DNN: Final Presentation

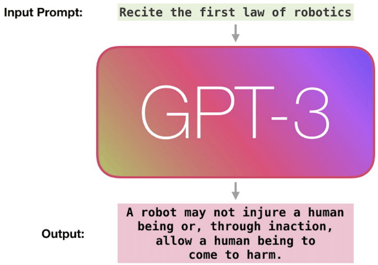
Ryan Jeong, Will Barton, Maricela Ramirez

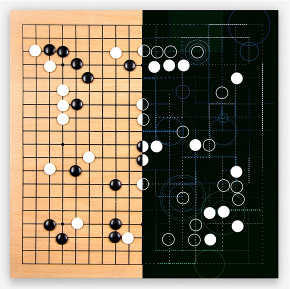
Summer@ICERM

June-July 2020

Outline

Motivation





Source: Jay Alammar, How GPT3 Works

Source: European Go Federation

DL Fundamentals

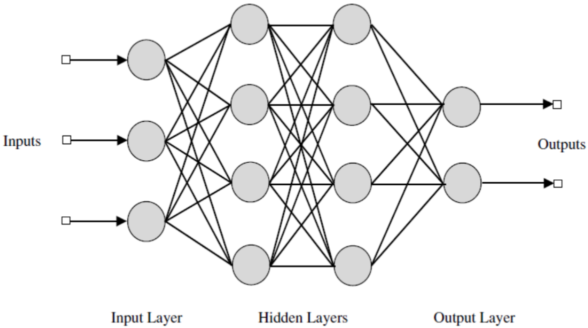


Figure: Source: Virtual Labs, Multilayer Feedforward networks

- General idea of a Feedforward network
  - $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$
  - $a^{(l)} = ReLU(z^{(l)})$

DL Fundamentals

- Activation function
- Cost function
- Minimize cost function through backpropogation
    - Gradient Descent

DL Fundamentals

- Activation function
- Cost function
- Minimize cost function through backpropogation
  - Gradient Descent
  - $\frac{\partial C_0}{\partial w^{(l)}} = \frac{\partial C_0}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}}$
  - $\frac{\partial C_0}{\partial b^{(l)}} = \frac{\partial C_0}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial b^{(l)}}$
  - Stochastic Gradient Descent

Convolutional Networks

- Motivation: computational issues with feedforward networks

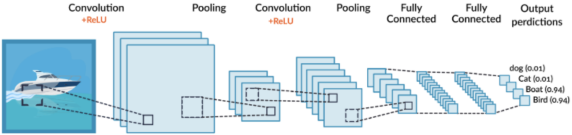


Figure: Source: missinglink.ai

- For image recognition problems, on which CNNs are largely applied:

Convolutional Networks

- Motivation: computational issues with feedforward networks

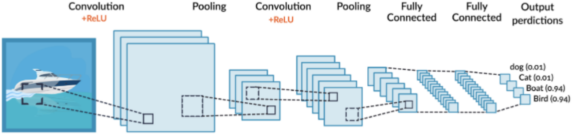


Figure: Source: missinglink.ai

- For image recognition problems, on which CNNs are largely applied:
  - Convolution layer: slide small kernel of fixed weights along the image, performing the same computation every time; activation performed on output of a convolution layer

Convolutional Networks

- Motivation: computational issues with feedforward networks

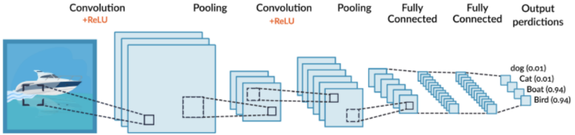


Figure: Source: missinglink.ai

- For image recognition problems, on which CNNs are largely applied:
  - Convolution layer: slide small kernel of fixed weights along the image, performing the same computation every time; activation performed on output of a convolution layer
  - Pooling layer: divides result of convolution into regions and computes a function on each one (usually just max); intuitively summarizes information and compresses dimension

