 (The Duality Theorem, [11, Proposition 3.1.6])**.** *Let $f(x, y)$ be a tropical polynomial with $\Delta(f)$ two-dimensional. Then the tropical curve $\mathcal{T}(f)$ is dual to the subdivision of f induced by the coefficients of $f(x, y)$ in the following sense:*

- *Vertices of $\mathcal{T}(f)$ correspond to polygons in the subdivision of $\Delta(f)$.*
- *Edges of $\mathcal{T}(f)$ correspond to interior edges in the subdivision of $\Delta(f)$.*
- *Rays of T (p) correspond to boundary edges in the subdivision of $\Delta(f)$.*
- *Regions of $\mathbb{R}^2$ separated by $\mathcal{T}(f)$ correspond to the lattice points of $\Delta(f)$ used in the subdivision.*

Moreover, two vertices of $\mathcal{T}$ are connected by an edge if and only if their corresponding polygons in the subdivision share an edge, and the edge in the Newton polygon is perpendicular to the edge in the subdivision; and the rays emanating from a vertex in $\mathcal{T}(f)$ correspond to boundary edges of the corresponding polygon in the subdivision, with the rays in the outward perpendicular directions to the boundary edges of $\Delta(f)$.

Remark 2.11. By only considering tropical curve as a set of rays and segments, we can lose information about the starting polynomial. This leads us to decorate the edges and rays of our tropical curves with weights. In particular, each edge or ray is given a positive integer weight m, where m is equal to one less than the number of lattice points on the dual edge of the subdivision.

Proposition 2.12. *Any tropical curve satisfies the following balancing condition: choose a vertex, and let $\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, ..., \langle a_n, b_n\rangle$ be the outgoing directions of the rays and edges emanating from the vertex, where $a_i, b_i \in Z$ and $gcd(a_i, b_i) = 1$ for all i. Let m_i denote the weight of the ith edge/ray. Than*

$$m_1 * \langle a_1, b_1\rangle + m_2 * \langle a_2, b_2\rangle + \cdots + m_n * \langle a_n, b_n\rangle = \langle 0, 0\rangle$$

Proposition 2.13. *For two tropical polynomials, f and g, we have*

- $\Delta(f \odot g) = \Delta(f) + \Delta(g)$
- $\mathcal{T}(f \odot g) = \mathcal{T}(f) \cup \mathcal{T}(g)$

Example 2.1. *For polynomial $4 \odot a \oplus 4 \odot a^2 \odot b^4 \oplus 8 \odot a^8 \odot b^4 \oplus a^7 \odot b^6 \oplus b^8$ from 3.5 let's visualize a tropical curve and corresponding dual subdivision*

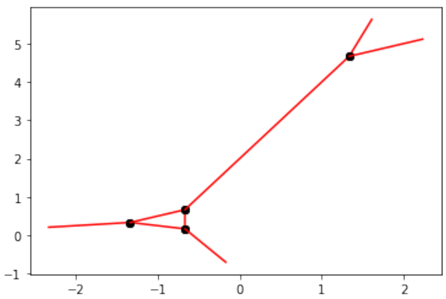


Figure 2.1: Tropical curve

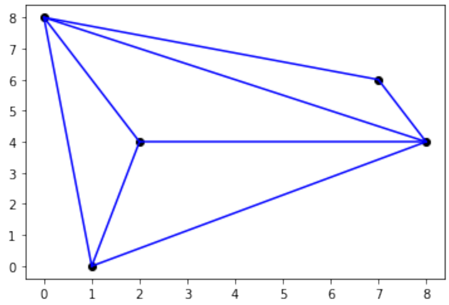


Figure 2.2: Dual subdivision

2.3 Neural Networks

Definition 2.14. An *L-layer feedforward neural network* is a map $v : \mathbb{R}^d \to \mathbb{R}^p$ defined by a composition of functions

$$v^{(L)} = \sigma^{(L)} \circ \rho^{(L)} \circ \cdots \circ \sigma^{(1)} \circ \rho^{(1)}$$

with fixed *preactivation functions* $\rho^{(k)}(x) = W^{(k)}x + b^{(k)}$, where $W \in \mathbb{R}^{n_k \times n_{k-1}}$ is called the *weight matrix* and $b^{(k)}$ is the *bias vector* of $\rho^{(k)}$, and usually non-linear *activation functions* $\sigma^{(k)} : \mathbb{R}^{n_k} \to \mathbb{R}^{n_k}$.

Definition 2.15. *ReLU neural networks* is a class of neural networks $v : \mathbb{R}^d \to \mathbb{R}^p$ with ReLU activation function

$$\sigma^{(k)}(x) = \max\{x, t\}$$

where $t \in \mathbb{R}$ is called a threshold vector and max is taken coordinate-wise. Networks from this class are piecewise-linear functions.

Definition 2.16. The *decision boundary* of a neural network with output $v = (v_1, \ldots, v_p)$ is the set of inputs that give (at least) two nodes with equal output i.e.

$$\mathcal{B}(v) = \{x \in \mathbb{R}^d : v_i(x) = v_j(x) \text{ for some } i \neq j\}$$

We will restrict ourselves to integer weight matrices, i.e. $W^{(k)} \in \mathbb{Z}^{n_k \times n_{k-1}}$. This restriction is not very strict because one can always use approximation to rational numbers and clear denominators to obtain integer weights. A similar feature is provided by the popular deep learning library Pytorch via it's quantization module.[1]

[1] https://pytorch.org/docs/stable/quantization.html

Chapter 3

Tropical Algebra in Python

In Section 3.1, we provide short description of computational tools that help with tropical mathematics. Then in Section 3.2 we present our tropical framework in Python. Section 3.5 contains examples of tropical computing with our code.

3.1 Useful Resources

- polymake [2], which is open-source software for research in polyhedral geometry. Among many other things, it can deal with polytopes and tropical hypersurfaces. A substantial contribution to polymake is the extension a-tint for tropical intersection theory [7].
- SageMath [14], which is open-source mathematics software with package Gfan [8] for computing Grobner fans and tropical varieties.
- Macaulay2 [3], a computer algebra system. Especially useful for us are the Polyhedra and Tropical packages.

For our purposes, we needed a python framework that allowed us to perform algebraic operations on tropical polynomials, implement algorithms for converting a tropical polynomial into a neural network and vice versa, and calculate tropical hypersurfaces and their subdivisions.

3.2 Tropical Polynomials in Python

In our github repo [16] we provide an implementation of Tropical and TropicalPolynomial classes describing tropical numbers and tropical polynomials. A tropical polynomail is internally represented by a dictionary whose values are monomials and keys are tuples of coefficients without a free term. We equipped these classes with simple algebraic operations, such as addition and multiplication, printing in pretty form, and evaluating in point. We also provide useful properties as minification of a polynomial in the sense of removing redundant monomials(see Remark 2.1), plotting dual subdivision corresponding to polynomial in two variables, conversion a tropical polynomial into a neural network and vice versa.

