Nature-Inspired Machine Learning

Marissa Dominijanni

University at Buffalo

2019 May 03

Euclidean Neural Maps

Motivation

- Self-organizing maps (SOM) are a class of neural network models which utilize competitive learning and self-organization to learn dimensionality reduction
- SOM have a fundamental limitation in terms of the dimensionality of the feature space they can work with
- We propose a new model called a Euclidean neural map (ENM) with the intent of tractably generalizing SOM to high-dimensional datasets
- An ENM is parameterized by a set of "neural particles" which float in the output space and "bend space" around themselves orthogonally to the output space with the same dimensionality as the feature space

Self-Organizing Maps

- SOM is trained in an unsupervised manner to perform feature extraction, the process of discovering latent variables
- Neural network model in which neurons are not organized into layers, but instead are connected to each other, typically in a Cartesian grid
- Grid of neurons can also be connected in order to form an n-torus
- Structure seeks to mimic the topology of neurons found in the brain
- Each neuron contains a weight vector the same length as the dimensionality of the input space, position of the neuron corresponds to the value in the output space
- Iterate each time t up to a maximum of $lambda$

Self-Organizing Maps (cont.)

- Best Matching Unit (BMU) is computed as minimizing the distance d between the provided input x and the neuron weight w:

$$u = \arg\min_{v} d\left(w_v - x\right)$$

- SOM update rule typically takes the following form:

$$w_v \longleftarrow w_v + \left(p - w_v\right)\theta\left(v, u, t\right)\alpha\left(t\right)$$

$$\alpha\left(t\right) = \alpha_0 \exp\left(-\frac{t}{\lambda}\right)$$

$$\sigma\left(t\right) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right)$$

$$\theta\left(v, u, t\right) = \exp\left(-\frac{\|u - v\|_2^2}{2\sigma^2\left(t\right)}\right)$$

Self-Organizing Maps – Learning Procedure

1. Randomize the weight vectors of each neuron in the map, and set $t = 0$
2. Randomly draw a sample $x \sim P$
3. Compute the BMU as $u = \arg\min_v \|w_v - x\|_2^2$
4. For each neuron v, update the weight as

$$w_v \longleftarrow w_v + (p - w_v)\,\theta\,(v, u, t)\,\alpha\,(t)$$

5. Set $t = t + 1$
6. If $t < \lambda$, goto step 2, otherwise stop

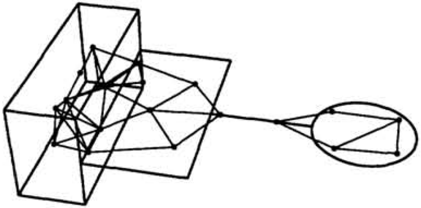
Comparison Between SOM and ENM

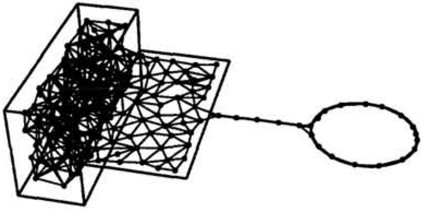
- Principal goal of ENM is to remain close to the original SOM model while eliminating issues of exponentially increasing complexity,
  - MNIST dataset has 784 input dimensions (pixels) and 10 output dimensions (digits)
  - In an SOM with 10 neurons along each output dimension, 10 billion neurons and 7.84 trillion parameters would be required
- We seek to do this by reducing the number of neurons ("neural particles" in ENM terminology) required to represent a near-to equivalent amount of information
- Attempt to achieve this by using a comparatively small number of neural particles which "float" in a continuous output space, rather than defining a discrete output space defined by each neuron
- Capitalize on the notion that the activation of a BMU in an SOM should generally exert an influence which exponentially decays to surrounding neurons, so a few neural particles can "sculpt" an approximation

Comparison Between SOM and ENM (cont.)

- Unlike in an SOM, neural particles are not evenly distributed in a range of the output space, but instead travel to “areas of activation” so more dynamic areas are more complexly represented
- The aforementioned continuous output space in an ENM (versus the discrete one in an SOM) means that inference cannot be performed by exhaustive search, but instead must be solved for via mathematical optimization (gradient descent in our implementation)
- Design decision which sacrifices this is also what allows the reduced number of parameters
- An ENM defines neural particles which can be thought of as exerting a field and is defined over an unbounded continuous space
- If the optimization procedure to find the BMU fails to find the global optimum, then the behavior of an ENM fundamentally diverges from that of an equivalent SOM, which could result in undefined behavior

Growing Neural Gas





https://papers.nips.cc/paper/893-a-growing-neural-gas-network-learns-topologies.pdf

- Growing neural gas (GNG) is another existing model which seeks to solve the high-dimensional intractability issue of SOM
- Unlike SOMs with topology fixed in the output space and morphed to fit in the input space, GNG topology is dynamically generated
- Follows the principle of Hebbian learning: if two neurons fire in tandem then the connection between them is strengthened

Growing Neural Gas – Learning Procedure

1. Generate two nodes a and b at random values in $\mathbb{R}^S$
2. Draw a random sample $\xi \sim P$ and find two nodes v_1 and v_2 such that v_1 is closest to (and v_2 is second closest to) ξ
3. For each edge e connected to v_1, increment its age α_e by 1
4. Increment the error for the BMU: $\epsilon_{v_1} \longleftarrow \epsilon_{v_1} + \|w_{v_1} - \xi\|_2^2$
5. Move v_1 and its neighbors towards ξ by η_b and η respectively: $w_v \longleftarrow w_v + \epsilon\left(\xi - w_v\right)$
6. If an edge connects v_1 and v_2, set its age to zero, $e\left(v_1, v_2\right) \longleftarrow 0$. Otherwise create such an edge with an age of zero.
7. For each edge $e \in E$, if $\alpha_e > \alpha_{\max}$ remove it. If after removing said edges any nodes have a degree of zero, remove them.
8. If the number of samples drawn is divisible by λ, insert a new node halfway between the node with maximum error and its neighbor
9. Decay all errors by multiplying them by the global decay rate of error
10. If a stopping criterion is not yet met, return to step 2 and repeat.

Comparison Between GNG and ENM

1. Both GNG and ENM have the goal of implementing an SOM-like algorithm while avoiding the issue of intractability
2. GNG uses a variable number of nodes, opposed to the fixed number of neural particles in ENM (“neural gas”, the algorithm GNG is directly based on also uses a fixed number of nodes)
3. GNG uses explicitly defined connections between nodes, whereas connectivity is defined implicitly in ENM, where the strength of a connection is inversely proportional to the distance between them (this connectivity decays exponentially)

Euclidean Neural Maps

- Main intuition is to start with an SOM, but change the rigid discrete output space to a continuous deformable space
- ENM is defined by a set of neural particles, each of which simultaneously exists in both the feature space $\mathbb{R}^S$ and the output space $\mathbb{R}^D$
- Each neural particle exists in the output space, where each neural particle bends space around itself orthogonally to the output space, with magnitude and direction determined by its representation in the feature space
- Deformation at a point in the output space corresponds to the unit's weight vector in an SOM
- Curvature extends to the output space surrounding a neural particle, decaying exponentially (with width parameterized on a particle-wise basis) back to its rest state

Euclidean Neural Maps (cont.)