Convolutional Networks

- Motivation: computational issues with feedforward networks

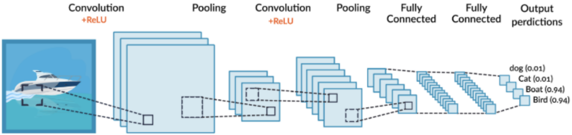


Figure: Source: missinglink.ai

- For image recognition problems, on which CNNs are largely applied:
  - Convolution layer: slide small kernel of fixed weights along the image, performing the same computation every time; activation performed on output of a convolution layer
  - Pooling layer: divides result of convolution into regions and computes a function on each one (usually just max); intuitively summarizes information and compresses dimension
  - Finishes with fully connected layers

SVD classification model

- Initial model from linear algebraic techniques as a baseline
- MNIST dataset: handwritten digit classification for digits 0 to 9

SVD classification model

- Initial model from linear algebraic techniques as a baseline
- MNIST dataset: handwritten digit classification for digits 0 to 9
- Split the data into 10 classes, by the label of each data point

SVD classification model

- Initial model from linear algebraic techniques as a baseline
- MNIST dataset: handwritten digit classification for digits 0 to 9
- Split the data into 10 classes, by the label of each data point
- Compute SVD for each of the 10 new design matrices; number of columns is a hyperparameter; yields matrix U_c for class c

SVD classification model

- Initial model from linear algebraic techniques as a baseline
- MNIST dataset: handwritten digit classification for digits 0 to 9
- Split the data into 10 classes, by the label of each data point
- Compute SVD for each of the 10 new design matrices; number of columns is a hyperparameter; yields matrix U_c for class c
- Key idea behind algorithm: PCA equal to SVD

SVD classification model

- Initial model from linear algebraic techniques as a baseline
- MNIST dataset: handwritten digit classification for digits 0 to 9
- Split the data into 10 classes, by the label of each data point
- Compute SVD for each of the 10 new design matrices; number of columns is a hyperparameter; yields matrix U_c for class c
- Key idea behind algorithm: PCA equal to SVD
    - PCA: finding a lower-dimensional representation of the data that preserves the most variance

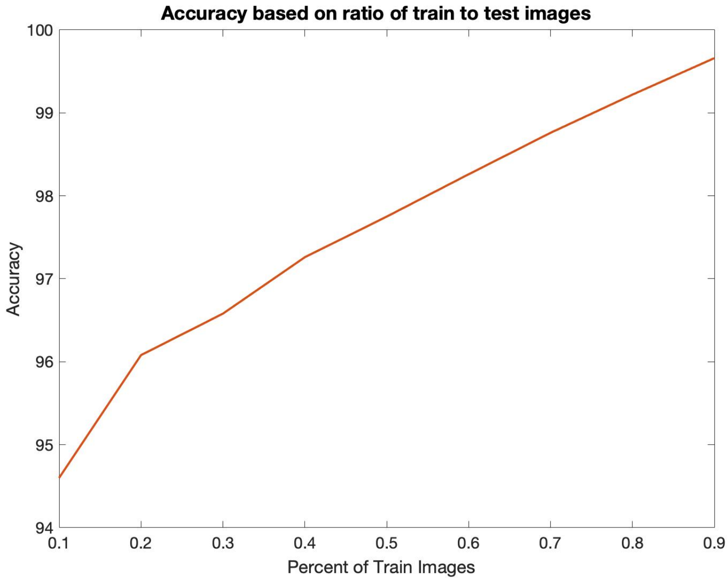
SVD classification model

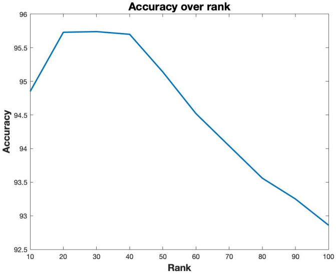
- Initial model from linear algebraic techniques as a baseline
- MNIST dataset: handwritten digit classification for digits 0 to 9
- Split the data into 10 classes, by the label of each data point
- Compute SVD for each of the 10 new design matrices; number of columns is a hyperparameter; yields matrix U_c for class c
- Key idea behind algorithm: PCA equal to SVD
  - PCA: finding a lower-dimensional representation of the data that preserves the most variance
  - Basis vectors come from SVD