3.2.1 Class definitions

```python
class Tropical:
    def __init__(self, val)
    # Arithmetic
    def __add__(self, other)
    def __mul__(self, other)
    def __pow__(self, other)
    # Other
    def __abs__(self)
    def __str__(self)
    def __repr__(self)
    def __float__(self)
```

3.1: Tropical number class

```python
class TropicalPolynomial:
    def __init__(self, monoms)
    """Initialize polynomial as a dictionary whose values are monomials, and keys are tuples of coefficients without a free term"""
    def __eq__(self, other)
    def __add__(self, other)
    def __mul__(self, other)
    def __pow__(self, other)
    def __str__(self)
    def evaluate(self, point)
    """Evaluate the polynomial at a given point or points"""
    def minify(self)
    """Eliminate redundant monomials from polynomial"""
    def poly_to_str(self):
    """Return polynomial in polymake syntax"""
    def plot_dual_sub(self, color)
    """Visualize the dual subdivision of tropical hypersurface"""
```

3.2: Tropical polynomial class

3.3 Tropical Polynomial Minification

Definition 3.1. We say that the monomial $c_0^i \odot x_1^{c_1^i} \odot ... \odot x_n^{c_n^i}$ in

$$p(x_1, x_2, ..., x_n) = \bigoplus_{i=1}^{k} a_0^i \odot x_1^{a_1^i} \odot ... \odot x_n^{a_n^i} \oplus c_0^i \odot x_1^{c_1^i} \odot ... \odot x_n^{c_n^i}$$

is redundant if for all $(x_1, ..., x_n) \in R^n$

$$c_0 + c_1 x_1 + c_n x_n \leq \max_{1 \leq i \leq n} (a_0^i + a_1^i x_1 + ... + a_n^i x_n)$$

This means that the value of the polynomial at no point equals to the value of this and only this monomial.

Theorem 3.2 ([9], Theorem 3.11)**.** *Let*

$$p(x_1, x_2, ..., x_n) = \oplus_{i=1}^{k} a_0^i \odot x_1^{a_1^i} \odot ... \odot x_n^{a_n^i} \oplus c_0^i \odot x_1^{c_1^i} \odot ... \odot x_n^{c_n^i}$$

The monomial $c_0^i \odot x_1^{c_1^i} \odot ... \odot x_n^{c_n^i}$ is redundant if and only if $(c_0, ..., c_n)$ is contained in

$$C = \bigcup_{M=1}^{\infty} \text{Conv} \left(\bigcup_{i=1}^{k} \{(a_0^i, ..., a_n^i)\} \cup \bigcup_{i=1}^{k} \{(a_0^i - M, ..., a_n^i)\} \right)$$

To find such monomials, we consider the dual subdivision of our polynomial, where the vertices correspond to monomials. If we can remove vertex and the subdivision remains unchanged, then monomial corresponding to this vertex is redundant.

3.3.1 Implementation

We add minification functionality as *minify* method for TropicalPolynomial class. Given tropical polynomial, we obtain it's subdivision via *polymake*. After that we create polyhedral complex induced by the cells of the subdivision. Vertices of this complex define irredundant monomials.

We also added the ability to visualize tropical zero curve and it's dual subdivision as *plot_curve* and *plot_dual_sub* methods for TropicalPolynomial class.

We use JuPyMake [6] as a basic interface to call polymake from python.

3.3.2 Example

For the polynomial $f = 1 \odot a \odot b \oplus a^3 \odot b \oplus 1 \odot a^2 \oplus 1 \odot b^2 \oplus 1 \odot a^2 \odot b^2 \oplus 1 \odot a^4 \odot b^2 \oplus 2 \odot a^3 \odot b^4 \oplus 3 \odot a^3 \odot b^5$, we calculate its minified version and visualize the dual subdivision with and without redundant monomials.

```
> f = TropicalPolynomial
([[1,2,0],[1,0,2],[1,1,1],[2,3,4],[3,3,5],[1,4,2],[1,2,2],[0,3,1]])
> print(f)
$1 \odot a \odot b \oplus a^3 \odot b \oplus 1 \odot a^2 \oplus 1 \odot b^2 \oplus 1 \odot a^2 \odot b^2 \oplus 1 \odot a^4 \odot b^2 \oplus 2 \odot a^3 \odot b^4 \oplus 3 \odot a^3 \odot b^5$
> print(f.minify())
$1 \odot a^2 \oplus 1 \odot b^2 \oplus 1 \odot a^4 \odot b^2 \oplus 3 \odot a^3 \odot b^5$
```

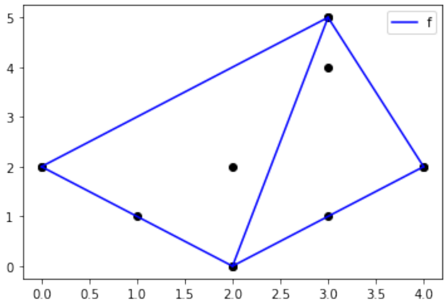


FIGURE 3.1: Dual subdivision with redundant monomials

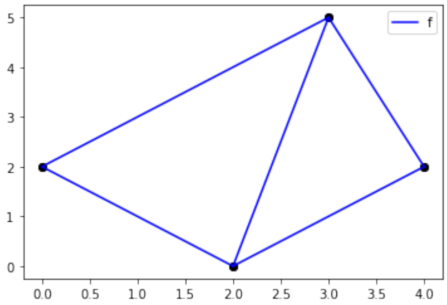


FIGURE 3.2: Dual subdivision without redundant monomials

3.4 Tropical Polynomial Factorization

In the case of one variable, we can formulate

Theorem 3.3 ([5], Fundamental Theorem of Tropical Algebra)**.** *Let $f(x) = a_n x^n \oplus a_{n-1}x^{n-1} \oplus \cdots \oplus a_r x^r$ be tropical polynomial without redundant monomials. Then $f(x)$ can be written uniquely as the product of linear factors $a_n x^r (x \oplus d_n)(x \oplus d_{n-1}) \ldots (x \oplus d_{r+1})$, where $d_i = a_{i-1} - a_i$. In other words, the roots of f(x) are the differences between consecutive coefficients.*

Although we can factor every single-variable polynomial into linear factors, it is impossible to do the same for polynomials of two or more variables. Tropical polynomials in more than one variable can have more than one possible factorization

$$x^2y \oplus xy^2 \oplus x^2 \oplus xy \oplus y^2 \oplus x \oplus y = (x \oplus 0)(y \oplus 0)(x \oplus y) = (x \oplus y \oplus 0)(xy \oplus x \oplus y)$$

Grigg [4] proved that factorization of tropical polynomials in two or more variables is NP-complete and presented an algorithm of constructing factorization using the structure of the tropical zero locus. However, he only gave a high-level description of the algorithm and we are not aware of any implementation of it.

3.4.1 Implementation

We provide factorization of two-dimensional polynomials as *factorize* method for TropicalPolynomial class. It returns a list of all possible divisors of the polynomial except trivial. For each of the divisors, for convenience, the result of dividing the original polynomial by it is also returned.

