A Survey of Methods for Deriving Bounds on the Generalization of Deep Neural Networks

Thomas Walker

Contents

I Research on the Generalization of Neural Networks **4**

1 Introduction **4**

2 Setup **4**

II PAC-Bayes Bounds **6**

3 Introduction **6**

4 Optimizing PAC-Bayes Bounds via SGD **6**
- 4.1 Kullback-Liebler Divergence . . . 7
- 4.2 Probabilistic Bounds . . . 7
- 4.3 PAC Generalization Bounds . . . 8
- 4.4 Optimizing PAC Generalization Bounds . . . 9
- 4.5 Details of the Optimization Process . . . 9
- 4.6 Implementation . . . 9

5 Compression **11**
- 5.1 Establishing the Notion of Compression . . . 11
- 5.2 Compression of a Linear Classifier . . . 12
- 5.3 Compression of a Fully Connected Network . . . 14

Affiliation Imperial College London
Correspondence thomas.walker21@imperial.ac.uk
Date July 2023

6 PAC Compression Bounds **18**
6.1 Setup 18
6.2 Using Compression to Tighten PAC Bounds 19
6.3 Extension 20

III Information Theoretic Approaches 21

7 Introduction **21**

8 Setup **21**

9 Mutual Information **21**

10 Information Plane **22**

11 Information Bottleneck **22**

12 Empirical Results **23**

13 Implementation **23**
13.1 Student-Teacher Setup 23
13.2 Non-Linear Setting via KDE 23

IV Appealing to Gradients 26

14 Introduction **26**

15 Stiffness **26**
15.1 Sign and Cosine Stiffness 27
15.2 Intuition of Stiffness 27
15.3 Class Membership Stiffness 28
15.4 Super-Classes 29
15.5 Stiffness and Distance 29

16 Neural Tangent Kernels **30**
16.1 Tangent Features and Kernels 30
16.2 Application as a Complexity Measure 32

17 Geometric Complexity Measure **34**
17.1 GC and ReLU Networks 35

18 Algorithmic Stability **36**
18.1 Motivating Example 36
18.2 Coherence 37
18.3 Application to the Generalization Gap 40
18.4 Coherence of Fully Connected Neural Networks 42
18.5 Suppressing Weak Gradients 43
18.5.1 Winsorized Gradient Descent 43

V Other Approaches 45

19 Unit-Wise Capacity Measures **45**
19.1 Introduction 45
19.2 Setup 45
19.3 Generalization Bounds 46

20 Validation Paradigm **48**
20.1 Introduction 48
20.2 Setup 48
20.3 Generalization Bounds 48

VI Appendix 51

21 Bayesian Machine Learning **51**
21.1 Framework 51

22 SGD **51**

Part I
Research on the Generalization of Neural Networks

1 Introduction

The capacity of deep neural networks to generalize has been a major focus of enquiry. Over the last few years, investigations have been conducted to understand why it is that deep neural networks do not overfit training samples despite having a far greater number of parameters. Some investigations approached the problem from a theoretical perspective, whilst others took more empirical approaches. From the theoretical perspective, assumptions are made about the underlying network that facilitates the application of mathematical arguments to explain this phenomenon. Although this approach provides enlightenment of the underlying processes, often the assumptions are rather strict and lead to results that are vacuous in practice. On the other hand, empirical work tries to make sense of this phenomenon by performing controlled experiments. The results from these are largely correlational and are not grounded like the theoretical results. However, they provide insight into how this phenomenon can be exploited and amplified to construct motivated networks with enhanced performance.
In this report we aim to trek a rough chronological path through the results regarding bounds on the generalization error of deep neural networks. We look at how both theoretical and empirical investigations have led to quantitative and qualitative insights into the capacity of deep neural networks to learn from training samples in a manner that generalizes well to unseen data. Throughout bounds will be tested and explored in the simplified setting of shallow ReLU networks, with the accompanying code being available here.