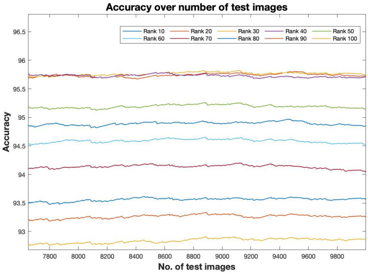
SVD classification model

- Initial model from linear algebraic techniques as a baseline
- MNIST dataset: handwritten digit classification for digits 0 to 9
- Split the data into 10 classes, by the label of each data point
- Compute SVD for each of the 10 new design matrices; number of columns is a hyperparameter; yields matrix U_c for class c
- Key idea behind algorithm: PCA equal to SVD
  - PCA: finding a lower-dimensional representation of the data that preserves the most variance
  - Basis vectors come from SVD
- Classification objective: $\underset{c=0,1,\ldots,9}{\operatorname{argmin}} ||x - U_c(U_c^T x)||_2$

Results from SVD Classification Algorithm



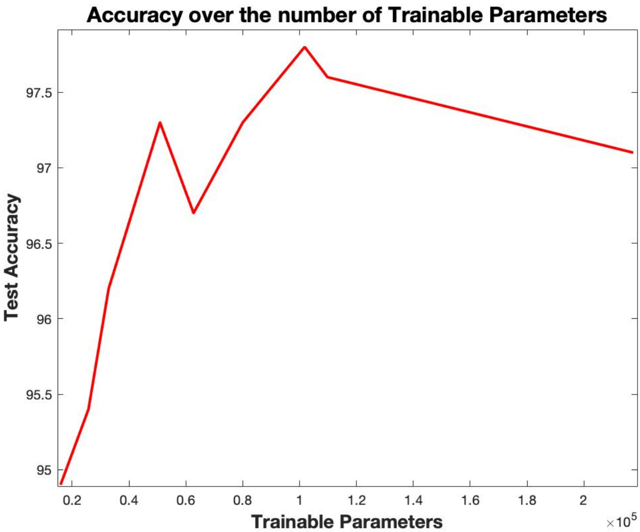

Accuracy over rank
Accuracy
Rank
96
95.5
95
94.5
94
93.5
93
92.5
10
20
30
40
50
60
70
80
90
100


Accuracy over number of test images
Rank 10
Rank 20
Rank 30
Rank 40
Rank 50
Rank 60
Rank 70
Rank 80
Rank 90
Rank 100
Accuracy
No. of test images
96.5
96
95.5
95
94.5
94
93.5
93
7800
8000
8200
8400
8600
8800
9000
9200
9400
9600
9800

Neural networks for MNIST

- Feedforward neural networks: different architectures
  - Best performance, 1 hidden layer, with 128 units over 12 EPOCHS, 97.8% accuracy
- Convolutional neural network: two convolutional layers (convolution + max pooling) and three fully connected layers (one of them being output)
- CNN was the best model, achieving 98.35 percent accuracy after 3 epochs (nearing 99 percent accuracy after roughly 10 epochs)

Feedforward Network Results



Motivating questions in NN approximation theory

- What class of functions can be approximated/expressed by a standard feedforward neural network of depth k?

Motivating questions in NN approximation theory

- What class of functions can be approximated/expressed by a standard feedforward neural network of depth k?
- How do we think about the shift to rectified activations from sigmoidal functions, which perform well in practice?

Motivating questions in NN approximation theory

- What class of functions can be approximated/expressed by a standard feedforward neural network of depth k?
- How do we think about the shift to rectified activations from sigmoidal functions, which perform well in practice?
- What subset of all functions that are provably approximable by neural networks are actually learnable in practice?