- Main intuition is to start with an SOM, but change the rigid discrete output space to a continuous deformable space
- ENM is defined by a set of N neural particles, each of which simultaneously exists in both the feature space $\mathbb{R}^S$ and the output space $\mathbb{R}^D$
- Each neural particle exists in the output space, where each neural particle bends space around itself orthogonally to the output space, with magnitude and direction determined by its representation in the feature space
- Deformation at a point in the output space corresponds to the unit's weight vector in an SOM
- Curvature extends to the output space surrounding a neural particle, decaying exponentially (with width parameterized on a particle-wise basis) back to its rest state

Euclidean Neural Maps – Parameters & Hyperparameters

Parameters

$Q \in \mathbb{R}^{N\times S}$	Neural particle locations in feature space
$Z \in \mathbb{R}^{N\times D}$	Neural particle locations in output space
$K \in \mathbb{R}^{N\times 1}$	Curvature width of neural particles (used during inference)

Hyperparameters

- η and σ affect how the curvature (Q) is updated
- ζ and τ affect how the neural particles in Z are attracted to the area of activation in Z (we call the spread of neural particles in the output space "coverage").
- ϵ and υ affect how the neural particles in Z a repelled from each other in order to control any "clumping"
- η, ζ, and ϵ control the influence width of neural particles
- σ, τ, and υ control the step size of the update
- λ controls how quickly the curvature learning radius σ is decayed

Euclidean Neural Maps – Learning Procedure

1. Randomize the location of each neural particle in the feature space and output space, and assign a random curvature width in a narrow range to each neural particle
2. Randomly draw a sample $x \sim P$
3. Compute the activation point as

$$z^* = \arg\min_z \left\| \sum_{n \in N} \left[Q^{(n)} \odot \exp\left(-\frac{\left\| Z^{(n)} - z \right\|_2^2}{2{K^{(n)}}^2} \right) \right] - x \right\|_2^2$$

4. For each neural particle $n \in N$, update its induced extrinsic curvature as

$$Q^{(n)} \leftarrow Q^{(n)} - \eta \left(Q^{(n)} - x \right) \exp\left(-\frac{\left\| Z^{(n)} - z^* \right\|_2^2}{2\sigma^2} \right)$$

Euclidean Neural Maps – Learning Procedure (cont.)

5. For each neural particle $n \in N$, update its location in the output space as:

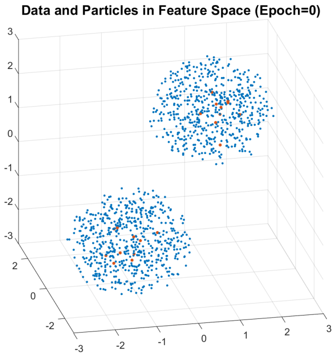
$$Z^{(n)} \leftarrow Z^{(n)} - \zeta \left(Z^{(n)} - z^* \right) \exp\left(-\frac{\left\| Z^{(n)} - z^* \right\|_2^2}{2\tau^2} \right) + \sum_{i \in N} \left[\epsilon \left(Z^{(n)} - Z^{(i)} \right) \exp\left(-\frac{\left\| Z^{(n)} - Z^{(i)} \right\|_2^2}{2\upsilon^2} \right) \right]$$

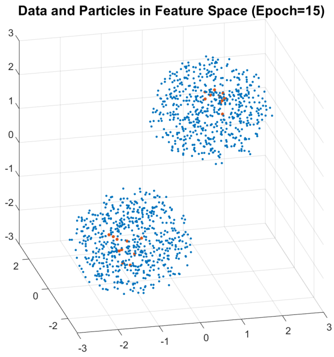
6. Reduce the curvature learning radius to ensure convergence $\sigma \leftarrow \frac{\sigma}{\lambda}$
7. If some stopping criterion is met, halt. Otherwise goto step 2.

Two-Spheres Test

- Synthetic dataset generated as "carving" two spheres from uniform random points
- 562 points in the sphere in the first octant $(+,+,+)$ and 600 in sphere in the seventh octant $(-,-,-)$
- In first test, neural particles placed within each sphere and labeled as one-hot vectors with Gaussian noise added to the labels as a "good start"
- In second test, neural particles placed in random positions with random labels as a regular start

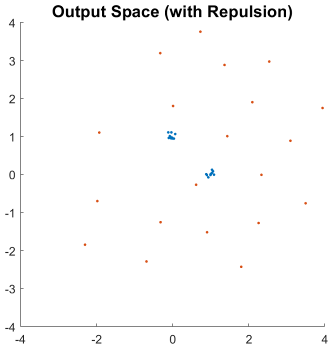
Two-Spheres Test (Good Start) – Input Space

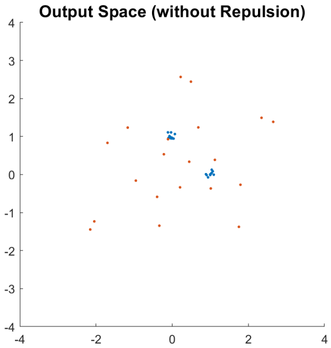




Blue	–	*Points in the Dataset*
Orange	–	*Neural Particles*

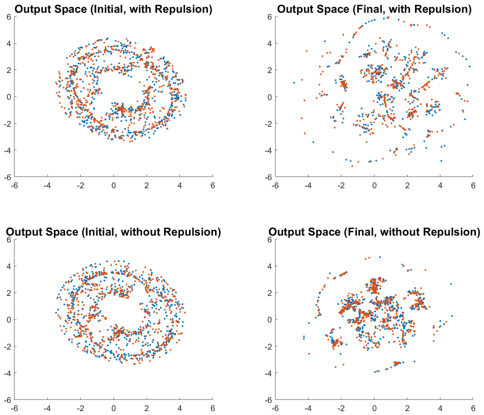
Two-Spheres Test (Good Start) – Output Space





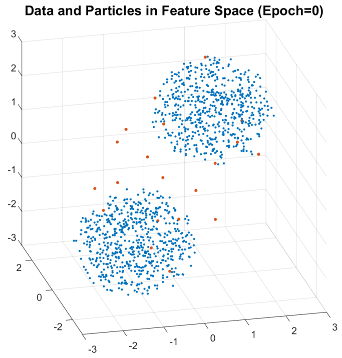
Blue	–	*Initial Location of Neural Particles*
Orange	–	*Final Location of Neural Particles*

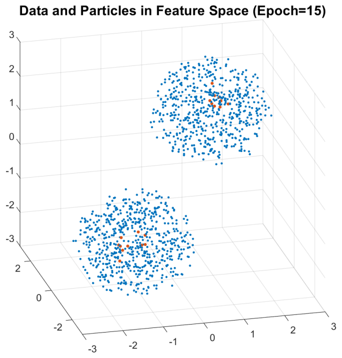
Two-Spheres Test (Good Start) – Activation Pattern



Blue	–	*Activations from First Octant Sphere*
Orange	–	*Activations from Seventh Octant Sphere*

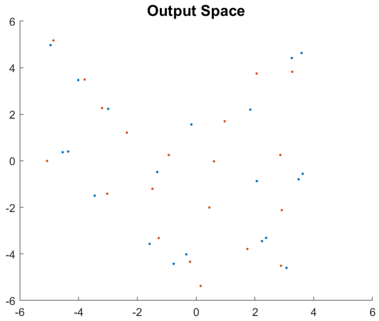
Two-Spheres Test (Random Start) – Input Space





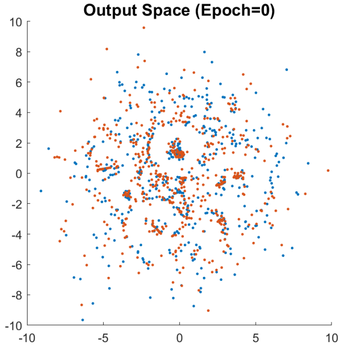
Blue	–	*Points in the Dataset*
Orange	–	*Neural Particles*

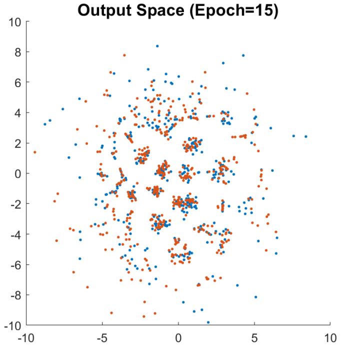
Two-Spheres Test (Random Start) – Output Space



Blue	–	*Initial Location of Neural Particles*
Orange	–	*Final Location of Neural Particles*