2 Setup

The setting that we will consider in this report is that of fully connected neural networks performing a classification task on a set of inputs. The network will receive training data, from which it aims to learn a function to classify that data. The learning procedure will be dictated by a loss function, which is designed to quantify the quality of the learned function. Fully connected neural networks are a collection of layers of hidden units, where each hidden unit is connected to every hidden unit in the previous layer and to every hidden unit in the subsequent layer. The parameters of the neural networks are quantities that dictate how connected hidden units interact. Usually, each connection between hidden units has an associated weight and bias parameter. Inputs at a hidden unit are propagated through to the next layer by multiplying the weight parameter and adding the bias parameter to the input value along each of the branching connections. The value at a hidden unit

of the next layer is simply the sum of the manipulated inputs of the previous layer that are incident to that particular node. Typically, an activation function is then applied to introduce non-linearities into the network and increases its capacity to learn complex functions.
Gradient based learning procedures are the main tool used to train networks set up in this way. Recall that quality is captured by the loss function, one can think of the loss landscape as a space in a dimension equal to the number of parameters. The goal is then for the learning algorithm to manipulate the weights such they arrive at the global minimum of the loss landscape. Gradient-based methods observe how the loss changes with respect to perturbations in the parameters to dictate the direction it should change the weights to move down the loss landscape. Training is often carried out in batches to ease computational costs, but also introduce some noise into the learning trajectory. The noise helps the algorithm avoid local minima in the landscape and find the global minimum. This procedure is called Stochastic Gradient Descent (SGD) and has a central role in endowing neural networks with their generalization capabilities. Hence, it will be the focus of many investigations we explore in this report.
Throughout the report we will see how this setup can be formalized and then investigated rigorously to get probabilistic bounds on how the learned function performs on test data, we will see how properties of this setup can be exploited to compress the network whilst still maintaining performance, empirically we will see how metrics can be devised to give a heuristic sense on how the network is generalizing through the training procedure, and finally, we will see how the implementation of this algorithm can affect the generalizing capacity of the learned function.

Part II
PAC-Bayes Bounds

3 Introduction

Bayesian machine learning (21) is a framework for managing randomness and uncertainty in learning tasks. It allows us to deduce Probably Approximately Correct (PAC) bounds, which bound the performance of a network with high probabilities [9]. The first bounds were given by [1] in a time when deep neural networks were not a mainstream technique in the field of machine learning. In this part of the survey, we investigate how this preliminary work has been contextualised into modern deep neural networks to generate tightened bounds that prove useful in practice. For example, [2] exploits the properties of stochastic gradient descent (SGD) to establish PAC-Bayes bounds that are non-vacuous in practice.
Later on, we will understand how the over-parameterization of deep neural networks leads to compression frameworks [4] that can be used to generate tight PAC-Bayes that hold over compressed networks [10].

4 Optimizing PAC-Bayes Bounds via SGD

Here our data is the set $S_m = \{(x_i, y_i)\}_{i=1}^m$ where $x_i \in \mathcal{X} \subseteq \mathbb{R}^k$ are data points and $y_i \in \{\pm 1\}$ is the associated label. We consider a set $\mathcal{M}$ of probability measures on the data space $\mathbb{R}^k \times \{\pm 1\}$, and we assume our training data is an i.i.d sample from some $\mu \in \mathcal{M}$. The hypothesis space of classifiers is

$$\mathcal{H} = \left\{h_w = H(w, \cdot) : \mathbb{R}^k \to \{\pm 1\} : w \in \mathbb{R}^d\right\}.$$

The loss function from which we want to quantify the quality of a hypothesis is $l : \mathbb{R} \times \{\pm 1\} \to \{0, 1\}$, the 0-1 loss where $l(y, \hat{y}) = \mathbb{I}(\text{sgn}(\hat{y}) = y)$. However, for optimization purposes, we will employ the convex surrogate loss function $\tilde{l} : \mathbb{R} \times \{\pm 1\} \to \mathbb{R}_+$ where

$$\tilde{l}(y, \hat{y}) = \frac{\log\left(1 + \exp\left(-\hat{y}\, y\right)\right)}{\log(2)}.$$

We now quantify the various errors associated with these loss functions.

