Finding Low-Rank Functions Using Linear Layers in Neural Networks

Sue Parkinson

University of Chicago Committee on Computational and Applied Mathematics

Table of contents

Background/Motivation

Why do Neural Networks Perform Well?

Theorem (Universal Approximation Theorem for Wide Networks)

Arbitrarily wide neural networks with nonlinear activation functions can approximate any continuous function arbitrarily well. [4]

Why do *Deep* Neural Networks Perform Well?

Several approaches...

- Universal Approximators

Theorem (Universal Approximation Theorem for Deep Networks)

Width $n+4$ ReLU networks can approximate any Lebesgue integrable function on an n-dimensional input space w.r.t. the L^1 distance arbitrarily well if depth is allowed to grow arbitrarily. [2]

If width $\leq n$, this is no longer true.

- *Depth Separation Analysis* $\exists f$ which can be efficiently represented at one depth but require exponential width to represent them with shallower network. [1, 6] Such functions are often high oscillatory; results don't hold for functions with bounded Lipchitz constant. [5]

Question

Why does adding layers to a neural network improve performance in approximating the *same* function?

Why do *Deep* Neural Networks Perform Well?

Several approaches...

- Universal Approximators

Theorem (Universal Approximation Theorem for Deep Networks)

Width $n + 4$ ReLU networks can approximate any Lebesgue integrable function on an n-dimensional input space w.r.t. the L^1 distance arbitrarily well if depth is allowed to grow arbitrarily. [2]

If width $\leq n$, this is no longer true.

- *Depth Separation Analysis* $\exists f$ which can be efficiently represented at one depth but require exponential width to represent them with shallower network. [1, 6] Such functions are often high oscillatory; results don't hold for functions with bounded Lipchitz constant. [5]

Question

Why does adding layers to a neural network improve performance in approximating the *same* function?

Why do *Deep* Neural Networks Perform Well?

Several approaches...

- Universal Approximators

Theorem (Universal Approximation Theorem for Deep Networks)

Width $n + 4$ ReLU networks can approximate any Lebesgue integrable function on an n-dimensional input space w.r.t. the L^1 distance arbitrarily well if depth is allowed to grow arbitrarily. [2]

  If width $\leq n$, this is no longer true.

- *Depth Separation Analysis* $\exists f$ which can be efficiently represented at one depth but require exponential width to represent them with shallower network. [1, 6] Such functions are often high oscillatory; results don't hold for functions with bounded Lipchitz constant. [5]

Question

Why does adding layers to a neural network improve performance in approximating the *same* function?

Why do *Deep* Neural Networks Perform Well?

Several approaches...

- Universal Approximators

Theorem (Universal Approximation Theorem for Deep Networks)

Width $n + 4$ ReLU networks can approximate any Lebesgue integrable function on an n-dimensional input space w.r.t. the L^1 distance arbitrarily well if depth is allowed to grow arbitrarily. [2]

  If width $\leq n$, this is no longer true.
- *Depth Separation Analysis* $\exists f$ which can be efficiently represented at one depth but require exponential width to represent them with shallower network. [1, 6] Such functions are often high oscillatory; results don't hold for functions with bounded Lipchitz constant. [5]

Question

Why does adding layers to a neural network improve performance in approximating the *same* function?

Setup

- Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma(\mathbf{W}\mathbf{x} - \mathbf{b}) + c$$

- Deep Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma_3(\mathbf{W}_3\sigma_2(\mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{x} - b_1) - b_2) - b_3) + c$$

- ReLU Networks: $\sigma(x) = \max(x, 0) := [x]_+$
- Ideally, we could answer why ReLU activation deep neural networks work as they do
- Simplify by assuming previous layers are linear

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i\mathbf{x} - b\right]_+ + c$$

- Why would this help?