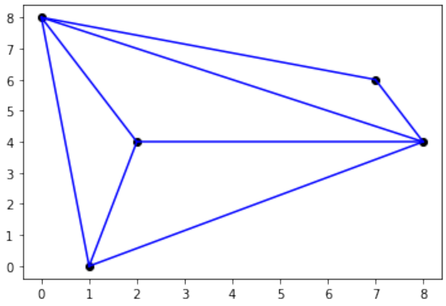
The procedure for finding the polynomial divisor consists of the following steps

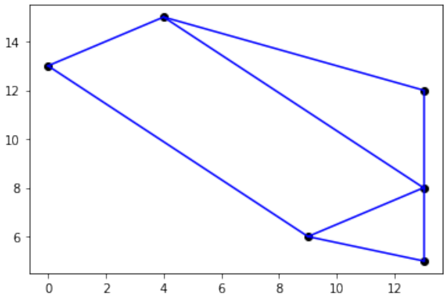
1. having a balanced tropical curve, choose its arbitrary balanced sub-curve
2. find the degrees of monomials defining this sub-curve using balancing condition
3. find free terms of this monomials using nodes of original curve

3.5 Examples of tropical computing

Let's create first polynomial $h = 4 \odot a \oplus 4 \odot a^2 \odot b^4 \oplus 8 \odot a^8 \odot b^4 \oplus a^7 \odot b^6 \oplus b^8$ and second polynomial $e = 9 \odot a^{13} \odot b^{12} \oplus 2 \odot b^{13} \oplus 9 \odot a^4 \odot b^{15} \oplus 4 \odot a^{13} \odot b^5 \oplus 2 \odot a^9 \odot b^6 \oplus 9 \odot a^{13} \odot b^8$.

```
> f = TropicalPolynomial([[4,2,4],[2,2,5],[0,0,8]])
> g = TropicalPolynomial([[7,7,4],[8,8,4],[0,7,6],[4,1,0]])
> e = TropicalPolynomial([[2,9,10],[9,13,12],[4,4,12],[2,0,13],
[9,4,15],[4,13,5],[2,9,6],[9,13,8],[4,13,9]])
> h = f + g
> print(h)
4 ⊙ a ⊕ 4 ⊙ a^2 ⊙ b^4 ⊕ 8 ⊙ a^8 ⊙ b^4 ⊕ a^7 ⊙ b^6 ⊕ b^8
> h.plot_dual_sub()
> e.plot_dual_sub()
```





We can calculate product of this polynomials, plot dual subdivision and tropical curve

```
> k = (h * e).minify()
> print(k)
```

$$4 \odot a^{10} \odot b \oplus 11 \odot a^{5} \odot b^{10} \oplus 15 \odot a^{21} \odot b^{11} \oplus 7 \odot a^{20} \odot b^{13} \oplus 11 \odot a^{14} \odot b^{3} \oplus 6 \odot a^{14} \oplus$$
$$15 \odot a^{12} \odot b^{14} \oplus 11 \odot a^{6} \odot b^{14} \oplus 10 \odot a^{21} \odot b^{4} \oplus b^{16} \oplus 7 \odot a^{11} \odot b^{16} \oplus 15 \odot a^{21} \odot b^{7} \oplus$$
$$7 \odot a^{4} \odot b^{18} \oplus 4 \odot a \odot b^{8}$$

```
> k.plot_dual_sub(color='blue')
> k.plot_curve(color='red')
```

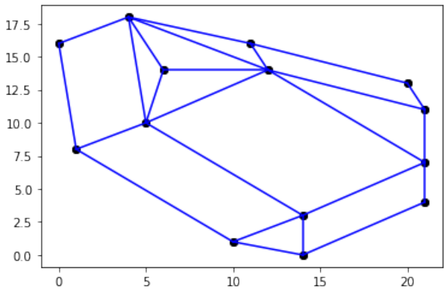


FIGURE 3.3: Dual subdivision

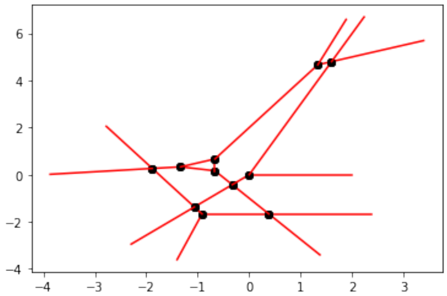


FIGURE 3.4: Tropical curve

Next, we can factor the resulting product and look at the tropical curves of the factors

```
> factors = poly.factorize()
> print(len(factors))
3
> k.plot_curve('red', length=3)
> factors[0].plot_curve('green', length=7)
> factors[1].plot_curve('brown', length=5)
> factors[2].plot_curve('orange', length=4)
```

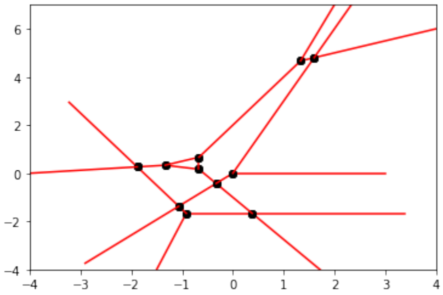


FIGURE 3.5: Tropical curve of the polynomaial

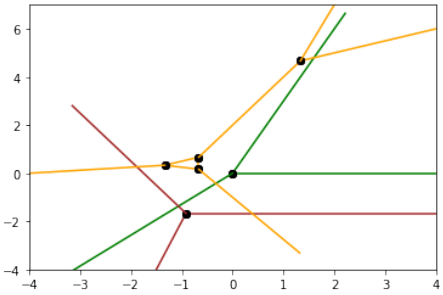


FIGURE 3.6: Tropical curves of its factors

Chapter 4

Tropical Polynomials and Neural Networks

Assumption 1. In this section, we consider the class of neural networks with

- integer-valued weight matrices
- real-valued bias vectors
- ReLU activation functions

Zhang proved following theorem

Theorem 4.1 ([17], Theorem 5.4)**.** *ReLU neural networks and tropical rational functions are equivalent in the following sense.*

- *Let $v : \mathbb{R}^d \to \mathbb{R}^p$. Then v can be defined by a tropical rational function if and only if v is a feedforward neural network under assumptions 1*
- *A tropical rational function $F \oslash G$ can be represented as an L-layer network with*

  $$L \leq \max 1 + [\log_2(r_F)], 1 + [\log_2(r_G)], [\log_2(d+1)] + 1$$

  where r_F and r_G are the number of monomials in the tropical polynomials F and G respectively.

From the proof of this theorem, we extract an algorithm for converting a neural network into a tropical rational function in Chapter 4.1 and the inverse algorithm in Chapter 4.2. We also provide details of implementation and show that when applied sequentially, these algorithms return non-optimal results.

4.1 Network to Tropical Rational Function

In this section we will define inductive procedure of conversion ReLU-network into quotient of two tropical polynomials. After that we transform this procedure into an algorithm and provide details of implementation.

Every matrix A can be decomposed as a difference of two non-negative integer-valued matrices $A = A_+ - A_-$ by taking $a_{ij}^+ = \max\{a_{ij}, 0\}$ and $a_{ij}^- = \max\{-a_{ij}, 0\}$.