Two-Spheres Test (Random Start) – Activation Pattern



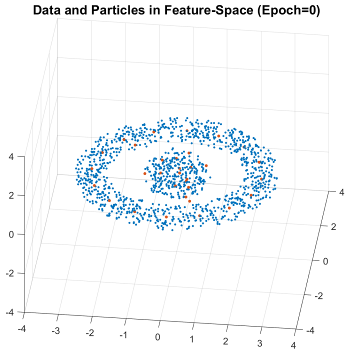


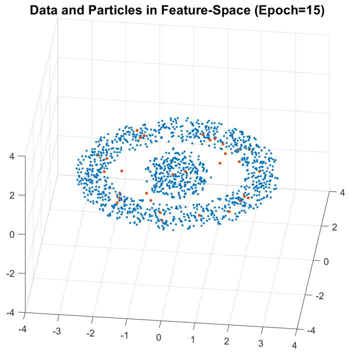
Blue	–	*Activations from First Octant Sphere*
Orange	–	*Activations from Seventh Octant Sphere*

Sphere-in-Torus Test

- Synthetic dataset generated as “carving” a sphere and a torus centered around the origin from uniform random points
- 293 points in the sphere 967 points in the torus
- Neural particles placed within the sphere and uniformly spaced along the circle defined by the major radius and labeled as one-hot vectors with Gaussian noise added to the labels as a “good start”

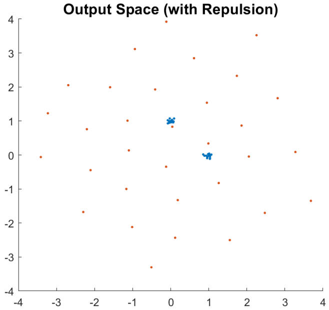
Sphere-in-Torus Test – Input Space

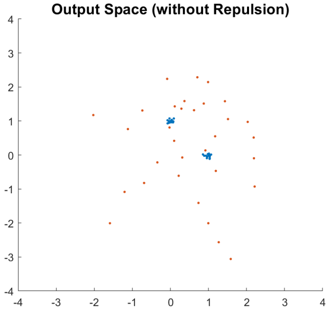




Blue	–	*Points in the Dataset*
Orange	–	*Neural Particles*

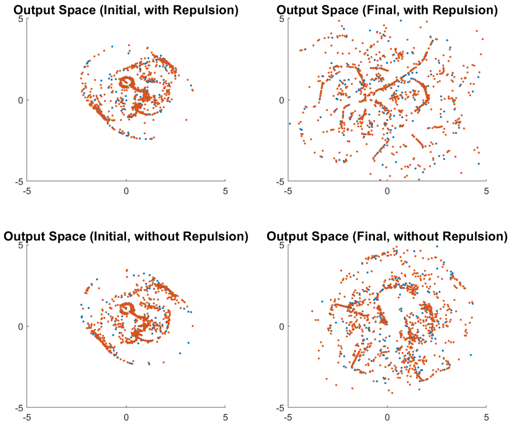
Sphere-in-Torus Test – Output Space





Blue	–	*Initial Location of Neural Particles*
Orange	–	*Final Location of Neural Particles*

Sphere-in-Torus – Activation Pattern



Blue – *Activations from the Sphere*

Orange – *Activations from the Torus*

Analysis of Results

- Stable convergence implies that the model is learning *something*, just not what we want it to (desirable final configuration is not stable)
- Lack of distinction in the output space between points in each cluster shows that it is unable to differentiate the two in this manner, even in the linearly separable case
- Lack of normalization means that overlapping neural particle neighborhoods to may lead to undesired interference, leading to local curvature with values greatly exceeding the magnitude of the values in the input space

Examples of Prior Work

- Comparison of gradient descent based optimization techniques (e.g. SGD, Adam, RMSProp) on a feedforward neural network
- Comparison of machine learning techniques for forensic handwriting analysis (Bayesian network, perceptron, and convolutional neural network with concatenated ReLU)
- Semi-supervised learning with deep generative models (comparing a VAE+MLP to a VAE with a second dense layer alongside the code layer sampled using the Gumbel-Softmax trick)
- Comparison of SVM and MLP (using SIFT features) and CNN for CIFAR-10 classification

Theoretical Properties for Neural Networks with Weight Matrices of Low Displacement Rank; Zhao, Liao, Wang et al.

arXiv:1703.00144

Low Displacement Rank Construction

- LDR construction is a method of imposing structure on the weights of a neural network to reduce space and computational complexity
- Accomplish this by defining the weight matrix for a layer of a neural network as a composition of structured matrices (or a single structured matrix in the case of a square weight matrix)
- A structured matrix is a kind of square matrix which can be procedural constructed from a smaller number of parameters (on the order of $\mathcal{O}\left(n\right)$ versus $\mathcal{O}\left(n^2\right)$ for a non-LDR matrix)
- These LDR neural networks can be viewed as using a kind of parameter sharing scheme

Benefits of LDR Neural Networks

- Fast matrix-vector multiplication on LDR matrices can reduce computational complexity from $\mathcal{O}\left(n^2\right)$ on regular neural networks to $\mathcal{O}\left(n \log n\right)$ or $\mathcal{O}\left(n \log^2 n\right)$
- More traditional methods such as heuristic weight-pruning produce irregular pruned networks that don't fit as well with SIMD computation models
- Heuristic weight-pruning requires additional retraining whereas LDR construction is a "train from scratch" method Up until now LDR networks perform with the same (or near same) accuracy as their non-LDR counterparts, but there has been a lack of theoretical analysis

Commonly Used Structured Matrices

Circulant $\left(c_{n-j+i \bmod n}\right)_{i,j=0}^{n-1}$

$$\begin{pmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & \cdots & c_2 \\ \vdots & c_1 & c_0 & \cdots & \vdots \\ c_{n-2} & \vdots & \vdots & \vdots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{pmatrix}$$

Cauchy $\left(1/(u_i - y_j)\right)_{i,j=0}^{n-1}$

$$\begin{pmatrix} 1/(u_0 - y_0) & \cdots & 1/(u_0 - y_{n-1}) \\ 1/(u_1 - y_0) & \cdots & 1/(u_1 - y_{n-1}) \\ \vdots & \vdots & \vdots \\ 1/(u_{n-1} - y_0) & \cdots & 1/(u_{n-1} - y_{n-1}) \end{pmatrix}$$

Toeplitz $\left(t_{i-j}\right)_{i,j=0}^{n-1}$

$$\begin{pmatrix} t_0 & t_{-1} & \cdots & t_{1-n} \\ t_1 & t_0 & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{pmatrix}$$

Hankel $\left(h_{i+j}\right)_{i,j=0}^{n-1}$

$$\begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_1 & h_2 & \cdots & h_n \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ h_{n-1} & h_n & \cdots & h_{2n-2} \end{pmatrix}$$

Vandermonde $\left(v_i^j\right)_{i,j=0}^{n-1}$

$$\begin{pmatrix} 1 & v_0 & \cdots & v_0^{n-1} \\ 1 & v_1 & \cdots & v_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & v_{n-1} & \cdots & v_{n-1}^{n-1} \end{pmatrix}$$

Matrix Displacement

- An $n \times n$ matrix $\boldsymbol{M}$ is a *structured matrix* when its displacement rank γ is much less than n
- More precisely, with proper choice of operator matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the Sylvester displacement, $\nabla_{\boldsymbol{A},\boldsymbol{B}}(\boldsymbol{M}) := \boldsymbol{AM} - \boldsymbol{MB}$, and Stein displacement, $\Delta_{\boldsymbol{A},\boldsymbol{B}}(\boldsymbol{M}) := \boldsymbol{M} - \boldsymbol{AMB}$, of $\boldsymbol{M}$ has a rank γ bounded by a value independent of the size of $\boldsymbol{M}$
- When determining these operator matrices, let $\boldsymbol{s}$ and $\boldsymbol{t}$ denote vectors defining Vandermonde and Cauchy matrices respectively
- Let $\boldsymbol{Z}_f$ represent the f-unit-circulant matrix

$$\boldsymbol{Z}_f = [e_2, e_3, \dots, e_n, fe_1] = \begin{bmatrix} 0 & 0 & \cdots & f \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix}$$