Setup

- Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma(\mathbf{W}\mathbf{x} - \mathbf{b}) + c$$

- Deep Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma_3(\mathbf{W}_3 \sigma_2(\mathbf{W}_2 \sigma_1(\mathbf{W}_1 \mathbf{x} - b_1) - b_2) - b_3) + c$$

- ReLU Networks: $\sigma(x) = \max(x, 0) := [x]_+$
- Ideally, we could answer why ReLU activation deep neural networks work as they do
- Simplify by assuming previous layers are linear

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i \mathbf{x} - b \right]_+ + c$$

- Why would this help?

Setup

- Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma(\mathbf{W}\mathbf{x} - \mathbf{b}) + c$$

- Deep Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma_3(\mathbf{W}_3 \sigma_2(\mathbf{W}_2 \sigma_1(\mathbf{W}_1 \mathbf{x} - b_1) - b_2) - b_3) + c$$

- ReLU Networks: $\sigma(x) = \max(x, 0) := [x]_+$
- Ideally, we could answer why ReLU activation deep neural networks work as they do
- Simplify by assuming previous layers are linear

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i \mathbf{x} - b \right]_+ + c$$

- Why would this help?

Setup

- Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma(\mathbf{W}\mathbf{x} - \mathbf{b}) + c$$

- Deep Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma_3(\mathbf{W}_3\sigma_2(\mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{x} - b_1) - b_2) - b_3) + c$$

- ReLU Networks: $\sigma(x) = \max(x, 0) := [x]_+$
- Ideally, we could answer why ReLU activation deep neural networks work as they do
- Simplify by assuming previous layers are linear

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i\mathbf{x} - b\right]_+ + c$$

- Why would this help?

Setup

- Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma(\mathbf{W}\mathbf{x} - \mathbf{b}) + c$$

- Deep Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma_3(\mathbf{W}_3 \sigma_2(\mathbf{W}_2 \sigma_1(\mathbf{W}_1 \mathbf{x} - b_1) - b_2) - b_3) + c$$

- ReLU Networks: $\sigma(x) = \max(x, 0) := [x]_+$
- Ideally, we could answer why ReLU activation deep neural networks work as they do
- Simplify by assuming previous layers are linear

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i \mathbf{x} - b \right]_+ + c$$

- Why would this help?

Setup

- Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma(\mathbf{W}\mathbf{x} - \mathbf{b}) + c$$

- Deep Neural Network:

$$f(\mathbf{x}) = \mathbf{a}^\top \sigma_3(\mathbf{W}_3\sigma_2(\mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{x} - b_1) - b_2) - b_3) + c$$

- ReLU Networks: $\sigma(x) = \max(x, 0) := [x]_+$
- Ideally, we could answer why ReLU activation deep neural networks work as they do
- Simplify by assuming previous layers are linear

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i\mathbf{x} - b\right]_+ + c$$

- Why would this help?

Setup: Linear Layers

- $L - 1$ Linear Layers followed by ReLU final layer:

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i \mathbf{x} - b \right]_+ + c$$

- Adding linear layers increase the capacity of a shallow network. Just reparameterizes it
- When we train with weight decay, reparameterizations can affect the associated inductive bias, and therefore which Neural Network is selected

Setup: Linear Layers

- $L-1$ Linear Layers followed by ReLU final layer:

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i \mathbf{x} - b\right]_+ + c$$

- Adding linear layers increase the capacity of a shallow network. Just reparameterizes it
- When we train with weight decay, reparameterizations can affect the associated inductive bias, and therefore which Neural Network is selected