- $\hat{e}(h, S_m) = \frac{1}{m}\sum_{i=1}^m l(h(x_i), y_i)$, empirical classification error of hypothesis h on S_m.
- $\tilde{e}(h, S_m) = \frac{1}{m}\sum_{i=1}^m \tilde{l}(h(x_i), y_i)$, empirical error of a hypothesis h on S_m.
- $e_\mu(h) = \mathbb{E}_{S_m \sim \mu^m}\left(\hat{e}(h, S_m)\right)$, expected error for hypothesis h under μ.
- $\hat{e}(Q, S_m) = \mathbb{E}_{w \sim Q}\left(\hat{e}(h_w, S_m)\right)$, expected empirical error under randomized classifier Q on $\mathcal{H}$.
- $e(Q) = \mathbb{E}_{w \sim Q}(e_\mu(h_w))$, expected error for Q on $\mathcal{H}$.

4.1 Kullback-Liebler Divergence

KL divergence is a measure of similarity between probability measures defined on a measurable space. For the investigations conducted in this section, we will focus on the formulation of KL divergence on Euclidean space $\mathbb{R}^d$.

Definition 4.1 *Let Q, P be probability measures on Euclidean space $\mathbb{R}^d$ with densities q, p respectively. The Kullback-Liebler (KL) divergence of P from Q is*

$$\mathrm{KL}(Q, P) := \int \log\left(\frac{q(x)}{p(x)}\right) q(x)dx.$$

Remark 4.2 *Note the asymmetry in the definition.*

For the multivariate normal distributions $N_q \sim \mathcal{N}(\mu_q, \Sigma_q)$ and $N_p \sim \mathcal{N}(\mu_p, \Sigma_p)$ we have that

$$\mathrm{KL}(N_q, N_p) = \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_p^{-1}\Sigma_q\right) - k + (\mu_p - \mu_q)^\top \Sigma_p^{-1}(\mu_p - \mu_q) + \log\left(\frac{\det \Sigma_p}{\det \Sigma_1}\right)\right).$$

Similarly, for Bernoulli distributions $\mathcal{B}(q) \sim \mathrm{Bern}(q)$ and $\mathcal{B}(p) \sim \mathrm{Bern}(p)$ we have that

$$\mathrm{KL}(q, p) := \mathrm{KL}(\mathcal{B}(q), \mathcal{B}(p)) = q\log\left(\frac{q}{p}\right) + (1-q)\log\left(\frac{1-q}{1-p}\right),$$

where in the first equality we are slightly abusing notation. For $p^* \in [0, 1]$ we will be interested in bounds of the form $\mathrm{KL}(q, p^*) \leq c$ for some $q \in [0, 1]$ and $c \geq 0$. Hence, we introduce the notation

$$\mathrm{KL}^{-1}(q|c) := \sup\{p \in [0, 1] : \mathrm{KL}(q, p) \leq c\}.$$

4.2 Probabilistic Bounds

The following theorems give probabilistic bounds that will prove useful in generating probabilistic generalization bounds.

Theorem 4.3 *Let $E_1, E_2, \ldots$ be events. Then $\mathbb{P}\left(\bigcup_n E_n\right) \leq \sum_n \mathbb{P}(E_n)$.*

Theorem 4.4 *For every $p, \delta \in (0, 1)$ and $n \in \mathbb{N}$, with probability at least $1 - \delta$ over $\mathbf{x} \sim \mathcal{B}(p)^n$*

$$\mathrm{KL}\left(\frac{1}{n}\sum_{i=1}^{n} x_i, p\right) \leq \frac{\log\left(\frac{2}{\delta}\right)}{n}.$$