Where e_j represents the j^{th} standard basis column vector

Matrix Displacement Table

$\boldsymbol{A}$	$\boldsymbol{B}$	Structure	Rank
$\boldsymbol{Z}_1$	$\boldsymbol{Z}_0$	Circulant	≤ 2
$\boldsymbol{Z}_1$	$\boldsymbol{Z}_0$	Toeplitz	≤ 2
$\boldsymbol{Z}_0$	$\boldsymbol{Z}_1$	Henkel	≤ 2
diag$(\boldsymbol{t})$	$\boldsymbol{Z}_0$	Vandermonde	≤ 1
diag$(\boldsymbol{s})$	diag$(\boldsymbol{t})$	Cauchy	≤ 1

Working with LDR Matrices

- General procedure of handling LDR matrices can be broken down into three steps
  - Compression – means to obtain a low-rank displacement of the matrices
  - Computation with Displacements
  - Decompression – converting the results from displacement computations to the answer to the original computation problem
- If we assume one of the displacement operators have a specific property, we can decompress directly
- If $\boldsymbol{A}$ is an a-potent matrix (that is, $\boldsymbol{A}^q = a\boldsymbol{I}$ for some positive integer $q \leq n$), then

$$\boldsymbol{M} = \left[\sum_{k=0}^{q-1} \boldsymbol{A}^k \Delta_{\boldsymbol{A},\boldsymbol{B}}(\boldsymbol{M})\boldsymbol{B}^k\right](\boldsymbol{I} - a\boldsymbol{B}^q)^{-1}$$

LDR Neural Networks

- For this analysis, assume a feed-forward network with one dense hidden layer (n input neurons and kn hidden neurons)
- These are connected by a weight matrix $\boldsymbol{W} \in \boldsymbol{R}^{n \times kn}$ of displacement rank $r \ll n$ corresponding to displacement operators (using zero-padding with the nearest k allows for generalization to $n \times m$) $(\boldsymbol{A}, \boldsymbol{B})$
- Let the domain for the input vector $\boldsymbol{x}$ be $[0,1]^n$ and let the output layer have only one neuron
- Network can then be expressed as

$$y = G_{\boldsymbol{W},\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{kn} \alpha_j \sigma \left(\boldsymbol{w}_j^\intercal \boldsymbol{x} + \theta_j \right)$$

Where $\sigma(\cdot)$ is the activation function, $\boldsymbol{w}_j$ is the j^{th} column of $\boldsymbol{W}$, and $\alpha_j, \theta_j \in \mathbb{R}$ for $j = 1, \ldots, kn$

Representation Property

- Let $\boldsymbol{A}, \boldsymbol{B}$ be two $n \times n$ non-singular diagonalizable matrices (i.e. a matrix with an inverse, nonzero determinant, where an invertible matrix $\boldsymbol{P}$ exists such that $\boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}$ is diagonal) satisfying:
  - $\boldsymbol{A}^q = a\boldsymbol{I}$ for some positive integer $q \leq n$ and a scalar $a \neq 0$
  - $(\boldsymbol{I} - a\boldsymbol{B}^q)$ is nonsingular
  - Eigenvalues of $\boldsymbol{B}$ have distinguishable absolute values
- Let S be the set of matrices $\boldsymbol{M}$ such that the $\Delta_{\boldsymbol{A},\boldsymbol{B}}(\boldsymbol{M})$ has rank 1

$$S_{\boldsymbol{A},\boldsymbol{B}} = \left\{\boldsymbol{M} \in \mathbb{R}^{n\times n} \mid \exists \boldsymbol{g}, \boldsymbol{h} \in \mathbb{R}^n, \Delta_{\boldsymbol{A},\boldsymbol{B}}(\boldsymbol{M}) = \boldsymbol{g}\boldsymbol{h}^\intercal\right\}$$

- Then for any vector $\boldsymbol{v} \in \mathbb{R}^n$, there exists a matrix $\boldsymbol{M} \in S_{\boldsymbol{A},\boldsymbol{B}}$ and an index $v \in \{1, \ldots, n\}$ such that the i^{th} column of $\boldsymbol{M}$ equals vector $\boldsymbol{v}$
- This family of matrices S then is said to have *representation property*

Discriminatory Functions

- A function $\sigma(u)\colon \mathbb{R} \to \mathbb{R}$ is called discriminatory if the zero measure is the only measure μ that satisfies the following property:

$$\int_{\boldsymbol{I}_n} \sigma\left(\boldsymbol{w}^\intercal \boldsymbol{x} + \theta\right) d\mu(\boldsymbol{x}) = 0, \forall \boldsymbol{w} \in \mathbb{R}^n, \theta \in \mathbb{R}$$

  That is, for a measure μ, if the above integral is zero, then $\mu = 0$
- We say that σ is sigmoidal if

$$\sigma(t) \to \begin{cases} 1 & \text{as } t \to +\infty \\ 0 & \text{as } t \to -\infty \end{cases}$$

- Any bounded, measurable sigmoidal function is discriminatory

Universal Approximation Theorem for LDR Neural Networks

For any continuous function $f(\boldsymbol{x})$ defined on $\boldsymbol{I}_n$, $\epsilon > 0$, and any $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times n}$ satisfying the requirements established by the representation property of S, then there exists a function $G(\boldsymbol{x})$ in the form of $y = G_{\boldsymbol{W},\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{kn} \alpha_j \sigma\left(\boldsymbol{w}_j^\intercal \boldsymbol{x} + \theta_j\right)$ so that its weight matrix consists of k submatrices with a displacement rank of 1 and

$$\max_{x \in \boldsymbol{I}_n} |G(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon$$

Error Bounds on LDR Neural Networks

- For LDR matrices defined by $\mathcal{O}(n)$ parameters (n number of rows and the same order number of columns), the corresponding structured neural network is achieving integrated squared error of order $\mathcal{O}\left(\frac{1}{n}\right)$ with n parameters
- This is asymptotically equivalent to results previously determined for general neural networks, implying that there is essentially no loss when restricting to LDR matrices
- Functions we want to approximate are those defined on an n-ball $B_r = \{\boldsymbol{x} \in \mathbb{R}^n : |\boldsymbol{x}| \leq r\}$ such that $\int_{B_r} |\boldsymbol{x}||f(\boldsymbol{x})|\mu(d\boldsymbol{x}) \leq C$, where μ is an arbitrary measure normalized such that $\mu(B_r) = 1$, call this set of functions Γ_{C,B_r}
- Consider the following set of bounded multiples of a sigmoidal function composed of linear functions:

$$G_\sigma = \{\alpha\sigma(\boldsymbol{y}^\intercal \boldsymbol{x} + \theta) : |\alpha| \leq 2C, \boldsymbol{y} \in \mathbb{R}^n, \theta \in \mathbb{R}\}$$

Error Bound of One Hidden Layer Neural Network

- For every disk $B_r \subset \mathbb{R}^n$, every function Γ_{C,B_r}, every sigmoidal function σ, every probability measure, every normalized measure μ, and every $k \geq 1$, there exists linear combination of sigmoidal function $f_k(\boldsymbol{x})$ of the form:

$$f_k(\boldsymbol{x}) = \sum_{j=1}^{k} \alpha_j \sigma\Big(\boldsymbol{y}_j^\intercal + \theta_j\Big)$$

  such that

$$\int_{B_r} (f(\boldsymbol{x}) - f_k(\boldsymbol{x}))^2 \mu(d\boldsymbol{x}) \leq \frac{4r^2C}{k}$$

- $\boldsymbol{y}_j \in \mathbb{R}^n$ and $\theta_j \in \mathbb{R}$ for every $j = 1, 2, \ldots, N$
- Coefficients of the linear combination may be restricted to satisfy $\sum_{j=1}^{k} |c_j| \leq 2rC$

Extending to LDR Neural Networks

- Fix operator $(\boldsymbol{A}, \boldsymbol{B})$ and define:

$$S_\sigma = \{ \sum_{j=1}^{kn} \alpha_j \sigma \left(\boldsymbol{y}_j^\intercal \boldsymbol{x} + \theta_j \right) : |\alpha_j| \leq 2C, \boldsymbol{y}_j \in \mathbb{R}^n, \theta_j \in \mathbb{R}, j = 1, \dots, N,$$

$$\text{and } \left[\boldsymbol{y}_{(i-1)n+1} \mid \boldsymbol{y}_{(i-1)n+2} \mid \cdots \mid \boldsymbol{y}_{in} \right] \text{ is an LDR matrix,}$$

$$\forall i = 1, \dots, k \}$$

- $B_r \subset \mathbb{R}^n$, every function in Γ_{C,B_r}, every sigmoidal function σ, every normalized measure μ, and every $k \geq 1$, there exists a neural network defined by a weight weight matrix consisting of k LDR submatrices such that:

$$\int_{B_r} \left(f(\boldsymbol{x}) - f_{kn}(\boldsymbol{x}) \right)^2 \mu(d\boldsymbol{x}) \leq \frac{4r^2 C}{k}$$