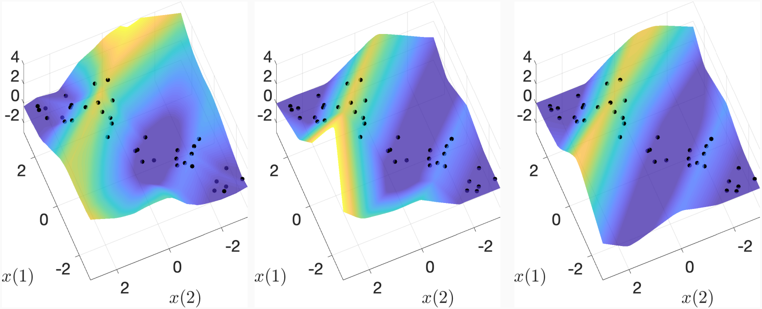
Setup: Linear Layers

- $L - 1$ Linear Layers followed by ReLU final layer:

$$f(\mathbf{x}) = \mathbf{a}^\top \left[\prod_{i=1}^{L-1} \mathbf{W}_i \mathbf{x} - b \right]_+ + c$$

- Adding linear layers increase the capacity of a shallow network. Just reparameterizes it
- When we train with weight decay, reparameterizations can affect the associated inductive bias, and therefore which Neural Network is selected

Example



Inductive bias

Inductive bias

Regularized Empirical Risk Minimization Framework

- Parameterization view:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\mathbf{x}_i), y_i) + \eta \underbrace{C_L(\theta)}_{\text{Regularization}}$$

- Function-space view:

$$\hat{f} = \arg\min_{f} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{R_L(f)}_{\text{Regularization}}$$

- Weight Decay:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{\frac{1}{L}\left(\|\mathbf{a}\|_2^2 + \sum_{j=1}^{L-1} \|\mathbf{W}_j\|_F^2 \right)}_{C_L(\theta)}$$

- Inductive bias: What is $R_L(f)$? What kinds of functions will we learn using regularization penalty R_L?

Inductive bias

Regularized Empirical Risk Minimization Framework

- Parameterization view:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\mathbf{x}_i), y_i) + \eta \underbrace{C_L(\theta)}_{\text{Regularization}}$$

- Function-space view:

$$\hat{f} = \arg\min_{f} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{R_L(f)}_{\text{Regularization}}$$

- Weight Decay:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{\frac{1}{L}\left(\|\mathbf{a}\|_2^2 + \sum_{j=1}^{L-1} \|\mathbf{W}_j\|_F^2 \right)}_{C_L(\theta)}$$

- Inductive bias: What is $R_L(f)$? What kinds of functions will we learn using regularization penalty R_L?

Inductive bias

Regularized Empirical Risk Minimization Framework

- Parameterization view:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\mathbf{x}_i), y_i) + \eta \underbrace{C_L(\theta)}_{\text{Regularization}}$$

- Function-space view:

$$\hat{f} = \arg\min_{f} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{R_L(f)}_{\text{Regularization}}$$

- Weight Decay:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{\frac{1}{L}\left(\|\mathbf{a}\|_2^2 + \sum_{j=1}^{L-1} \|\mathbf{W}_j\|_F^2\right)}_{C_L(\theta)}$$

- Inductive bias: What is $R_L(f)$? What kinds of functions will we learn using regularization penalty R_L?

Inductive bias

Regularized Empirical Risk Minimization Framework

- Parameterization view:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\mathbf{x}_i), y_i) + \eta \underbrace{C_L(\theta)}_{\text{Regularization}}$$

- Function-space view:

$$\hat{f} = \arg\min_{f} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{R_L(f)}_{\text{Regularization}}$$

- Weight Decay:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) + \eta \underbrace{\frac{1}{L} \left(\|\mathbf{a}\|_2^2 + \sum_{j=1}^{L-1} \|\mathbf{W}_j\|_F^2 \right)}_{C_L(\theta)}$$

- Inductive bias: What is $R_L(f)$? What kinds of functions will we learn using regularization penalty R_L?