Proposition 4.2. *Suppose $v(x) = F(x) \oslash G(x)$ is a tropical rational map, $\rho(x) = Ax+b$ a preactivation function with integer-valued matrix A and real-valued vector b, and $\sigma(x)$ a ReLU activation function with threshold t. Then there exist tropical polynomials $F'(x)$ and $G'(x)$ that*

$$\sigma \circ \rho \circ v(x) = F'(x) \oslash G'(x)$$

Proof.

$$\begin{aligned}
\rho(v(\boldsymbol{x})) &= A(F(x) - G(x)) + b \\
&= AF(x) - AG(x) + b \\
&= (A_+F(x) - A_-F(x)) - (A_+G(x) - A_-G(x)) + b \\
&= (A_+F(x) + A_-G(x) + b) - (A_-F(x) + A_+G(x)) \\
&= H'(x) - G'(x)
\end{aligned}$$

where $H'(x) \coloneqq A_+F(x) + A_-G(x) + b$ and $G'(x) \coloneqq A_-F(x) + A_+G(x)$ are tropical polynomial maps. Then

$$\begin{aligned}
\sigma(\rho(v(x))) &= \max\{H'(x) - G'(x), \boldsymbol{t}\} \\
&= \max\{H'(x), G'(x) + t\} - G'(x) \\
&= F'(x) - G'(x)
\end{aligned}$$

where $F'(x) \coloneqq \max\{H'(x), G'(x) + t\}$ is a tropical polynomial map. □

Alternatively, denoting $F_i^{(l)}$, $G_i^{(l)}$, $H_i^{(l)}$, $b_i^{(l)}$, and $t_i^{(l)}$ as the i^{th} coordinate of $F^{(l)}$, $G^{(l)}$, $H^{(l)}$, $b^{(l)}$, and $t^{(l)}$ respectively, we can rewrite the above in tropical notation:

$$\begin{align}
v^{(l+1)}(x) &= \sigma(\rho(v^{(l)}(x))) = F^{(l+1)}(x) \oslash G^{(l+1)}(x) \tag{4.1}\\
H_i^{(l+1)}(x) &= \left[\bigodot_{j=1}^{n} (F_j^{(l)}(x))^{(A_+^{(l+1)})_{ij}}\right] \odot \left[\bigodot_{j=1}^{n} (G_j^{(l)}(x))^{(A_-^{(l+1)})_{ij}}\right] \odot b_i^{(l+1)} \tag{4.2}\\
G_i^{(l+1)}(x) &= \left[\bigodot_{j=1}^{n} (F_j^{(l)}(x))^{(A_-^{(l+1)})_{ij}}\right] \odot \left[\bigodot_{j=1}^{n} (G_j^{(l)}(x))^{(A_+^{(l+1)})_{ij}}\right] \tag{4.3}\\
F_i^{(l+1)}(x) &= H_i^{(l+1)}(x) \oplus G_i^{(l+1)}(x) \odot t_i^{(l+1)} \tag{4.4}
\end{align}$$

where A_{ij} represents the $(i,j)^{th}$ coordinate of matrix A.

From the definition of a neural network

$$v^{(1)}(x) = \max\{Ax + b, t\} = \max\{A_+x - A_-x + b, t\} = \max\{A_+x + b, A_-x + t\} - A_-x$$

then $F^{(1)}(x) = \max\{A_+x + b, A_-x + t\}$ and $G^{(1)}(x) = A_-x$.

To satisfy this, $F^{(0)}$ should be initialized as tropical polynomial map $(f_1(x), \ldots, f_n(x))$, where each polynomial $f_i(x)$ has zero free term and zero coefficients except the i-th.

$G^{(0)}$ should be initialized as tropical polynomial map $(f_1(x), \ldots, f_n(x))$, where each polynomial $f_i(x)$ has zero free term and all zero coefficients.

4.1.1 Algorithm

We can transform recurrences 4.1-4.4 into an algorithm for converting a neural network into a tropical polynomial

Algorithm 1 Net2Trop algorithm

```
Input: $net$ – neural network, $t$ — ReLU parameter
Output: $f, h, g$ – tropical polynomials
function NET2TROP($net$, $t$)
    Initialize F and G
    for all layers in $net$ do
        create $H_{new}$ from F and G                    ▷ 4.2
        create $G_{new}$ from F and G                    ▷ 4.3
        update F from $H_{new}$ and $G_{new}$            ▷ 4.4
        update G from $G_{new}$
    end for
end function
```

4.1.2 Implementation

We provide an implementation of Algorithm 4.1.1 in our *tropical* framework. At the input, the algorithm takes a neural network in PyTorch, consists of linear layers with integer-valued weight matrices and real biases. We assume that ReLU function wraps each linear layer of our network except the last.

4.2 Tropical Polynomial to Network

In order to evaluate tropical polynomial at a point, we can, for example, find the value of each of the monomials, and then determine the maximum of the obtained values. These steps can be represented as a neural network: we will first construct a single layer computing the monomials, and then $\log_2(n)$ layers computing their maximum. Below we describe this construction in more details.

Given a tropical polynomial

$$f(x) = c_1 x^{\alpha_1} \oplus \cdots \oplus c_n x^{\alpha_n}$$

we can define

$$c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \qquad A = \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \vdots \\ \boldsymbol{\alpha}_n \end{bmatrix} = \begin{bmatrix} \alpha_1^1 & \dots & \alpha_1^n \\ \alpha_2^1 & \dots & \alpha_2^n \\ \vdots & \ddots & \vdots \\ \alpha_n^1 & \dots & \alpha_n^n \end{bmatrix} \tag{4.5}$$

Given a point $x = (x_1, \dots, x_n)$, we can calculate the vector of monomials values as $y = Ax + c$. Maximum of the coordinates of this vector is equal to the value of the polynomial f at the point x.

Maximum of two numbers y_1 and y_2 can be found by performing the following procedure

1. Finding product z between matrix $W_1 = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}$ and $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$
2. Finding $z' = \max(z, 0)$ coordinatewise
3. Product m of z' and $W_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$ will be equal to desired maximum

We can generalize this method to find the maximum of n-element vector y. For simplicity, suppose that $n = 2k$. Following procedure returns maximum between pairs of coordinates, i.e. $m = [\max(y_1, y_2), \max(y_3, y_4), \ldots, \max(y_{n-1}, y_n)]$

1. Finding product z between $W_1 = \underbrace{\left.\begin{pmatrix} \begin{bmatrix}1 & -1\\ 0 & 1\\ 0 & -1\end{bmatrix} & & 0 \\ & \ddots & \\ 0 & & \begin{bmatrix}1 & -1\\ 0 & 1\\ 0 & -1\end{bmatrix} \end{pmatrix}\right\}\text{3k rows}}_{\text{2k columns}}$ and $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$
2. Finding $z' = \max(z, 0)$ coordinatewise
3. Product of z' and $W_2 = \underbrace{\left.\begin{pmatrix} \begin{bmatrix}1\\ 1\\ -1\end{bmatrix} & & 0 \\ & \ddots & \\ 0 & & \begin{bmatrix}1\\ 1\\ -1\end{bmatrix} \end{pmatrix}\right\}\text{3k rows}}_{\text{k columns}}$

   gives us k-dimensional vector m.

In the case when $n = 2k + 1$ it is sufficient to change the matrices

$$W_1 = \underbrace{\left.\begin{pmatrix} \begin{bmatrix}1 & -1\\ 0 & 1\\ 0 & -1\end{bmatrix} & & 0 & 0 \\ & \ddots & & \\ 0 & & \begin{bmatrix}1 & -1\\ 0 & 1\\ 0 & -1\end{bmatrix} & 0 \\ 0 & \ldots & 0 & 1 \\ 0 & \ldots & 0 & -1 \end{pmatrix}\right\}\text{3k+2 rows}}_{\text{2k+1 columns}} \qquad W_2 = \underbrace{\left.\begin{pmatrix} \begin{bmatrix}1\\ 1\\ -1\end{bmatrix} & & 0 & 0 \\ & \ddots & & \\ 0 & & \begin{bmatrix}1\\ 1\\ -1\end{bmatrix} & 0 \\ 0 & \ldots & 0 & 1 \\ 0 & \ldots & 0 & -1 \end{pmatrix}\right\}\text{3k+2 rows}}_{\text{k+1 columns}}$$

then output vector will be $m = [\max(y_1, y_2), \max(y_3, y_4), \ldots, \max(y_{2k-1}, y_{2k}), y_{2k+1}]$

We can repeat this process using m as y and the output will get the desired maximum.