- Coefficients of the linear combination may be restricted to satisfy $\sum_{j=1}^{k} |c_k| \leq 2rC$

Training LDR Neural Networks

- Need to reformulate gradient computation for LDR neural networks
- Computation for propagating through a fully-connected layer can be written as

$$\boldsymbol{y} = \sigma\left(\boldsymbol{W}^\intercal \boldsymbol{x} + \boldsymbol{\theta}\right)$$

  Where $\sigma(\cdot)$ is the activation function, $\boldsymbol{W} \in \mathbb{R}^{n \times kn}$ is the weight matrix, $\boldsymbol{x} \in \mathbb{R}^n$ is the input vector, and $\boldsymbol{\theta} \in \mathbb{R}^{kn}$ is the bias vector
- If $\boldsymbol{W}_i$ is an LDR matrix with operators $(\boldsymbol{A_i}, \boldsymbol{B}_i)$ satisfying previously stated conditions, then it is determined by two matrices $\boldsymbol{G}_i, \boldsymbol{H}_i \in \mathbb{R}^{n \times r}$ as:

$$\boldsymbol{W}_i = \left[\sum_{k=0}^{q-1} \boldsymbol{A}_i^k \boldsymbol{G}_i \boldsymbol{H}_i^\intercal \boldsymbol{B}_i^k\right] \left(\boldsymbol{I} - a\boldsymbol{B}_i^q\right)^{-1}$$

Fitting Backpropagation

- In order to fit backprop, we need to compute derivatives $\frac{\partial J}{\partial \boldsymbol{G}_i}$, $\frac{\partial J}{\partial \boldsymbol{H}_i}$, and $\frac{\partial J}{\partial \boldsymbol{x}}$ for any objective function $J = J(\boldsymbol{W}_1, \dots, \boldsymbol{W}_k)$
- In general, given $\boldsymbol{a} = \boldsymbol{W}^\intercal \boldsymbol{x} + \boldsymbol{\theta}$, we have (where $\mathbf{1}$ is a column vector of ones):

$$\frac{\partial J}{\partial \boldsymbol{W}} = \boldsymbol{x}\left(\frac{\partial J}{\partial \boldsymbol{a}}\right)^\intercal \qquad \frac{\partial J}{\partial \boldsymbol{x}} = \boldsymbol{W}\frac{\partial J}{\partial \boldsymbol{a}} \qquad \frac{\partial J}{\partial \boldsymbol{\theta}} = \frac{\partial J}{\partial \boldsymbol{a}}\mathbf{1}$$

- Let $\hat{\boldsymbol{G}}_{ik} = \boldsymbol{A}_i^k \boldsymbol{G}_i$, $\hat{\boldsymbol{H}}_{ik} = \boldsymbol{H}_i^\intercal \boldsymbol{B}_i^k (\boldsymbol{I} - a\boldsymbol{B}_i^q)^{-1}$, and $\boldsymbol{W}_{ik} = \hat{\boldsymbol{G}}_{ik}\hat{\boldsymbol{H}}_{ik}$, then:

$$\frac{\partial J}{\partial \boldsymbol{W}_{ik}} = \frac{\partial J}{\partial \boldsymbol{W}_i}$$

Fitting Backpropagation (cont.)

- If we let $\boldsymbol{a} = \boldsymbol{W}_{ik}$, $\boldsymbol{W} = \hat{\boldsymbol{G}}_{ik}^{\intercal}$, and $\boldsymbol{x} = \hat{\boldsymbol{H}}_{ik}$, then:

$$\frac{\partial J}{\partial \hat{\boldsymbol{G}}_{ik}} = \left[\frac{\partial J}{\hat{\boldsymbol{G}}_{ik}^{\intercal}}\right]^{\intercal} = \left[\hat{\boldsymbol{H}}_{ik} \frac{\partial J}{\partial \boldsymbol{W}_{ik}}\right]^{\intercal} = \left(\frac{\partial J}{\partial \boldsymbol{W}_{ik}}\right)^{\intercal} \hat{\boldsymbol{H}}_{ik}^{\intercal}$$

$$\frac{\partial J}{\partial \hat{\boldsymbol{H}}_{ik}} = \hat{\boldsymbol{G}}_{ik} \frac{\partial J}{\partial \boldsymbol{W}_{ik}}$$

- If we let $\boldsymbol{a} = \hat{\boldsymbol{G}}_{ik}$, $\boldsymbol{W} = \left(\boldsymbol{A}_i^k\right)^{\intercal}$, and $\boldsymbol{x} = \boldsymbol{G}_i$, then:

$$\frac{\partial J}{\partial \boldsymbol{G}_i} = \sum_{k=0}^{q-1} \left(\boldsymbol{A}_i^k\right)^{\intercal} \left(\frac{\partial J}{\partial \hat{\boldsymbol{G}}_{ik}}\right) = \sum_{k=0}^{q-1} \left(\boldsymbol{A}_i^k\right)^{\intercal} \left(\frac{\partial J}{\partial \boldsymbol{W}_{ik}}\right)^{\intercal} \hat{\boldsymbol{H}}_{ik}^{\intercal}$$

Fitting Backpropagation (cont.)

- If we let $\boldsymbol{a} = \hat{\boldsymbol{H}}_{ik}$, $\boldsymbol{W} = \boldsymbol{H}_i^\intercal$, and $\boldsymbol{x} = \boldsymbol{B}_i^k \left(\boldsymbol{I} - a\boldsymbol{B}_i^q\right)^{-1}$, then:

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{H}_i} &= \sum_{k=0}^{q-1} \boldsymbol{B}_i^k \left(\boldsymbol{I} - a\boldsymbol{B}_i^q\right)^{-1} \left(\frac{\partial J}{\partial \hat{\boldsymbol{H}}_{ik}}\right)^\intercal \\ &= \sum_{k=0}^{q-1} \boldsymbol{B}_i^k \left(\boldsymbol{I} - a\boldsymbol{B}_i^q\right)^{-1} \left(\frac{\partial J}{\partial \boldsymbol{W}_{ik}}\right)^\intercal \hat{\boldsymbol{G}}_{ik}\end{aligned}$$

- Using these derivations, $\frac{\partial J}{\partial \boldsymbol{G}_i}$ and $\frac{\partial J}{\partial \boldsymbol{H}_i}$ can be computed given $\frac{\partial J}{\partial \boldsymbol{W}_{ik}}$, which itself is equal to $\frac{\partial J}{\partial \boldsymbol{W}_i}$
- Given the nature of backprop, $\frac{\partial J}{\partial \boldsymbol{W}_i}$ can be calculated from the previous layer and $\frac{\partial J}{\partial \boldsymbol{x}}$ will be propagated to the next layer if required

Overview of Training LDR Neural Networks

- For practical purposes, matrices $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ can be chosen such that fast multiplication methods exist (e.g. diagonal matrices, permutation matrices, banded matrices)
- The space complexity of $\boldsymbol{W}_i$ is then $\mathcal{O}(2n + 2nr)$ with displacement rank r, rather than $\mathcal{O}\left(n^2\right)$
- $2n$ comes from $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$, and $2nr$ from $\boldsymbol{G}_i$ and $\boldsymbol{H}_i$
- Time complexity of $\boldsymbol{W}_i^\intercal \boldsymbol{x}$ is $\mathcal{O}(q(3n + 2nr)))$ versus $\mathcal{O}\left(n^2\right)$ with a dense matrix
- When $\boldsymbol{W_i}$ is a Toeplitz matrix, the space complexity is $\mathcal{O}(2n)$ as its defined by $2n$ parameters and since matrix-vector multiplication can be accelerated with FFT, time complexity becomes $\mathcal{O} n \log n$