Notation

- $f : \mathbb{R}^d \to \mathbb{R}$
- Neural network parameterizations

$$\theta = (\mathbf{W}, \mathbf{a}, \mathbf{b}, c) = \left(\prod_{i=1}^{L-1} \mathbf{W}_i, \mathbf{a}, \mathbf{b}, c \right)$$

- Denote row k in $\mathbf{W}$ by $\mathbf{w}_k$, and number of ReLU units (i.e. rows in $\mathbf{W}$) is K
- A generic neural network with parameterization θ:

$$h_\theta(\mathbf{x}) = \mathbf{a}^\top [\mathbf{W}\mathbf{x} - \mathbf{b}]_+ + c = \sum_{k=1}^{K} a_k \left[\mathbf{w}_k^\top \mathbf{x} - b_k\right]_+ + c$$

Definition

$$R_L(f) := \min_{\theta : h_\theta^{(2)} = f} C_L(\theta) = \min_{\theta : h_\theta^{(2)} = f} \frac{1}{L} \left(\|\mathbf{a}\|_2^2 + \sum_{j=1}^{L-1} \|\mathbf{W}_j\|_F^2 \right)$$

Expressions for R_L

Definition (Schatten (Quasi)-Norm)

Given a matrix M,

$$\|\mathsf{M}\|_{S^q} := \left(\sum_{i=1}^{\operatorname{rank}(\mathsf{M})} \sigma_i(\mathsf{M})^q \right)^{1/q}.$$

This is a norm for $q \in [1, \infty]$ and a quasi-norm for $q \in (0, 1)$

Fact

As $q \to 0$, $\|\mathsf{M}\|_{S^q}^q \to \operatorname{rank}(\mathsf{M})$

Lemma (Ongie & Willett)

$$R_L(f) = \min_{\theta : h_\theta^{(2)} = f} \frac{1}{L} \|\mathbf{a}\|_2^2 + \frac{L-1}{L} \|\mathsf{W}\|_{S^{2/(L-1)}}^{2/(L-1)}$$

Rescaling Invariance

- Observe that $\forall \lambda > 0$,

$$a_k \left[\mathbf{w}_k^\top \mathbf{x} - b_k\right]_+ + c = \frac{a_k}{\lambda}\left[\lambda \mathbf{w}_k^\top \mathbf{x} - b_k\right]_+ + c$$

- Similarly, $\forall \lambda \succ 0$,

$$\mathbf{a}^\top [\mathbf{W}\mathbf{x} - b]_+ + c = \left(\mathbf{D}_\lambda^{-1}\mathbf{a}\right)^\top [\mathbf{D}_\lambda \mathbf{W}\mathbf{x} - b]_+ + c$$

- Using this rescaling invariance, we get

$$R_L(f) = \min_{\theta : h_\theta^{(2)} = f} \underbrace{\inf_{\lambda \succ 0} \frac{1}{L}\|\mathbf{D}_\lambda^{-1}\mathbf{a}\|_2^2 + \frac{L-1}{L}\|\mathbf{D}_\lambda \mathbf{W}\|_{S^{2/(L-1)}}^{2/(L-1)}}_{\Phi_L(\mathbf{W},\mathbf{a})}$$

Rescaling Invariance

- Observe that $\forall \lambda > 0$,

$$a_k \left[\mathbf{w}_k^\top \mathbf{x} - b_k\right]_+ + c = \frac{a_k}{\lambda} \left[\lambda \mathbf{w}_k^\top \mathbf{x} - b_k\right]_+ + c$$

- Similarly, $\forall \lambda \succ 0$,

$$\mathbf{a}^\top \left[\mathbf{W}\mathbf{x} - b\right]_+ + c = \left(\mathbf{D}_\lambda^{-1}\mathbf{a}\right)^\top \left[\mathbf{D}_\lambda \mathbf{W}\mathbf{x} - b\right]_+ + c$$

- Using this rescaling invariance, we get

$$R_L(f) = \min_{\theta : h_\theta^{(2)} = f} \underbrace{\inf_{\lambda \succ 0} \frac{1}{L}\|\mathbf{D}_\lambda^{-1}\mathbf{a}\|_2^2 + \frac{L-1}{L}\|\mathbf{D}_\lambda \mathbf{W}\|_{S^{2/(L-1)}}^{2/(L-1)}}_{\Phi_L(\mathbf{W},\mathbf{a})}$$