Example 4.1. *Find the maximum element of a vector* $y = \begin{bmatrix} -3 \\ 1 \\ 2 \\ 0 \\ 4 \end{bmatrix}$

1. $z = W_1 y = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix} y = \begin{bmatrix} -4 \\ 1 \\ -1 \\ 2 \\ 0 \\ 0 \\ 4 \\ -4 \end{bmatrix}$

2. $z' = \max(z, 0) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 2 \\ 0 \\ 0 \\ 4 \\ 0 \end{bmatrix}$

3. $z = W_1 m = W_1 W_2^T z' = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}^T z' = \begin{bmatrix} -1 \\ 2 \\ -2 \\ 4 \\ -4 \end{bmatrix}$

4. $z' = \max(z, 0) = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 4 \\ 0 \end{bmatrix}$

5. $z = W_1 m = W_1 W_2^T z' = \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}^T z' = \begin{bmatrix} -2 \\ 4 \\ -4 \end{bmatrix}$

6. $z' = \max(z, 0) = \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix}$

7. $m = W_2^T z' = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}^T z' = 4$

We can match each matrix multiplication with a linear layer and non-linear maximum with ReLU and construct corresponding neural network.

4.2.1 Algorithm

4.2.2 Implementation

We provide an implementation of Algorithm 4.1.1 in our *tropical* framework and it's extended version for conversion the quotient of two tropical maps.

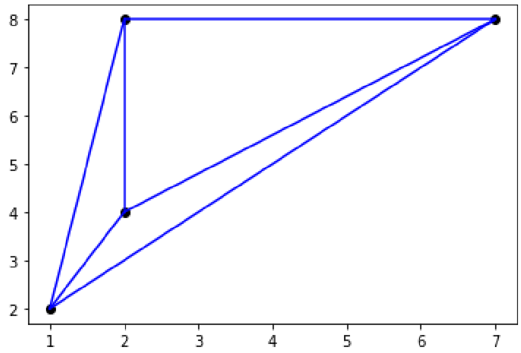
4.3 Examples and optimality

Let's generate small polynomial with random coefficients

Algorithm 2 Trop2Net algorithm

```
Input: $f$ – tropical polynomial
Output: $net$ – neural network
function TROP2NET($net$)
    Initialize empty $net$
    Initialize $W_2$ from polynomial coefficients                       ▷ 4.2
    Create $W_1$                                                        ▷ Step 1
    $W = W_1 \times W2$
    Add linear layer with weigth W and ReLU layer to $net$              ▷ Step 2
    while len of $W_2 > 3$ do
        Create $W_2$                                                    ▷ Step 3
        Create $W_1$
        $W = W_1 \times W2$
        Add linear layer with weigth W and ReLU layer to $net$
    end while
end function
```

```
> points = np.random.randint(10,size=(6,3)).tolist()
> b = TropicalPolynomial(points).minimize()
> print(b)
$4 \odot a \odot b^{2} \oplus 9 \odot a^{2} \odot b^{4} \oplus 1 \odot a^{2} \odot b^{8} \oplus a^{7} \odot b^{8}$
> b.plot_dual_sub()
```



We can convert this polynomial to neural network.

```
> b_net = PolyNet(b)
> b_net
  PolyNet(
      (linears): ModuleList(
        (0): Linear(in_features=2, out_features=6, bias=True)
        (1): Linear(in_features=6, out_features=3, bias=True)
        (2): Linear(in_features=3, out_features=1, bias=True)
      )
    )
```

We can perform inverse transformation of this network to quotient of two tropical polynomials

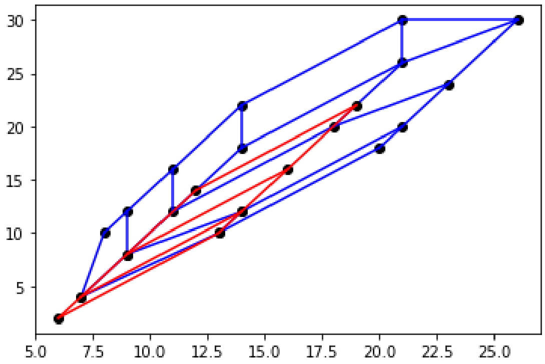
```
> h, g = convert_net_to_tropical(b_net)
> h_min = h.minimize()
> g_min = g.minimize()
> print(h_min)
```

$$-8.0 \odot a^{13} \odot b^{10} \oplus -7.0 \odot a^{8} \odot b^{10} \oplus -3.0 \odot a^{21} \odot b^{20} \oplus 15.0 \odot a^{18} \odot b^{20} \oplus 19.0 \odot a^{21} \odot b^{30} \oplus 18.0 \odot a^{26} \odot b^{30} \oplus 15.0 \odot a^{11} \odot b^{12} \oplus -3.0 \odot a^{14} \odot b^{12} \oplus -2.0 \odot a^{9} \odot b^{12} \oplus 19.0 \odot a^{14} \odot b^{22} \oplus 6.0 \odot a^{23} \odot b^{24} \oplus -4.0 \odot a^{7} \odot b^{4} \oplus 7.0 \odot a^{11} \odot b^{16} \oplus 27.0 \odot a^{21} \odot b^{26} \oplus -8.0 \odot a^{20} \odot b^{18} \oplus 27.0 \odot a^{14} \odot b^{18} \oplus 6.0 \odot a^{9} \odot b^{8}$$

```
> print(g_min)
```

$$-8.0 \odot a^{13} \odot b^{10} \oplus -3.0 \odot a^{14} \odot b^{12} \oplus 18.0 \odot a^{19} \odot b^{22} \oplus -8.0 \odot a^{6} \odot b^{2} \oplus 18.0 \odot a^{12} \odot b^{14} \oplus -3.0 \odot a^{7} \odot b^{4} \oplus 6.0 \odot a^{16} \odot b^{16} \oplus 6.0 \odot a^{9} \odot b^{8}$$

```
> h_min.plot_dual_sub(color='blue')
> g_min.plot_dual_sub(color='red')
```



Moreover, the quotient of the obtained polynomials gives the original polynomial b

```
> print((g_min*b).minimize())
```

$$-8.0 \odot a^{13} \odot b^{10} \oplus -7.0 \odot a^{8} \odot b^{10} \oplus -3.0 \odot a^{21} \odot b^{20} \oplus 15.0 \odot a^{18} \odot b^{20} \oplus 19.0 \odot a^{21} \odot b^{30} \oplus 18.0 \odot a^{26} \odot b^{30} \oplus 15.0 \odot a^{11} \odot b^{12} \oplus -3.0 \odot a^{14} \odot b^{12} \oplus -2.0 \odot a^{9} \odot b^{12} \oplus 19.0 \odot a^{14} \odot b^{22} \oplus 6.0 \odot a^{23} \odot b^{24} \oplus -4.0 \odot a^{7} \odot b^{4} \oplus 7.0 \odot a^{11} \odot b^{16} \oplus 27.0 \odot a^{21} \odot b^{26} \oplus -8.0 \odot a^{20} \odot b^{18} \oplus 27.0 \odot a^{14} \odot b^{18} \oplus 6.0 \odot a^{9} \odot b^{8}$$

```
> (g_min*b).minimize() == h_min
True
```

Thus, we can use the polynomial factorization, also provided by our library 3.4, to reduce the resulting polynomials in more compact form.