Tropical Geometry of Deep Neural Networks; Zhang, Naitzat, Lim

arXiv:1805.07091

Tropical Algebra

- Replacing $+, \cdot$ with $\max, +$ (denoted $\oplus, \odot$ in tropical algebra) results in a system of algebra very similar to standard algebra
- Operates on the tropical semiring $\mathbb{T} := \{\mathbb{R} \cup \{-\infty\}, \oplus, \odot\}$
- Extend this to tropical power, such that $x^{\odot a} \equiv a \cdot x$
- Tropical monomials can therefore be constructed where $\odot$ (standard addition) replaces $\cdot$ (standard multiplication)
- Tropical polynomials are then the maximum of a set of tropical monomials ($\oplus$ replaces $+$)

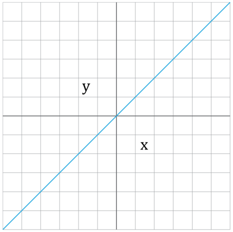
Tropical Rational Functions

- In standard algebra, we define a rational function as any function defined by an algebraic fraction (that is, a fraction where the numberator and denominator are polynomials)
- Logically extends to tropical polynomials by taking the tropical quotient, $\oslash$ (equivalent to standard difference), of two tropical polynomials
- Since a tropical polynomial f can be written $f \oslash 0$, the semiring of tropical polynomials is therefore a subset of the semifield of tropical rational functions
- Each tropical polynomial defines a convex function (as $\oplus$ and $\odot$ preserve convexity)
- Each tropical rational function is therefore a DC (difference-convex) function
- $F\colon \mathbb{R}^d \to \mathbb{R}^p, x = (x_1, \dots, x_d) \mapsto (f_1(x), \dots, f_p(x))$
  - Is a tropical polynomial map if $f_1, \dots, f_p$ are tropical polynomials
  - Is a tropical rational map if $f_1, \dots, f_p$ are tropical rational functions

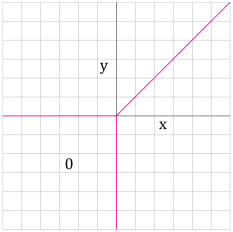
Tropical Hypersurfaces

- The tropical hypersurface $\mathcal{T}(f)$ of a tropical polynomial f is the set of points $\boldsymbol{x}$ at which the value of f at $\boldsymbol{x}$ is attained by at least two monomials in f
- The cells between the tropical hypersurface each represent a monomial of f, and the hypersurface itself represents the points at which the values of monomials of adjacent cells are equal

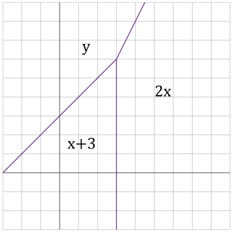
$f(x, y) = x \oplus y$



$f(x, y) = x \oplus y \oplus 0$

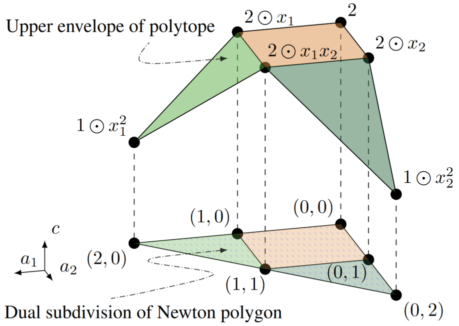


$f(x, y) = x^{\odot 2} \oplus 3 \odot x \oplus y$



Newton Polygons of Tropical Polynomials

- The Newton polygon $\Delta(f)$ for a tropical polynomial $f(\boldsymbol{x})$ is defined by the convex hull of the tropical exponent tuples of the monomials of $f(\boldsymbol{x})$
- The dual-subdivision $\delta(f)$ is $\Delta(f)$ "lifted" by the constant portion of each monomial (denoted $\mathcal{P}(f)$), and the upper faces thereof (denoted $\mathrm{UF}(\mathcal{P}(f))$) are projected down back to the original space



$$(1 \odot x_1^2) \oplus (1 \odot x_2^2) \oplus (2 \odot x_1 \odot x_2) \oplus (2 \odot x_1) \oplus (2 \odot x_2) \oplus 2$$

Linear Regions of Tropical Maps

- Linear region of $F \in \mathrm{Rat}(d, m)$ is a maximal connected subset of the domain on which F is linear, with the number of linear regions denoted $\mathcal{N}(F)$
- A tropical polynomial map $F \in \mathrm{Pol}(d, m)$ has convex linear regions but a tropical rational map $F \in \mathrm{Rat}(d, n)$ generally has non-convex linear regions

Transformations of Tropical Polynomials

- Tropical power acts in such a way that it scales a polytope $\mathcal{P}(f)$, changing its volume, while maintaining its shape

$$\mathcal{P}\left(f^{\odot a}\right) = a\mathcal{P}(f)$$

for $a \in \mathbb{N}$ and $f \in \mathrm{Pol}(d, 1)$

- Tropical product between two polytopes $\mathcal{P}(f)$ and $\mathcal{P}(g)$ is summing their vertices (denoted $\mathcal{V}(\cdot)$) via the Minkowski sum, and determining the convex hull thereof

$$\mathcal{P}(f \odot g) = \mathrm{Conv}\left(\mathcal{V}(\mathcal{P}(f)) + \mathcal{V}(\mathcal{P}(g))\right)$$

for $f, g \in \mathrm{Pol}(d, 1)$

- Tropical sum between two polytopes can then be thought of as the convex hull of the union of these vertices

$$\mathcal{P}(f \oplus g) = \mathrm{Conv}\left(\mathcal{V}(\mathcal{P}(f)) \cup \mathcal{V}(\mathcal{P}(g))\right)$$

for $f, g \in \mathrm{Pol}(d, 1)$

Neural Network Assumptions

- Set three assumptions of feedforward neural networks
  - Weight matrices $A^{(1)}, \ldots, A^{(L)}$ are integer-valued
  - Bias vectors $b^{(1)}, \ldots, b^{(L)}$ are real-valued
  - Activation functions $\sigma^{(1)}, \ldots, \sigma^{(L)}$ are of the form $\sigma^{(\ell)}(x) := \max\left\{x, t^{(\ell)}\right\}$ where $t^{(\ell)} \in \mathbb{T}^{n_\ell}$ is a threshold vector
- The activation function assumed is equivalent to ReLU when $t^{(\ell)} = 0$ and the identity function when $t^{(\ell)} = -\infty$
- There is no loss of generality in restricting the weights $A^{(1)}, \ldots, A^{(L)}$ to integers since:
  - Real weights can be approximated arbitrarily closely by rational weights
  - Denominators can be "cleared" in rational weights by multiplying them by the LCM of their denominators to obtain integer weights
  - Scaling of all weights and biases by the same positive constant does not change the workings of a neural network

Tropical Algebra of Neural Networks

- Deep feedforward neural networks are generally non-convex, whereas tropical polynomials are always convex
- Since most non-convex functions are the difference of two convex functions, we can intuit that a feedforward network is the difference between two tropical polynomials (i.e. a tropical rational function)

Layer Decomposition

- Consider the output from the first layer of the neural network:

$$\nu(x) = \max\{Ax + b, t\}$$

- Decompose A as a difference of two non-negative integer matrices, $A = A_+ - A_-$ such that:

$$a_{ij}^+ = \max\{a_{ij}, 0\} \qquad a_{ij}^- = \max\{-a_{ij}, 0\}$$

- We can then see that each coordinate is a difference of two polymonials since:

$$\max\{Ax + b, t\} = \max\{A_+x + b, A_-x + t\} - A_-x$$

  And for deep networks, this decomposition can be applied recursively

Recursive Decomposition

- Let A, b, t parameterize the $(\ell+1)^{\text{th}}$ layer
- Let the ℓ^{th} layer be given by the following tropical rational functions: $\nu^{(\ell)} = F^{(\ell)} \oslash G^{(\ell)}$ (each coordinate of $F^{(\ell)}$ and $G^{(\ell)}$ is a tropical polynomial in x
- Then the preactivation and output of the $(\ell+1)^{\text{th}}$ layer are given by:

$$\begin{aligned} \rho^{(\ell+1)} \circ \nu^{(\ell)}(x) &= H^{(\ell+1)}(x) - G^{(\ell+1)}(x) \\ \nu^{(\ell+1)}(x) = \sigma \circ \rho^{(\ell+1)} \circ \nu^{(\ell)}(x) &= F^{(\ell+1)}(x) - G^{(\ell+1)}(x) \end{aligned}$$