Rescaling Invariance

- Observe that $\forall \lambda > 0$,

$$a_k \left[\mathbf{w}_k^\top \mathbf{x} - b_k \right]_+ + c = \frac{a_k}{\lambda} \left[\lambda \mathbf{w}_k^\top \mathbf{x} - b_k \right]_+ + c$$

- Similarly, $\forall \lambda \succ 0$,

$$\mathbf{a}^\top \left[\mathbf{W}\mathbf{x} - b \right]_+ + c = \left(\mathbf{D}_\lambda^{-1} \mathbf{a} \right)^\top \left[\mathbf{D}_\lambda \mathbf{W}\mathbf{x} - b \right]_+ + c$$

- Using this rescaling invariance, we get

$$R_L(f) = \min_{\theta : h_\theta^{(2)} = f} \underbrace{\inf_{\lambda \succ 0} \frac{1}{L} \|\mathbf{D}_\lambda^{-1} \mathbf{a}\|_2^2 + \frac{L-1}{L} \|\mathbf{D}_\lambda \mathbf{W}\|_{S^{2/(L-1)}}^{2/(L-1)}}_{\Phi_L(\mathbf{W}, \mathbf{a})}$$

Expression for Φ_L

Lemma (Ongie & Willett)

$$\Phi_L(\mathbf{W}, \mathbf{a}) = \inf_{\substack{\lambda \succ 0 \\ \|\lambda\|_2 = 1}} \|\mathbf{D}_\lambda^{-1} \mathbf{D}_\mathbf{a} \mathbf{W}\|_{S^{2/(L-1)}}^{2/L}$$

Definition (Path Norm)

When $L = 2$, the infimum in Φ_L can be computed explicitly as

$$\Phi_2(\mathbf{W}, \mathbf{a}) = \sum_{k=1}^{K} |a_k| \|\mathbf{w}_k\|_2$$

and

$$R_2(f) = \min_{\theta : h_\theta^{(2)} = f} \sum_{k=1}^{K} |a_k| \|\mathbf{w}_k\|_2.$$

This is sometimes called the *path norm*. [3]

Bounds for Φ_L

Lemma (P & Ongie & Willett)

$$\|\mathbf{D_a W}\|_{S^{2/L}}^{2/L} \leq \Phi_L(\mathbf{W}, \mathbf{a}) \leq \operatorname{rank}(\mathbf{D_a W})^{\frac{L-2}{L}} \left(\sum_{k=1}^{K} |a_k| \|\mathbf{w}_k\|_2 \right)^{2/L}$$

Mixed-Variation Functions

Mixed-Variation Functions

Definition

A function f is a *mixed-variation function* if $f(\mathsf{x}) = f(\mathsf{P}_S\mathsf{x})\ \forall x$ where P_S is the projection onto a subspace S. The subspace S of smallest dimension satisfying the above is called the *active subspace* of f. The *rank* of f is the dimension of the active subspace.

- V an orthonormal basis for $S \implies \mathsf{P}_S = \mathsf{V}\mathsf{V}^\top$
- $f(\mathsf{x}) = g(\mathsf{V}^\top x)$ for some function $g : \mathbb{R}^r \to \mathbb{R}$
- $\forall f$ that can be represented as a two-layer neural network,

$$\operatorname{rank}(f) = \min_{\theta : h_\theta^{(2)} = f} \operatorname{rank}(D_{\mathsf{a}}\mathsf{W})$$

Mixed-Variation Functions

Definition

A function f is a *mixed-variation function* if $f(\mathbf{x}) = f(\mathbf{P}_S\mathbf{x})\ \forall x$ where $\mathbf{P}_S$ is the projection onto a subspace S. The subspace S of smallest dimension satisfying the above is called the *active subspace* of f. The *rank* of f is the dimension of the active subspace.