Theorem 4.5 *For every $\delta > 0, m \in \mathbb{N}$, distribution μ on $\mathbb{R}^k \times \{\pm 1\}$ and distribution P on $\mathcal{H}$, with probability at least $1 - \delta$ over $S_m \sim \mu^m$, for all distributions Q on $\mathcal{H}$,*

$$\mathrm{KL}(\hat{e}(Q, S_m), e(Q)) \leq \frac{\mathrm{KL}(Q, P) + \log\left(\frac{m}{\delta}\right)}{m - 1}.$$

Theorem 4.5 motivates the following PAC learning algorithm:

- Fix a probability $\delta > 0$ and a distribution P on $\mathcal{H}$,
- Collect an i.i.d sample S_m of size m,
- Compute the optimal distribution Q of $\mathcal{H}$ that minimizes

$$\mathrm{KL}^{-1}\left(\hat{e}(Q, S_m), \frac{\mathrm{KL}(Q, P) + \log\left(\frac{m}{\delta}\right)}{m-1}\right), \tag{1}$$

- Return the randomized classifier given by Q.

4.3 PAC Generalization Bounds

For a parametric family of classifiers H let $h_w = H(w, \cdot) \in H$. Here w represents the weights and biases, and h_w is the corresponding neural network. Fix $\delta \in (0, 1)$, distribution P on $\mathbb{R}^d$, and i.i.d sample $S_m \sim \mu^m$. We aim to minimize (1) with respect to Q. To do this we consider a constrained setting so that the optimization is tractable. For $w \in \mathbb{R}^d$ and $s \in \mathbb{R}^d_+$ let $\mathcal{N}_{w,s} = \mathcal{N}(w, \mathrm{diag}(s))$. Then applying the simple upper bound $\mathrm{KL}^{-1}(q, c) \leq q + \sqrt{\frac{c}{2}}$ to (1), replacing the loss with the (convex surrogate) logistic loss and restricting Q to a multivariate normal distribution of the form $\mathcal{N}_{w,s}$ we obtain the optimization problem

$$\min_{w \in \mathbb{R}^d, s \in \mathbb{R}^d_+} \tilde{e}\left(\mathcal{N}_{w,s}, S_m\right) + \sqrt{\frac{\mathrm{KL}(\mathcal{N}_{w,s}, P) + \log\left(\frac{m}{\delta}\right)}{2(m-1)}}.$$

For our case we choose prior P as $\mathcal{N}(0, \lambda I)$. To choose λ we discretize the problem, at the cost of slightly expanding the eventual generalization bound. Let λ have the form $c \exp\left(-\frac{j}{b}\right)$ for $j \in \mathbb{N}$, so that c is an upper bound and b controls precision. By ensuring Theorem 4.5 holds with probability $1 - \frac{6}{\pi^2 j^2}$ for each $j \in \mathbb{N}$ we can apply Theorem 4.3 to get results that hold for probability $1 - \delta$. Treat λ as continuous during optimization and then discretize at the point of evaluating the bound. Hence, replace j with $b \log\left(\frac{c}{\lambda}\right)$ during optimization to yields the minimization problem

$$\min_{w \in \mathbb{R}^d, s \in \mathbb{R}^d_+, \lambda \in (0, c)} \tilde{e}(\mathcal{N}_{w,s}, S_m) + \sqrt{\frac{1}{2} B_{\mathrm{RE}}(w, s, \lambda; \delta)}, \tag{2}$$

where

$$B_{\mathrm{RE}}(w, s, \lambda; \delta) = \frac{\mathrm{KL}(\mathcal{N}_{w,s}, \mathcal{N}(w_0, \lambda I)) + 2\log\left(b \log\left(\frac{c}{\lambda}\right)\right) + \log\left(\frac{\pi^2 m}{6\delta}\right)}{m-1}.$$