Classical Results: Shallow Neural Networks as Universal Approximators

- **Definition:** let ϵ-*approximation* of a function $f(\mathbf{x})$ by another function $F(\mathbf{x})$ with a shared domain $\mathbf{X}$ denote that for arbitrary $\epsilon > 0$,

$$sup_{\mathbf{x} \in \mathbf{X}} |f(\mathbf{x}) - F(\mathbf{x})| < \epsilon$$

Classical Results: Shallow Neural Networks as Universal Approximators

- **Definition:** let ϵ-*approximation* of a function $f(\mathbf{x})$ by another function $F(\mathbf{x})$ with a shared domain $\mathbf{X}$ denote that for arbitrary $\epsilon > 0$,

$$sup_{\mathbf{x} \in \mathbf{X}} |f(\mathbf{x}) - F(\mathbf{x})| < \epsilon$$

- **Definition:** Call a function $\sigma(t)$ *sigmoidal* if

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

Classical Results: Shallow Neural Networks as Universal Approximators

- **Definition:** let ϵ-*approximation* of a function $f(\mathbf{x})$ by another function $F(\mathbf{x})$ with a shared domain $\mathbf{X}$ denote that for arbitrary $\epsilon > 0$,
$$sup_{\mathbf{x}\in\mathbf{X}}|f(\mathbf{x}) - F(\mathbf{x})| < \epsilon$$
- **Definition:** Call a function $\sigma(t)$ *sigmoidal* if
$$\sigma(t) \rightarrow \begin{cases} 1 & \text{as } t \rightarrow \infty \\ 0 & \text{as } t \rightarrow -\infty \end{cases}$$
- Cybenko (1989): Shallow neural networks (one hidden layer) with continuous sigmoidal activations are universal approximators, i.e. capable of ϵ-approximating any continuous function defined on the unit hypercube.

Classical Results: Shallow Neural Networks as Universal Approximators

- Loosening of restrictions on activation function that still yield notion of ϵ-approximation:
    - Hornik (1990): extension to any continuous and bounded activation functions, support extends to more than just the unit hypercube
    - Leshno (1993): extension to nonpolynomial activation functions

Classical Results: Shallow Neural Networks as Universal Approximators

- Loosening of restrictions on activation function that still yield notion of ϵ-approximation:
  - Hornik (1990): extension to any continuous and bounded activation functions, support extends to more than just the unit hypercube
  - Leshno (1993): extension to nonpolynomial activation functions
- By extension, deeper neural networks of depth k also enjoy the same theoretical guarantees.

Improved Expressivity Results with Depth

Let k denote the depth of a network. The following results are due to Rolnick and Tegmark (2018).

Improved Expressivity Results with Depth

Let k denote the depth of a network. The following results are due to Rolnick and Tegmark (2018).

- Let $f(\mathbf{x})$ be a multivariate polynomial function of finite degree d, and $N(x)$ be a network with nonlinear activation having nonzero Taylor coefficients up to degree d. Then there exists a number of neurons $m_k(f)$ that can ϵ-approximate $f(\mathbf{x})$, where m_k is *independent of* ϵ.

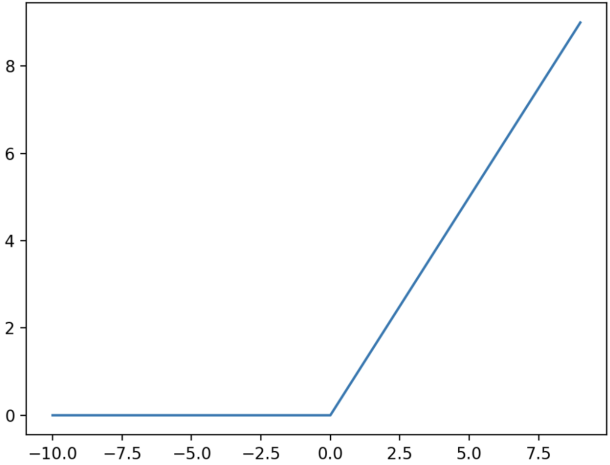
Improved Expressivity Results with Depth

Let k denote the depth of a network. The following results are due to Rolnick and Tegmark (2018).