- Where:

$$\begin{aligned} F^{(\ell+1)}(x) &= \max\left\{H^{(\ell+1)}(x), G^{(\ell+1)}(x) + t\right\} \\ G^{(\ell+1)}(x) &= A_+ G^{(\ell)}(x) + A_- F^{(\ell)}(x) \\ H^{(\ell+1)}(x) &= A_+ F^{(\ell)}(x) + A_- G^{(\ell)}(x) + b \end{aligned}$$

Tropical Characterization and Equivalence of Neural Networks

- A feedforward neural network under the aforementioned assumptions is a function $\nu\colon \mathbb{R}^d \to \mathbb{R}^p$ whose coordinates are tropical rational functions, i.e. $\nu(x) = F(x) \oslash G(x)$ where F and G are tropical polynomial maps, thus ν is a tropical rational map
- Let $t^{(1)}, \dots, t^{(L-1)} = 0$ and $t^{(L)} = -\infty$, and let $\nu\colon \mathbb{R}^d \to \mathbb{R}$ be a ReLU feedforward network with integer weights and linear output, then ν is a tropical rational function
- $\nu\colon \mathbb{R}^d \to \mathbb{R}$ is then a tropical rational function iff ν is a feedforward neural network satisfying the aforementioned assumptions
- A tropical rational function $f \oslash g$ can be represented as an L-layer neural network with

$$L \leq \max\{\lceil \log_2 r_f \rceil, \lceil \log_2 r_g \rceil\} + 2$$

layers, where r_f and r_g are the number of monomials in f and g

Decision Boundaries of a Neural Network

- Focus on the binary classification problem, with a neural network $\nu\colon \mathbb{R}^d \to \mathbb{R}^p$ and a score function $s\colon \mathbb{R} \to \mathbb{R}$, where if $s(\nu(x))$ exceeds a decision threshold c, it belongs to one class, otherwise it belongs to the other class
- Decision boundary $\mathcal{B} := \left\{x \in \mathbb{R}^d : \nu(x) = s^{-1}(c)\right\}$ partitions the input space into two disjoint sets, where connected regions above c are *positive regions* and below c are *negative regions*
- Let $\nu\colon \mathbb{R}^d \to \mathbb{R}$ be an L-layer network satisfying our assumptions with $t^{(L)} = -\infty$ and let $s\colon \mathbb{R} \to \mathbb{R}$ be injective with c in its range, if $\nu = f \oslash g$ with tropical polynomials f and g then:
  - $\mathcal{B}$ divides $\mathbb{R}^d$ into at most $\mathcal{N}(f)$ positive and $\mathcal{N}(g)$ negative regions
  - $B \subseteq \mathcal{T}\left(\left(s^{-1}(c) \odot g\right) \oplus f\right)$

  Where $\mathcal{N}(\cdot)$ represents the number of linear regions and $\mathcal{T}(\cdot)$ the tropical hypersurface
- $s^{-1}(c) \odot g \oplus f$ is not necessarily linear on every positive or negative region so $\mathcal{T}$ thereof may divide a positive or negative region into multiple linear regions

Zonotopes as Building Blocks of Neural Networks

- The number of regions of $\mathcal{T}(f)$ divides space into is equal to the number of vertices in $\delta(f)$
- Let $f_i^{(\ell)}$, $g_i^{(\ell)}$, $h_i^{(\ell)}$ be the tropical polynomials produced by the i^{th} node in the ℓ^{th} layer, then $\mathcal{P}\left(f_i^{(\ell)}\right), \mathcal{P}\left(g_i^{(\ell)}\right), \mathcal{P}\left(h_i^{(\ell)}\right) \subseteq \mathbb{R}^{d+1}$
- $\mathcal{P}\left(g_i^{(1)}\right)$ and $\mathcal{P}\left(h_i^{(1)}\right)$ are points (no $\max$ operation is applied)
- $\mathcal{P}\left(f_i^{(1)}\right)$ is a line segment (single $\max$ operation between points is applied)
- $\mathcal{P}\left(g_i^{(2)}\right)$ and $\mathcal{P}\left(h_i^{(2)}\right)$ are zonotopes (weighted Minkowski sum between line segments)
- For $\ell \geq 1$, $\mathcal{P}\left(f_i^{(\ell)}\right) = \operatorname{Conv}\left[\mathcal{P}\left(g_i^{(\ell)} \odot t_i^{(\ell)}\right) \cup \mathcal{P}\left(h_i^{(\ell)}\right)\right]$
- For $\ell \geq 1$, $\mathcal{P}\left(g_i^{(\ell+1)}\right)$ and $\mathcal{P}\left(h_i^{(\ell+1)}\right)$ are weighted Minkowski sums

Geometric Complexity of Deep neural Networks

- Use the number of linear regions $\mathcal{N}(\cdot)$ as our measure of complexity
- Let $\nu\colon \mathbb{R}^d \to \mathbb{R}$ be an L-layered real-valued feedforward neural network satisfying our assumptions, $t^{(L)} = -\infty$, and $n_\ell \geq d$ for all $\ell = 1, \ldots, L-1$
- Then $\nu = \nu^{(L)}$ has at most

$$\prod_{\ell=1}^{L-1} \sum_{i=0}^{d} \binom{n_\ell}{i}$$

  linear regions, where d is the dimensionality of the input space and n_ℓ is the width of the ℓ^{th} layer
- if $d \leq n_1, \ldots, n_{L-1} \leq n$, then $\mathcal{N}(\nu)$ is bounded by $\mathcal{O}\big(n^{d(L-1)}\big)$
- We can therefore say that the number of linear regions of the neural network grows polynomially with width n and exponentially with depth L

Appendix A: Detailed Learning Procedure for Growing Neural Gas;

Growing Neural Gas – Learning Procedure

1. Generate two nodes a and b at random values in $\mathbb{R}^S$
2. Draw a random sample ξ from the probability distribution being learned as $\xi \sim P$
3. Find two nodes v_1 and v_2 such that v_1 is closest to (and v_2 is second closest to) ξ as determined by finding the $v \in V$ which produces the first and second smallest $\|w_v - \xi\|_2^2$
4. For each edge e connected to v_1, increment its age α_e by 1
5. Increment the error for the BMU:

$$\epsilon_{v_1} \longleftarrow \epsilon_{v_1} + \|w_{v_1} - \xi\|_2^2$$

6. Move v_1 and its neighbors towards ξ by η_b and η_n respectively:

$$w_{v_1} \longleftarrow w_{v_1} + \epsilon_b \left(\xi - w_{v_1}\right) \qquad \text{for best matching unit}$$
$$w_{v_n} \longleftarrow w_{v_n} + \epsilon_n \left(\xi - w_{v_n}\right) \qquad \text{for neighbors } n \text{ of BMU}$$

Growing Neural Gas – Learning Procedure (cont.)