4.3.1 Analysis

From the previous example it is clear that the sequential application of the proposed algorithms can return object, which is functionally equivalent to the input, but with significantly more complex structure.

In the algorithm 4.1.1, the polynomial G_0 is initialized with zero coefficients, and from 4.3 they can become nonzero if and only if there are negative weights in the input network. However, the network obtained from the algorithm 4.2.1 will have negative weights regardless of the input polynomial.

The next example will also show us that algorithm 4.2.1 returns a network that is not optimal in terms of structure and number of layers.

Example 4.2. *Consider a polynomial* $a^4 \odot b^{10} \oplus a^{10} \odot b^4 \oplus a^9 \odot b^5 \oplus a^8 \odot b^6 \oplus a^7 \odot b^7 \oplus a^6 \odot b^8 \oplus a^5 \odot b^9$*. We can get the corresponding network using the algorithm 4.2.1*

```
    > b = TropicalPolynomial
    ([[0,4,10],[0,5,9],[0,6,8],[0,7,7],[0,8,6],[0,9,5],[0,10,4]])
    > b_net = PolyNet(b)
    > b_net
      PolyNet(
          (linears): ModuleList(
            (0): Linear(in_features=2, out_features=11, bias=True)
            (1): Linear(in_features=11, out_features=6, bias=True)
            (2): Linear(in_features=6, out_features=3, bias=True)
            (3): Linear(in_features=3, out_features=1, bias=True)
          )
)
```

However, we can represent this polynomial as a product $(a \oplus b) \odot (a \odot b^2 \oplus a^2 \odot b) \odot (a^2 \odot b^3 \oplus a^3 \odot b^2) \odot (a \odot b^4 \oplus a^4 \odot b)$ *and get a simpler network of just 3 layers*

1. *Create* $W_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 2 \\ 2 & 1 \\ 2 & 3 \\ 3 & 2 \\ 1 & 4 \\ 4 & 1 \end{pmatrix}$ *for calculating values each of monomials*

2. *Create* $W_1 = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$

3. *Create first linear layer with weight matrix* $W_1 W_2^T$

4. *Create ReLU layer*

5. *Create* $W_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix}$

6. *Create* $W_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

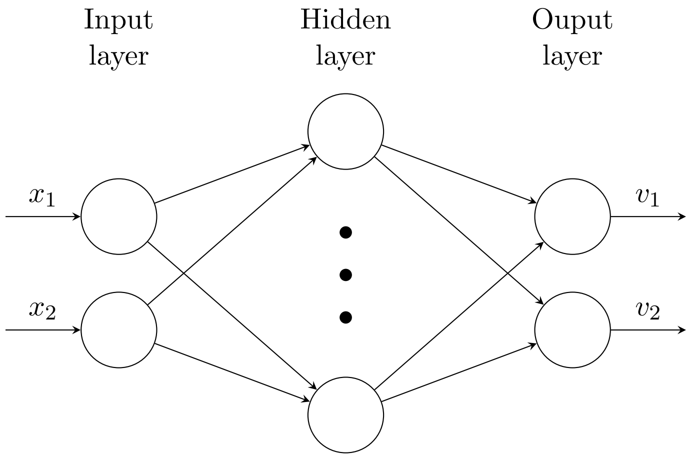
7. *Create second linear layer with weight matrix* $W_1 W_2^T$

Here we sequentially found the values of each of the monomials, then the values of each factor and then their product in a tropical sense.

Chapter 5

Decision Boundaries and Tropical Geometry

In order to obtain more insightful visualizations, we will narrow the class of neural networks in this section to the following architecture



Proposition 5.1. *For neural network* $v(x) = \begin{bmatrix} v_1(x) \\ v_2(x) \end{bmatrix} = \begin{bmatrix} F_1(x) \oslash G_1(x) \\ F_2(x) \oslash G_2(x) \end{bmatrix}$,

the decision boundary of v *is contained in some tropical hypersurface determined by* v*. Specifically*

$$\mathcal{B}(v) = \{x \in \mathbb{R}^d : v_1(x) = v_2(x)\} \subseteq \mathcal{T}(R),$$

where $R(x) = F_1(x) \odot G_2(x) \oplus F_2(x) \odot G_1(x)$*.*

Proof.

$$\begin{aligned}\mathcal{B}(v) &= \{x \in \mathbb{R}^d : v_1(x) = v_2(x)\} \\ &= \{x \in \mathbb{R}^d : F_1(x) - G_1(x) = F_2(x) - G_2(\boldsymbol{x})\} \\ &= \{x \in \mathbb{R}^d : F_1(x) + G_2(x)) = F_2(x) - G_2(\boldsymbol{x})\} \\ &= \{x \in \mathbb{R}^d : F_1(x) \odot G_2(x) = F_2(x) \odot G_1(x)\}\end{aligned}$$

For $x \in \mathcal{B}(v)$, $R(x) = F_1(x) \odot G_2(x) \oplus F_2(x) \odot G_1(x)$ has two of equal maximal monomial, so the value of R is attained by (at least) two of the constituent monomials - once in $F_1(x) \odot G_2(x)$ and once in $F_2(x) \odot G_1(x)$. Hence, x is contained in the tropical locus of $R(x)$, and so $\mathcal{B}(v) \subseteq \mathcal{T}(R)$. Moreover, geometrically $\mathcal{B}(v)$ is equal to $\mathcal{T}(R) - \mathcal{T}(F_1(x) \odot G_2(x)) - \mathcal{T}(F_2(x) \odot G_1(x)) + (\mathcal{T}(F_1(x) \odot G_2(x)) \cap \mathcal{T}(F_2(x) \odot G_1(x)))$ □

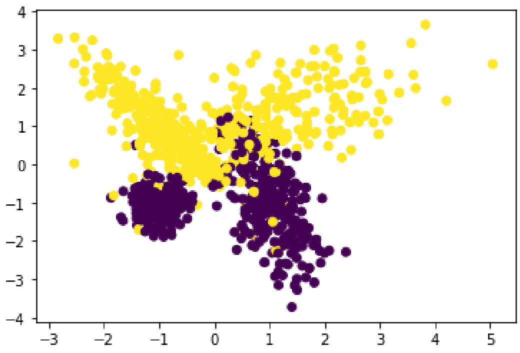
In the paper [1], the authors state Proposition 5.1 and discusses the decision boundary of a simple neural network with Linear-ReLU-Linear architecture, which also satisfy our Assumption 1. The main results obtained in the paper are based on an analysis not of the decision boundary itself, but of its superset $\mathcal{T}(R(x))$.

Below, using our *tropical* framework, we obtain an analytical expression directly for the decision boundary and demonstrate that it coincides with the estimated numerically. In addition, unlike [1], the network of our interest can consist of an arbitrary number of hidden layers.

5.1 Experiment

Let's create a standard classification task from *sklearn*

```
> X, Y = make_classification(n_samples=1000,n_features=2,n_classes=2,
  n_redundant=0,random_state=5)
```



and train a simple Linear-ReLU-Linear neural network with 4 neurons in hidden layer.

After training, we can convert our neural network into a tropical rational function defined by 4 tropical polynomials $H_1(x), G_1(x), H_2(x), G_2(x)$. We use the notation H instead F because the last layer of our network is not ReLU.