- Let $f(\mathbf{x})$ be a multivariate polynomial function of finite degree d, and $N(x)$ be a network with nonlinear activation having nonzero Taylor coefficients up to degree d. Then there exists a number of neurons $m_k(f)$ that can ϵ-approximate $f(\mathbf{x})$, where m_k is *independent of* ϵ.
- Let $f(\mathbf{x}) = x_1^{r_1} x_2^{r_2} \dots x_n^{r_n}$ be a monomial function of finite terms, network $N(x)$ has nonlinear activation having nonzero Taylor coefficients up to degree $2d$, and $m_k(f)$ defined as above. Then $m_1(f)$ is exponential, but linear in a log-depth network.

$$m_1(f) = \Pi_{i=1}^{n}(r_i + 1)$$
$$m(f) \leq \Sigma_{i=1}^{n} 7\lceil log_2(r_i)\rceil + 4$$

ReLU Networks as Partitioning Input Space



A ReLU activation function - used between affine transformations to introduce nonlinearities in the learned function.

Review: ReLU Networks as Partitioning Input Space

Known theorems:

Review: ReLU Networks as Partitioning Input Space

Known theorems:

- ReLU networks are bijective to the appropriate class of piecewise linear functions (up to isomorphism of network)

Review: ReLU Networks as Partitioning Input Space

Known theorems:

- ReLU networks are bijective to the appropriate class of piecewise linear functions (up to isomorphism of network)
- For one hidden layer neural networks (p-dimensional input, q-dimensional hidden layer): can combinatorially show an upper bound on the number of piecewise linear regions:

$$\text{r(q,p)} = \sum_{i=0}^{p} \binom{q}{i}$$

Review: ReLU Networks as Partitioning Input Space

Known theorems:

- ReLU networks are bijective to the appropriate class of piecewise linear functions (up to isomorphism of network)
- For one hidden layer neural networks (p-dimensional input, q-dimensional hidden layer): can combinatorially show an upper bound on the number of piecewise linear regions:

$$\mathrm{r(q,p)} = \sum_{i=0}^{p} \binom{q}{i}$$

- Montufar et al. (2014): The *maximal* number of linear regions of the functions computed by a NN with n_0 input units and L hidden layers, with $n_i \geq n_0$ rectifiers at the ith layer, is lower bounded by

$$\left(\prod_{i=1}^{L-1} \left\lfloor \frac{n_i}{n_0} \right\rfloor^{n_0}\right)\left(\Sigma_{j=0}^{n_0} \binom{n_L}{j}\right)$$

Visualizing the hyperplanes to depth k

Visualizing the hyperplanes to depth k

- First layer: hyperplanes through input space

Visualizing the hyperplanes to depth k

- First layer: hyperplanes through input space
- All proceeding layers: "bent hyperplanes" that bend at the established bent hyperplanes of previous layers

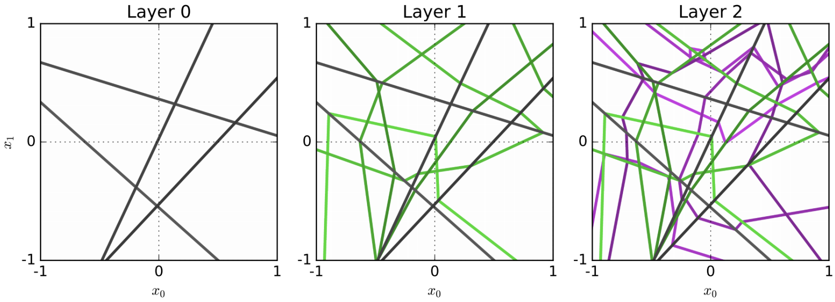


Figure 1 of Raghu et al. (2017)

The following is from the work of Hanin and Rolnick (2019ab), and concerns ReLU networks. The results are stated informally.

Expressibility vs. Learnability: Theorems

The following is from the work of Hanin and Rolnick (2019ab), and concerns ReLU networks. The results are stated informally.

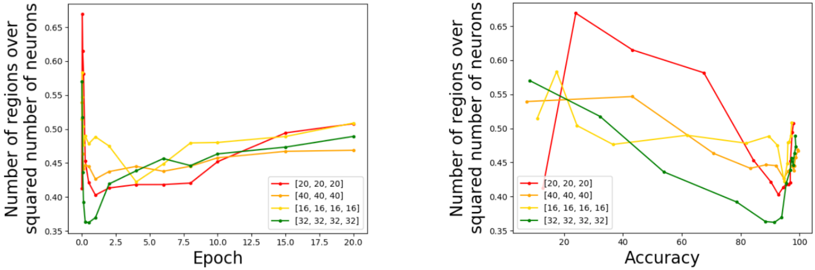
- For any line segment through input space, the average number of regions intersected is linear, and not exponential, in the number of neurons.