- $\mathbf{V}$ an orthonormal basis for $S \implies \mathbf{P}_S = \mathbf{V}\mathbf{V}^\top$
- $f(\mathbf{x}) = g(\mathbf{V}^\top x)$ for some function $g : \mathbb{R}^r \to \mathbb{R}$
- $\forall f$ that can be represented as a two-layer neural network,

$$\mathrm{rank}(f) = \min_{\theta : h_\theta^{(2)} = f} \mathrm{rank}(D_{\mathbf{a}}\mathbf{W})$$

Mixed-Variation Functions

Definition

A function f is a *mixed-variation function* if $f(\mathbf{x}) = f(\mathbf{P}_S \mathbf{x})\ \forall x$ where $\mathbf{P}_S$ is the projection onto a subspace S. The subspace S of smallest dimension satisfying the above is called the *active subspace* of f. The *rank* of f is the dimension of the active subspace.

- $\mathbf{V}$ an orthonormal basis for $S \implies \mathbf{P}_S = \mathbf{V}\mathbf{V}^\top$
- $f(\mathbf{x}) = g(\mathbf{V}^\top x)$ for some function $g : \mathbb{R}^r \to \mathbb{R}$
- $\forall f$ that can be represented as a two-layer neural network,

$$\mathrm{rank}(f) = \min_{\theta : h_\theta^{(2)} = f} \mathrm{rank}(D_{\mathbf{a}} \mathbf{W})$$

Mixed-Variation Functions

Definition

A function f is a *mixed-variation function* if $f(\mathsf{x}) = f(\mathsf{P}_S\mathsf{x})\ \forall x$ where P_S is the projection onto a subspace S. The subspace S of smallest dimension satisfying the above is called the *active subspace* of f. The *rank* of f is the dimension of the active subspace.

- V an orthonormal basis for $S \implies \mathsf{P}_S = \mathsf{V}\mathsf{V}^\top$
- $f(\mathsf{x}) = g(\mathsf{V}^\top x)$ for some function $g : \mathbb{R}^r \to \mathbb{R}$
- $\forall f$ that can be represented as a two-layer neural network,

$$\mathrm{rank}(f) = \min_{\theta : h_\theta^{(2)} = f} \mathrm{rank}(D_{\mathsf{a}}\mathsf{W})$$

Function-Rank Bound on R_L

Lemma (P & Ongie & Willett)

$$\min_{\theta : h_\theta^{(2)} = f} \|\mathrm{D_a W}\|_{S^{2/L}}^{2/L} \leq R_L(f) \leq \mathrm{rank}(f)^{\frac{L-2}{L}} R_2(f)^{2/L}$$

Gradient Covariance Matrix

Definition

Fix a probability distribution ρ on $\mathbb{R}^d$. The gradient covariance matrix of a function f is

$$C_f := \mathbb{E}_\rho \left[\nabla f(\mathsf{x}) \nabla f(\mathsf{x})^\top \right]$$

- $\mathrm{rank}(f) = \mathrm{rank}(C_f)$
- If $C_f = \mathsf{V}\Lambda\mathsf{V}^\top$ is an orthonormal eigendecomposition, then V is a basis for the active subspace.

Gradient Covariance Matrix

Definition

Fix a probability distribution ρ on $\mathbb{R}^d$. The gradient covariance matrix of a function f is

$$C_f := \mathbb{E}_\rho \left[\nabla f(\mathsf{x}) \nabla f(\mathsf{x})^\top\right]$$

- $\mathsf{rank}(f) = \mathsf{rank}(C_f)$
- If $C_f = \mathsf{V}\Lambda\mathsf{V}^\top$ is an orthonormal eigendecomposition, then V is a basis for the active subspace.