7. If an edge connects v_1 and v_2, set its age to zero, $e\left(v_1, v_2\right) \longleftarrow 0$. Otherwise create such an edge with an age of zero.
8. For each edge $e \in E$, if $\alpha_e > \alpha_{\max}$ remove it. If after removing said edges any nodes have a degree of zero, remove them.
9. If the number of samples drawn is an integer multiple of λ, insert a new node by finding the node with the maximum error and its neighbor, generate a new node halfway between the neighbor and the unit with maximum error, insert edges between the new node and the others (removing the original) and reduce the accumulated errors of the two existing nodes
10. Decay all error values by multiplying them by the global decay rate of error:

$$\forall n \in N, \epsilon_n \longleftarrow \eta_d \epsilon_n$$

11. If a stopping criterion such as model size or a performance metric is not yet met, return to step 2 and repeat.

Appendix B: Detailed Information on LDR Neural Networks;

Appendix C: Detailed Information on Tropical Algebra;

Tropical Algebra

- Define the *tropical semiring* as $\mathbb{T} \coloneqq (\mathbb{R} \cup \{-\infty\}, \oplus, \odot)$
- Define tropical sum, product, and quotient for two numbers $x, y \in \mathbb{R}$:
  - $x \oplus y \coloneqq \max(x, y) \quad x \odot y \coloneqq x + y \quad x \oslash y \coloneqq x - y$
- Tropical additive identity of 0 and multiplicative identity of $-\infty$
  - $-\infty \oplus x = 0 \odot x = x \quad -\infty \odot x = -\infty$
  - Lacks the additive inverse required for $(\mathbb{R} \cup \{-\infty\}, \oplus, \odot)$ to be a ring
- Operations follow the usual laws of arithmetic: associativity, commutativity, distributivity
- Define tropic power in a way analogous to power in the traditional sense

Tropical Power

- We can define *tropical power*, analogous to power in a traditional sense but where multiplication is substituted with tropical multiplication
- Given $a \in \mathbb{N}_0$ and $x \in \mathbb{R}$ we define this as:
  - $x^{\odot a} := x \odot \cdots \odot x = a \cdot x$
- Extends to $\mathbb{R} \cup \{-\infty\}$ for any $a \in \mathbb{N}$ as:

$$-\infty^{\odot a} := \begin{cases} -\infty & a > 0 \\ 0 & a = 0x \end{cases}$$

- $\mathbb{T}$ is a semifield since every $x \in \mathbb{R}$ has a tropical multiplicative inverse (equivalent to the standard additive inverse), $x^{\odot(-1)} := -x$
- $x \in \mathbb{R}$ can be raised to a negative power $a \in \mathbb{Z}$ by raising its tropical multiplicative inverse $-x$ to the positive power $-a$, i.e. $x^{\odot a} = (-x)^{\odot(-a)}$

Tropical Monomials and Polynomials

- A tropical monomial in d variables $x_1, \ldots, x_d$ is an expression of the form:

$$c \odot x_1^{\odot a_1} \odot x_2^{\odot a_2} \odot \cdots \odot x_d^{\odot a_d}$$

  where $c \in \mathbb{R} \cup \{-\infty\}$ and $a_1, \ldots, a_d \in \mathbb{N}$
- Use multiindex notation for shorthand as cx^{α} where $\alpha = (a_1, \ldots, a_d) \in \mathbb{N}^d$ and $x = (x_1, \ldots, x_d)$
- A tropical polynomial $f(x) = f(x_1, \ldots, x_d)$ is a finite tropical sum of tropical monomials
  - $f(x) = c_1 x^{\alpha_1} \oplus \cdots \oplus c_r x^{\alpha_r}$

  where $\alpha_i = (a_{i1}, \ldots, a_{id}) \in \mathbb{N}^d$ and $c_i \in \mathbb{R} \cup \{-\infty\}, i = 1, \ldots, r$

Tropical Rational Functions

- Define a tropical rational function as the standard difference (or tropical quotient) of two tropical polynomials $f(x)$ and $g(x)$:
  - $f(x) - g(x) = f(x) \oslash g(x)$

  denoted as $f \oslash g$ where f and g are tropical polynomial functions
- Set of tropical polynomials $\mathbb{T}[x_1, \ldots, x_d]$ forms a semiring under $\oplus$ and $\odot$, and the set of tropical rational functions $\mathbb{T}(x_1, \ldots, x_d)$ forms a semifield
- A tropical polynomial $f = f \oslash 0$ is a special case of tropical rational functions, therefore $\mathbb{T}[x_1, \ldots, x_d] \subseteq \mathbb{T}(x_1, \ldots, x_d)$
- A d-variate tropical polynomial defines a function $f \colon \mathbb{R}^d \to \mathbb{R}$ that is convex as taking max and sum of convex functions preserve convexity
- A tropical rational function $f \oslash g \colon \mathbb{R}^d \to \mathbb{R}$ is a DC (difference-convex) function

Tropical Maps

- $F\colon \mathbb{R}^d \to \mathbb{R}^p, x = (x_1, \ldots, x_d) \mapsto (f_1(x), \ldots, f_p(x))$
  - Is a tropical polynomial map if each $f_i\colon \mathbb{R}^d \to \mathbb{R}$ is a tropical polynomial, $i = 1, \ldots, p$
  - Is a tropical rational map if $f_1, \ldots, f_p$ are tropical rational functions
- $\mathrm{Pol}(d, p)$ is the set of tropical polynomial maps, so $\mathrm{Pol}(d, 1) = \mathbb{T}\,[x_1, \ldots, x_d]$
- $\mathrm{Rat}(d, p)$ is the set of tropical rational maps, so $\mathrm{Rat}(d, 1) = \mathbb{T}\,(x_1, \ldots, x_d)$

Tropical Hypersurfaces

- The tropical hypersurface of a tropical polynomial $f(x)$ is
  - $\mathcal{T}(f) := \left\{x \in \mathbb{R}^d : c_i x^{\alpha_i} = c_j x^{\alpha_j} = f(x) \text{ for some } \alpha_i \neq \alpha_j\right\}$
  - Otherwise stated, the set of points x at which the value $f(x)$ is attained by two or more monomials in f
- A tropical hypersurface divides the domain of f into convex cells on each of which f is linear
- Cells are convex polyhedra, defined by linear inequalities with integer coefficients: $\left\{x \in \mathbb{R}^d : Ax \leq b\right\}$ for $A \in \mathbb{Z}^{m \times d}$ and $b \in \mathbb{R}^m$
- Cell where a tropical monomial $c_j x^{\alpha_j}$ attains its maximum is $\left\{x \in \mathbb{R}^d : c_j + a_j^\intercal x \geq c_i + a_i^\intercal x \textit{ for all } i \neq j\right\}$

Minkowski Sum

- The Minkowski sum of two sets P_1 and P_2 in $\mathbb{R}^d$ is the set:

$$P_1 + P_2 = \left\{ x_1 + x_2 \in \mathbb{R}^d : x_1 \in P_1, x_2 \in P_2 \right\}$$

- Intuitively described as summing every term in P_1 with every term in P_2
- Weighted Minkowski sum with scalars $\lambda_1, \lambda_2 \geq 0$ is:

$$\lambda_1 P_1 + \lambda_2 P_2 = \left\{ \lambda_1 x_1 + \lambda_2 x_2 \in \mathbb{R}^d : x_1 \in P_1, x_2 \in P_2 \right\}$$

- Weighted Minkowski sum is commutative, associative, and generalizes to more than two sets
- Minkowski sum of line segments is called a *zonotope*

Elementwise Recursive Decomposition

Standard Notation:

$$\begin{aligned} F^{(\ell+1)}(x) &= \max\left\{H^{(\ell+1)}(x), G^{(\ell+1)}(x) + t\right\} \\ G^{(\ell+1)}(x) &= A_+ G^{(\ell)}(x) + A_- F^{(\ell)}(x) \\ H^{(\ell+1)}(x) &= A_+ F^{(\ell)}(x) + A_- G^{(\ell)}(x) + b \end{aligned}$$

Elementwise Tropical Notation:

$$\begin{aligned} f_i^{(\ell+1)} &= h_i^{(\ell+1)} \oplus \left(g_i^{(\ell+1)} \odot t_i\right) \\ g_i^{(\ell+1)} &= \left[\bigodot_{j=1}^{n} \left(f_j^{(\ell)}\right)^{\odot a_{ij}^-}\right] \odot \left[\bigodot_{j=1}^{n} \left(g_j^{(\ell)}\right)^{\odot a_{ij}^+}\right] \\ h_i^{(\ell+1)} &= \left[\bigodot_{j=1}^{n} \left(f_j^{(\ell)}\right)^{\odot a_{ij}^+}\right] \odot \left[\bigodot_{j=1}^{n} \left(g_j^{(\ell)}\right)^{\odot a_{ij}^-}\right] \odot b_i \end{aligned}$$