Expressibility vs. Learnability: Theorems

The following is from the work of Hanin and Rolnick (2019ab), and concerns ReLU networks. The results are stated informally.

- For any line segment through input space, the average number of regions intersected is linear, and not exponential, in the number of neurons.
- Both the number of regions and the distance to the nearest region boundary stay roughly constant during training.

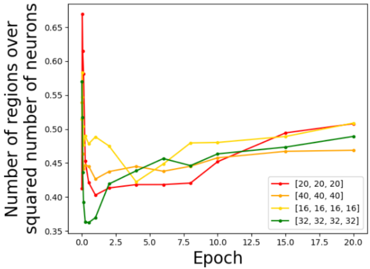
Expressibility vs. Learnability: Graphs



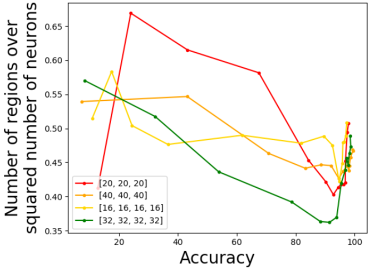
(a) Over Epochs (b) Over Test Accuracy

Figure: Normalization by squared number of neurons

Expressibility vs. Learnability: Graphs



(a) Over Epochs

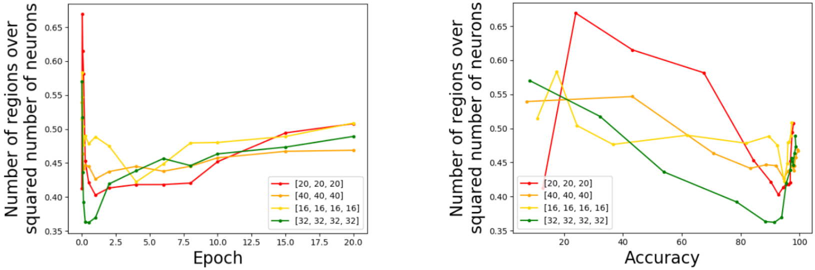


(b) Over Test Accuracy

Figure: Normalization by squared number of neurons

- Across different networks, number of piecewise linear regions is $O(n^2)$, and this doesn't change with greater depth.

Expressibility vs. Learnability: Graphs



(a) Over Epochs (b) Over Test Accuracy

Figure: Normalization by squared number of neurons

- Across different networks, number of piecewise linear regions is $O(n^2)$, and this doesn't change with greater depth.
- Upshot: empirical success of depth on certain problems is not because deep nets learn a complex function inaccessible to shallow networks.

Bridging the Gap

Two next questions (in context of ReLU networks):

Bridging the Gap

Two next questions (in context of ReLU networks):

- Are networks that learn an exponential number of linear regions "usable", or are they purely a theoretical guarantee?

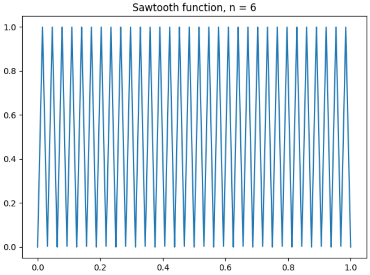
Bridging the Gap

Two next questions (in context of ReLU networks):

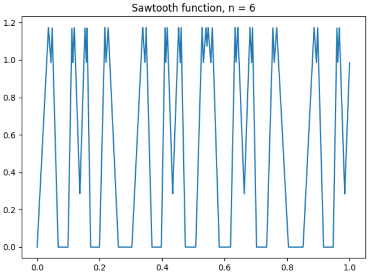
- Are networks that learn an exponential number of linear regions "usable", or are they purely a theoretical guarantee?
- How can we adjust the traditional deep learning pipeline to allow learning of piecewise linear functions in the exponential regime?