In this experiment we get

$$H_1(x) = 20.0 \oplus 72.0 \odot x_1^{398} \odot x_2^{354} \oplus 88.0 \odot x_1^{374} \odot x_2^{306} \oplus 4.0 \odot x_1^{24} \odot x_2^{48}$$
$$G_1(x) = 247.0 \odot x_1^{62} \odot x_2^{314} \oplus -128.0 \odot x_1^{562} \odot x_2^{564} \oplus x_1^{24} \odot x_2^{48} \oplus -375.0 \odot x_1^{524} \odot x_2^{298}$$
$$H_2(x) = 246.0 \odot x_1^{80} \odot x_2^{350} \oplus -84.0 \odot x_1^{520} \odot x_2^{570} \oplus -331.0 \odot x_1^{482} \odot x_2^{304} \oplus -1.0 \odot x_1^{42} \odot x_2^{84}$$
$$G_1(x) = 28.0 \oplus 100.0 \odot x_1^{396} \odot x_2^{324} \oplus x_1^{42} \odot x_2^{84} \oplus 72.0 \odot x_1^{438} \odot x_2^{408}$$

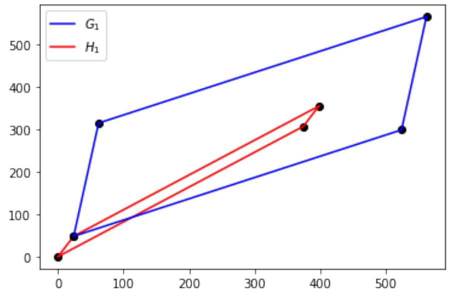


FIGURE 5.1: Dual subdivision of H_1 and G_1

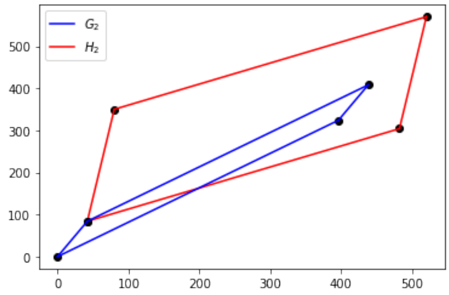


FIGURE 5.2: Dual subdivision of H_2 and G_2

Figure 5.1 shows the dual subdivisions for the tropical polynomial H_1 and G_1 and figure 5.1 for the tropical polynomial H_2 and G_2. It's interesting that H_1 looks very similar to G_2 and G_1 to H_2 and this similarity will remain in case of more complex polynomials.

Next, we can construct auxiliary polynomials

$$H_1(x) \odot G_2(x) = 48.0 \oplus 188.0 \odot a^{770} \odot b^{630} \oplus 4.0 \odot a^{66} \odot b^{132} \oplus 144.0 \odot a^{836} \odot b^{762}$$
$$H_2(x) \odot G_1(x) = -706 \odot a^{1006} \odot b^{602} \oplus -1 \odot a^{66} \odot b^{132} \oplus -212 \odot a^{1082} \odot b^{1134} \oplus 493 \odot a^{142} \odot b^{664}$$

$$\begin{aligned} R(x) = H_1(x) \odot G_2(x) \oplus F_2(x) \odot G_1(x) =& 48.0 \oplus 188.0 \odot a^{770} \odot b^{630} \oplus -706.0 \odot a^{1006} \odot b^{602} \oplus \\ & 144.0 \odot a^{836} \odot b^{762} \oplus -212.0 \odot a^{1082} \odot b^{1134} \oplus \\ & 493.0 \odot a^{142} \odot b^{664} \end{aligned}$$

Figure 5.3 shows the dual subdivisions for the tropical polynomials $H_1(x) \odot G_2(x)$, $H_2(x) \odot G_1(x)$ and R_{min} and figure 5.4 shows their tropical zero locus.

In both pictures, part of the R_{min} curve coincides with its components curves; however, we are primarily interested in its non-coincident blue part.

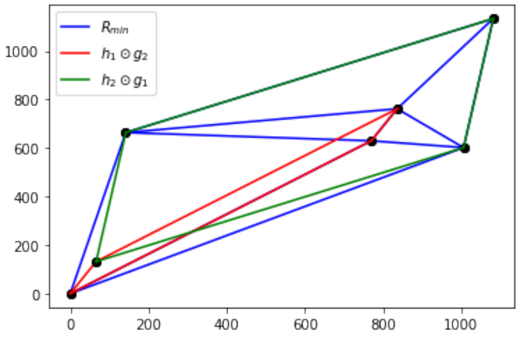


FIGURE 5.3: Dual subdivision of R_{min} and its components

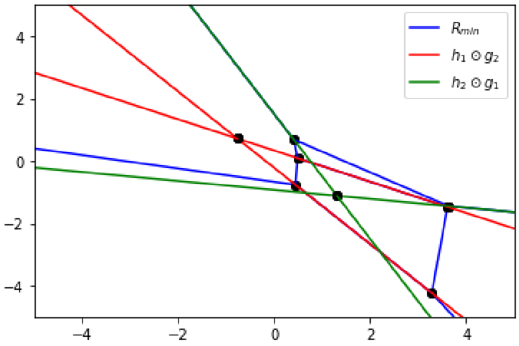


FIGURE 5.4: Tropical curves of R_{min} and its components

Let's visualize decision boundary of a trained neural network. To do this, calculate its predictions at each point of $[-5,5]^2$ square with 0.0005 step.

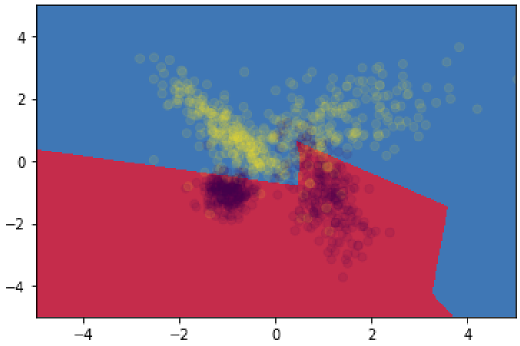


FIGURE 5.5: Real decision boundary of trained network

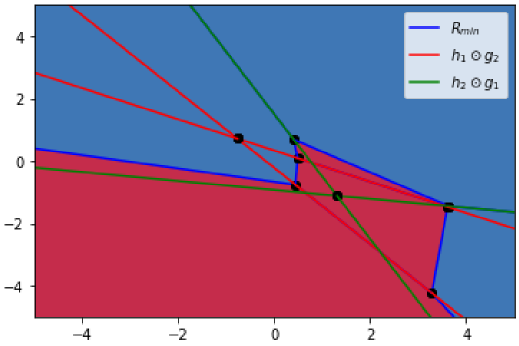


FIGURE 5.6: Real decision boundary and tropical curves

Figure 5.6 shows that the part of the R_{min} tropical curve, that does not coincide with the $H_1(x) \odot G_2(x)$ and $H_2(x) \odot G_1(x)$ curves, exactly matches with the empirical decision boundary. Our framework also provides an analytical representation of this boundary as union of rays and segments.

It is important to note that we can get similar boundaries for any neural network under Assumption 1, regardless of the number of layers and input features. However, in case of multiclass classification, the approach needs to be slightly changed by comparing classes in pairs.

We have conducted a similar experiment with a more complex Linear-ReLU-Linear-ReLU-Linear neural network with 4 and 3 neurons in hidden layers.