The goal is therefore to solve the optimization problem 2. As computing the gradient of $\tilde{e}(\mathcal{N}_{w,s}, S_m)$ is intractable in practice we instead compute the gradient of

$$\tilde{e}\left(h_{w+\xi \odot \sqrt{s}}, S_m\right)$$

where $\xi \sim \mathcal{N}_{0,I_d}$. Once good candidates for this optimization problem are found (how to do this will be outlined in Section 4.4) we then return to 1 to calculate our final error bound. With our choice of λ we get that with probability $1-\delta$, uniformly over all $w \in \mathbb{R}^d$, $s \in \mathbb{R}^d_+$ and λ (of the discrete form) the error rate of the random classifier $Q = \mathcal{N}_{w,s}$ is bounded by

$$\mathrm{KL}^{-1}(\hat{e}(Q, S_m), B_{\mathrm{RE}}(w, s, \lambda; \delta)).$$

However, computing $\hat{e}(Q, S_m)$ is intractable and so instead we obtain unbiased estimates by estimating Q. Given n i.i.d samples $w_1, \ldots, w_n$ from Q consider the Monte Carlo approximation $\hat{Q}_n = \sum_{i=1}^n \delta_{w_i}$, to get the bound

$$\hat{e}(Q, S_m) \leq \overline{\hat{e}_{n,\delta'}}(Q, S_m) := \mathrm{KL}^{-1}\left(\hat{e}\left(\hat{Q}_n, S_m\right), \frac{1}{n}\log\left(\frac{2}{\delta'}\right)\right),$$

which holds with probability $1-\delta'$. Finally, by Theorem 4.3

$$e(Q) \leq \mathrm{KL}^{-1}\left(\overline{\hat{e}_{n,\delta'}}(Q, S_m), B_{\mathrm{RE}}(w, s, \lambda; \delta)\right),$$

with probability $1-\delta-\delta'$.

4.4 Optimizing PAC Generalization Bounds

To ensure that our solution to the minimization problem 2 provides a bound that is non-vacuous we optimise the values of w, s, λ. To do this we first train the network via SGD to get a value of w. We then instantiate a stochastic neural network with a multivariate normal distribution $Q = \mathcal{N}_{w,s}$ over its weights, with $s = |w|$ being the diagonal entries of the covariance matrix. We then apply Algorithm 1 to deduce the values of w, s and λ that give us the tightest bound.

4.5 Details of the Optimization Process

When calculating the bound we need to computer $\overline{\hat{e}_{n,\delta'}}(Q, S_m) := \mathrm{KL}^{-1}\left(\hat{e}\left(\hat{Q}_n, S_m\right), \frac{1}{n}\log\left(\frac{2}{\delta'}\right)\right)$. First we note that

$$\hat{e}(\hat{Q}_n, S_m) = \sum_{i=1}^n \delta_{w_i}\left(\frac{1}{m}\sum_{j=1}^m l(h_{w_i}(x_j), y_j)\right).$$

Then as inverting KL divergence has no closed we appeal to Newton's method to find an approximation, which is outlined in Algorithm 2.

4.6 Implementation

Code available here.

Algorithm 1 Optimizing the PAC Bounds

Require:

$w_0 \in \mathbb{R}^d$, the network parameters at initialization.

$w \in \mathbb{R}^d$, the network parameters after SGD.

S_m, training examples.

$\delta \in (0,1)$, confidence parameter.

$b \in \mathbb{N}, c \in (0,1)$, precision and bound for λ

$\tau \in (0,1), T$, learning rate.