Sawtooth functions/triangular wave functions

Type of function that achieves an exponential number of regions in number of nodes/depth.



(a) No Perturbation



(b) Random Perturbation

Figure: Sawtooth functions

Sawtooth functions as feedforward neural networks

To achieve a feedforward neural network that represents a sawtooth function with 2^n affine regions in $2n$ hidden layers:

Sawtooth functions as feedforward neural networks

To achieve a feedforward neural network that represents a sawtooth function with 2^n affine regions in $2n$ hidden layers:

- Mirror map, defined on $[0, 1]$:

$$\mathrm{f(x)} = \begin{cases} 2x & \text{when } 0 \le x \le \frac{1}{2} \\ 2(1-x) & \text{when } \frac{1}{2} \le x \le 1 \end{cases}$$

Sawtooth functions as feedforward neural networks

To achieve a feedforward neural network that represents a sawtooth function with 2^n affine regions in $2n$ hidden layers:

- Mirror map, defined on $[0, 1]$:

$$\text{f(x)} = \begin{cases} 2x & \text{when } 0 \leq x \leq \frac{1}{2} \\ 2(1-x) & \text{when } \frac{1}{2} \leq x \leq 1 \end{cases}$$

- As a two-layer neural network:

$$f(x) = ReLU(2ReLU(x) - 4ReLU(x - \tfrac{1}{2}))$$

Sawtooth functions as feedforward neural networks

To achieve a feedforward neural network that represents a sawtooth function with 2^n affine regions in $2n$ hidden layers:

- Mirror map, defined on $[0, 1]$:

$$\text{f(x)} = \begin{cases} 2x & \text{when } 0 \le x \le \frac{1}{2} \\ 2(1-x) & \text{when } \frac{1}{2} \le x \le 1 \end{cases}$$

- As a two-layer neural network:

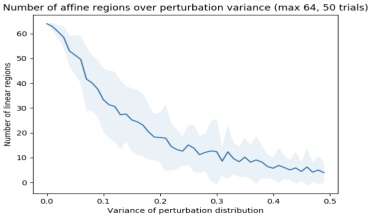
$$f(x) = ReLU(2ReLU(x) - 4ReLU(x - \tfrac{1}{2}))$$

- Composing mirror map with itself n times will yield a sawtooth function with 2^n equally spaced affine components on $[0, 1]$.

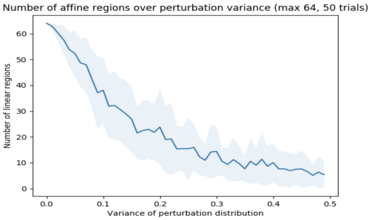
Questions regarding sawtooth functions

- How robust are sawtooth functions to multiplicative weight perturbation, of the form $w(1+\epsilon)$? (The perturbations are zero-mean Gaussians, and experiments changed the variance.)
- Can randomly initialized or perturbed networks re-learn the sawtooth function?