Below are visualizations similar to first experiment

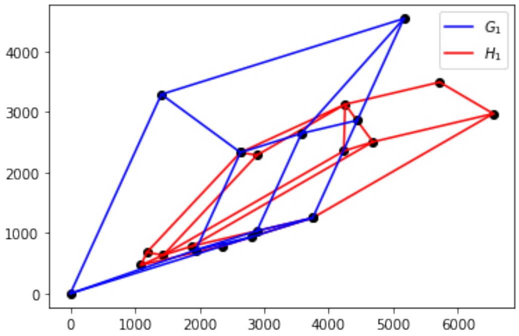


FIGURE 5.7: Dual subdivision of H_1 and G_1

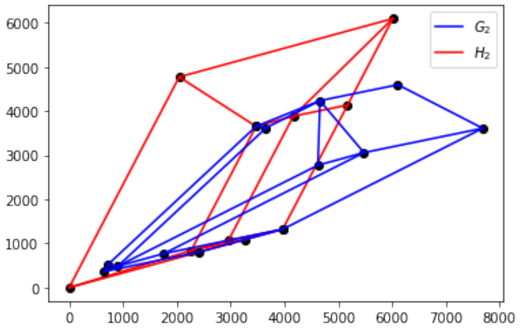


FIGURE 5.8: Dual subdivision of H_2 and G_2

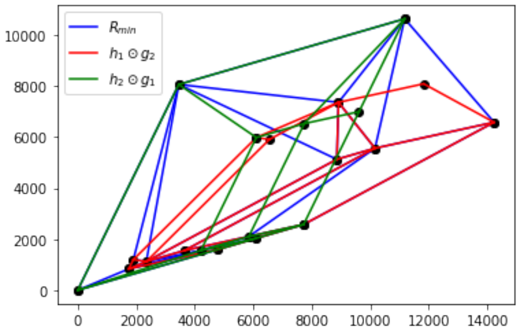


FIGURE 5.9: Dual subdivision of R_{min} and its components

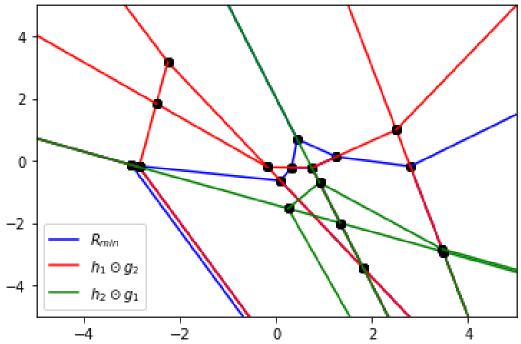


FIGURE 5.10: Tropical curves of R_{min} and its components

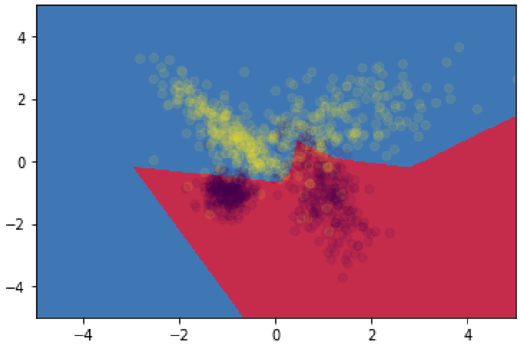


FIGURE 5.11: Real decision boundary of trained network

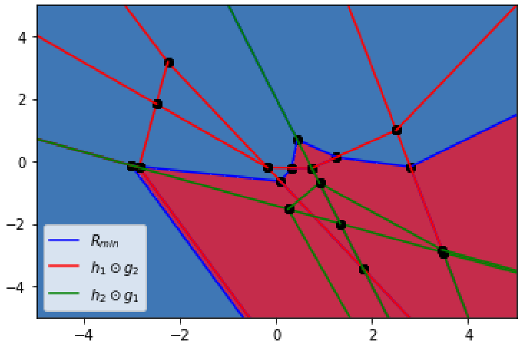


FIGURE 5.12: Real decision boundary and tropical curves

Chapter 6

Conclusion

We describe our *tropical* python framework [16], which provides a convenient way to perform simple arithmetic operations in a tropical semiring, construct tropical curves and their dual subdivisions, simplify the structure of tropical polynomials removing redundant monomials and perform factorization in some cases. We use this framework for the implementation of conversion algorithms between certain family of ReLU neural networks and tropical rational functions and vice versa and show their suboptimality. We also make use of the functionality it provides, in order to visualize the decision boundary of a fully connected ReLU network, trained for a binary classification problem.

The main directions for subsequent research are improving the optimality of the conversion algorithms, their use for training, pruning and initialization of neural networks, as well as the study of the geometry of their decision boundaries; application of the proposed in the work approach for more complex, in particular convolutional, neural networks.

Bibliography

[1] Alfarra, M., Bibi, A., Hammoud, H., Gaafar, M., and Ghanem, B. (2020). On the decision boundaries of deep neural networks: A tropical geometry perspective. ArXiv e-prints: 2002.08838.

[2] Gawrilow, E. and Joswig, M. (2000). `polymake`: a framework for analyzing convex polytopes. In *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, volume 29 of *DMV Sem.*, pages 43–73. Birkhäuser, Basel.

[3] Grayson, D. R. and Stillman, M. E. (2009). *Macaulay2: a software system for research in algebraic geometry*. http://www.math.uiuc.edu/Macaulay2/.

[4] Grigg, N. (2007). Factorization of tropical polynomials in one and several variables. Available at http://archive.nathangrigg.net/papers/honors-thesis.pdf.

[5] Grigg, N. and Manwaring, N. (2007). An elementary proof of the fundamental theorem of tropical algebra. ArXiv e-prints: 0707.2591.

[6] Gutsche, S. (2017). A small python wrapper for the polymake shell. https://github.com/sebasguts/JuPyMake.

[7] Hampe, S. (2014). a-tint: A polymake extension for algorithmic tropical intersection theory. *European Journal of Combinatorics*, 36:579–607.

[8] Jensen, A. N. (2005). Gfan, a software system for grobner fans and tropical varieties. Available at http://home.imf.au.dk/jensen/software/gfan/gfan.html.

[9] Kališnik, S. (2018). Tropical coordinates on the space of persistence barcodes. *Foundations of Computational Mathematics*, 19(1):101–129. http://skalisnikver.web.wesleyan.edu/StanfordPhD.pdf.

[10] Krivulin, N. (2014). Tropical optimization problems. ArXiv e-prints: 1408.0313.

[11] Maclagan, D. and Sturmfels, B. (2015). *Introduction to Tropical Geometry*. American Mathematical Society.

[12] Monod, A., Lin, B., Yoshida, R., and Kang, Q. (2018). Tropical geometry of phylogenetic tree space: A statistical perspective. ArXiv e-prints: 1805.12400.

[13] Morrison, R. (2019). Tropical geometry. ArXiv e-prints: 1908.07012.

[14] Stein, W. (2008). *Sage: Open Source Mathematical Software (Version 2.10.2)*. The Sage Group. http://www.sagemath.org.

[15] Tran, N. M. and Yu, J. (2015). Product-mix auctions and tropical geometry. ArXiv e-prints: 1505.05737.

[16] Zakharenkov, A. (2020). tropical. https://github.com/zachanton/tropical.

[17] Zhang, L., Naitzat, G., and Lim, L.-H. (2018). Tropical Geometry of Deep Neural Networks. *Proceedings of the 35th International Conference on Machine Learning.*