Single/Multi-Index Model Approach

1. Estimate Active Subspace $\hat{\mathsf{V}}$, e.g. the top r eigenvectors of some empirical estimate of $\hat{C}_f$
2. Estimate $\hat{g}$ in lower-dimensional space
3. Estimated function is $\hat{f}(\mathsf{x}) = \hat{g}(\hat{\mathsf{V}}^\top \mathsf{x})$

Conjecture

Adding linear layers to a neural network effectively does all of this at once, and adaptively chooses dimension of active subspace.

Singular Values of C_f & Mixed-Variation Norm

Let $\sigma_i(f) := \sigma_i(C_f^{1/2})$. Variance of directional derivative associated with eigenvector i of C_f.

Definition

$$\|f\|_{MV,q} := \|C_f^{1/2}\|_{S^q} = \left(\sum_{i=1}^{\mathrm{rank}(f)} \sigma_i(f)^q \right)^{1/q}$$

Lemma (P & Ongie & Willett)

$$\|f\|_{MV,2/(L-1)}^{2/L} \leq R_L(f)$$

Singular Values of Trained Neural Networks

Let

$$\hat{f}_L := \arg\min_f R_L(f) \text{ s.t. } f(\mathbf{x}_j) = y_j \ \forall j.$$

If a rank-r Neural Network interpolant f_r^* of the data exists, then let

$$A_r := \frac{R_2(f_r^*)}{\inf_L \|\hat{f}_L\|_{MV,\infty}}.$$

Then

$$\frac{\sigma_{k+1}(\hat{f}_L)}{\sigma_1(\hat{f}_L)} \leq \left(\frac{r A_r^{2/(L-1)} - 1}{k} \right)^{(L-1)/2}$$

Summary and Future Work

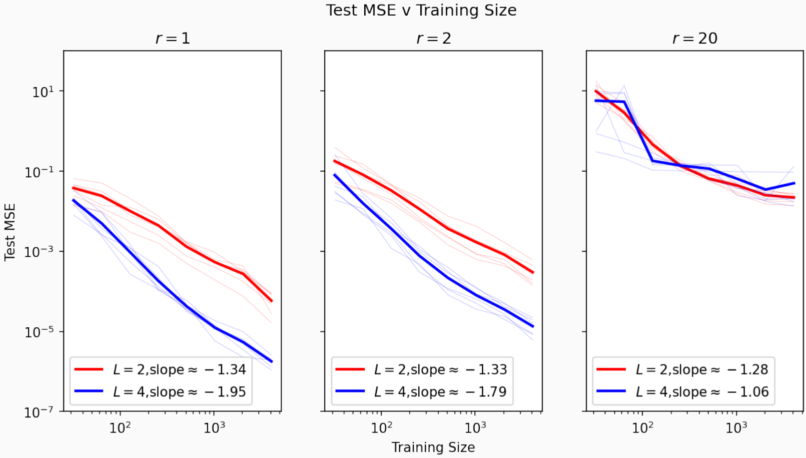
Takeaway

Linear layers = Approximate rank penalty

Takeaway

Linear layers = Approximate rank penalty = Better Generalization??

Preliminary Empirical Results



Questions?

References i

A. Daniely.
Depth separation for neural networks.
In *Conference on Learning Theory*, pages 690–696. PMLR, 2017.

Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang.
The expressive power of neural networks: A view from the width.
In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

B. Neyshabur, S. Bhojanapalli, D. Mcallester, and N. Srebro.
Exploring generalization in deep learning.
Advances in Neural Information Processing Systems, 30:5947–5956, 2017.

A. Pinkus.
Approximation theory of the mlp model in neural networks.
Acta Numerica, 8:143–195, 1999.

I. Safran, R. Eldan, and O. Shamir.
Depth separations in neural networks: what is actually being separated?
In *Conference on Learning Theory*, pages 2664–2666. PMLR, 2019.
`http://proceedings.mlr.press/v99/safran19a/safran19a.pdf`.

G. Vardi and O. Shamir.
Neural networks with small weights and depth-separation barriers.
arXiv preprint arXiv:2006.00625, 2020.