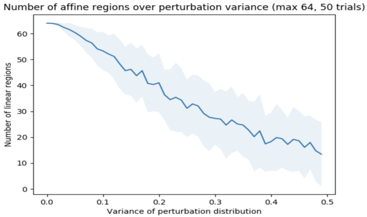
Perturbing the parameters



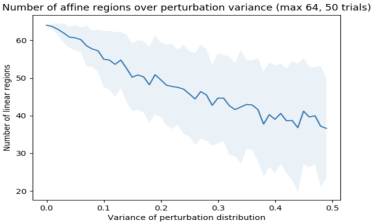
(a) All weights and biases



(b) All weights

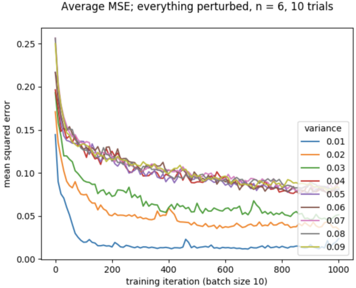


(c) All biases

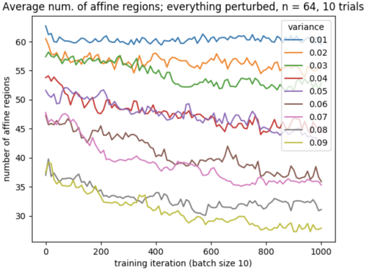


(d) First layer only

Re-learning sawtooth networks from different initializations

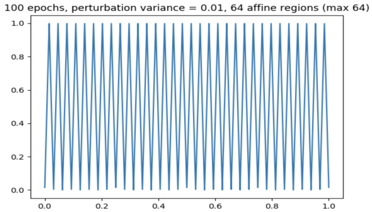


(a) Mean squared error

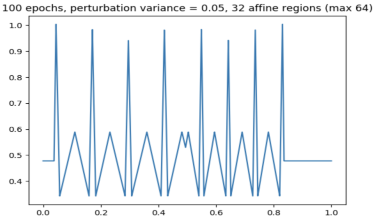


(b) Number of linear regions

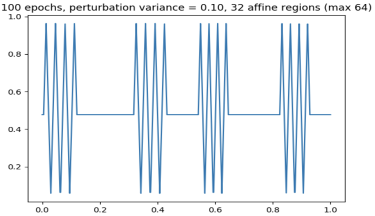
Re-learning sawtooth networks from different initializations



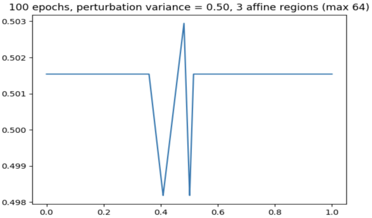
(c) Variance = 0.01



(d) Variance = 0.05



(e) Variance = 0.1



(f) Variance = 0.5

Implications

- "Sawtooth networks" fall apart with mild variance on the weights.
- Even when starting from perturbed versions of the original function, the original sawtooth network is not retained, implying that the set of parameters yielding exponential nets in the loss landscape is localized and challenging to learn.

Learning functions in the exponential regime (sketch)

How to adjust the DL framework to encourage feedforward model to learn functions currently in $F_{express} \setminus F_{learn}$?

Learning functions in the exponential regime (sketch)

How to adjust the DL framework to encourage feedforward model to learn functions currently in $F_{express} \setminus F_{learn}$?

- First attempt: adding terms to the objective function to encourage network to learn more complex functions ("anti-regularization")

Learning functions in the exponential regime (sketch)

How to adjust the DL framework to encourage feedforward model to learn functions currently in $F_{express} \setminus F_{learn}$?

- First attempt: adding terms to the objective function to encourage network to learn more complex functions ("anti-regularization")
- For 2D input, two hidden-layer ReLU nets, can think about encouraging all hyperplanes/bent hyperplanes to intersect:

Learning functions in the exponential regime (sketch)

How to adjust the DL framework to encourage feedforward model to learn functions currently in $F_{express} \setminus F_{learn}$?

- First attempt: adding terms to the objective function to encourage network to learn more complex functions ("anti-regularization")
- For 2D input, two hidden-layer ReLU nets, can think about encouraging all hyperplanes/bent hyperplanes to intersect:
  - Can think of an inactive ReLU activation function as replacing appropriate column of outer weight vector by zeros

$$w_2 ReLU(W_1 x + b_1) + b_2 = w_2'((W_1 x + b_1) + b_2)$$

Learning functions in the exponential regime (sketch)

How to adjust the DL framework to encourage feedforward model to learn functions currently in $F_{express} \setminus F_{learn}$?

- First attempt: adding terms to the objective function to encourage network to learn more complex functions ("anti-regularization")
- For 2D input, two hidden-layer ReLU nets, can think about encouraging all hyperplanes/bent hyperplanes to intersect:
  - Can think of an inactive ReLU activation function as replacing appropriate column of outer weight vector by zeros
  $$w_2 ReLU(W_1 x + b_1) + b_2 = w_2'((W_1 x + b_1) + b_2)$$
  - For a first-layer activation pattern, this yields a square matrix equation and a collection of inequalities, and one can add "perceptron-like" error terms to the objective function to encourage regions to intersect

Conclusion

- Thank you for listening!