Ensure: Optimal w, s, λ

$\zeta = \text{abs}(w)$ ▷ $s(\zeta) = e^{2\eta}$

$\rho = -3$ ▷ $\lambda(\rho) = e^{2\rho}$

$B(w, s, \lambda, w') = \tilde{e}(h_{w'}, S_m) + \sqrt{\frac{1}{2} B_{\text{RE}}(w, s, \lambda)}$

for $t = 1 \to T$ **do**

Sample $\xi \sim \mathcal{N}(0, I_d)$

$w'(w, \zeta) = w + \xi \odot \sqrt{s(\zeta)}$

$$\begin{pmatrix} w \\ \zeta \\ \rho \end{pmatrix} = -\tau \begin{pmatrix} \nabla_w B(w, s(\zeta), \lambda(\rho), w'(w, \zeta)) \\ \nabla_\zeta B(w, s(\zeta), \lambda(\rho), w'(w, \zeta)) \\ \nabla_\rho B(w, s(\zeta), \lambda(\rho), w'(w, \zeta)) \end{pmatrix}$$

end for

Return $w, s(\zeta), \lambda(\rho)$

Algorithm 2 Newton's Method for Inverting KL Divergence

Require:

q, c, initial estimate p_0 and $N \in \mathbb{N}$

Ensure: p such that $p \approx \text{KL}^{-1}(q, c)$

for $n = 1 \to N$ **do**

if $p \geq 1$ **then**

Return 1

else

$$p_0 = p_0 - \frac{q \log\left(\frac{q}{c}\right) + (1-q) \log\left(\frac{1-q}{1-c}\right) - c}{\frac{1-q}{1-p} - \frac{q}{p}}$$

end if

end for

Return p_0

5 Compression

Our whole discussion are revolving around the observation that neural networks are instantiated with many more parameters than are required to overfit the data. However, what we have seen empirically is that the trained networks are not overfitting to the data and with a high probability they generalize well to unseen data. In classical statistical theory, we only require as many parameters as training samples to overfit, which suggests that some of the extra capacity of the network is redundant in expressing the learned function. In [4] compression frameworks are constructed that aims to reduce the effective number of parameters required to express the function of a trained network whilst maintaining its performance. By capitalizing on how the network responds to noise injected into the weights of the network, we can motivate algorithms for compressing the network. Consequently, we can develop and prove generalization bounds by utilizing the compression mechanisms. In this section, we motivate these methods by considering a linear classifier, and then we will explore how these methods can be deployed on fully connected ReLU neural networks.
The setup of this section is that of [4]. We will suppose that we have training data and have learned a classifier f that has a high complexity but achieves a low empirical loss. We want to ask whether a simpler classifier g exists that attains a similar empirical loss as f on the training data.

5.1 Establishing the Notion of Compression

As in [4] we suppose we have learned a classifier f that has high complexity but achieves a low empirical loss. The aim is to find a simple classifier g that attains a similar empirical loss as f on the training set. We have a training set $S_m = \{(x_i, y_i)\}_{i=1}^m$ with x_i being data points and $y_i \in \{1, \dots, k\}$. We have a multi-class classifier f that maps x to $f(x) \in \mathbb{R}^k$. The classification loss for distribution $\mathcal{D}$ is given by $\mathbb{P}_{(x,y)\sim\mathcal{D}}\left(f(x)[y] \leq \max_{i\neq y} f(x)[i]\right)$ and the expected margin loss for margin $\gamma > 0$ is given by $L_\gamma(f) = \mathbb{P}_{(x,y)\sim\mathcal{D}}\left(f(x)[y] \leq \gamma + \max_{i\neq y} f(x)[i]\right)$. Now $\mathcal{D}$ is often unknown so we work with the empirical margin loss $\hat{L}_\gamma$. Use d to denote the number of fully connected layers in our network and x^i for vector before activation at layer $i = 0, \dots, d$ as x^0 the input we will denote it x. Our network will have A^i as the weight matrix at layer i and h^i hidden units at layer i, let $h = \max_{i=1}^d h^i$. The function calculated by the network will be denoted $f_A(x)$. For layers $i \leq j$ if we denote the operator for the composition of the layers as $M^{i,j}$, the Jacobian of the input x as $J_x^{i,j}$ and $\phi(\cdot)$ as component-wise ReLU we have the following identities,

$$x^i = A^i\phi\left(x^{i-1}\right),\ x^j = M^{i,j}\left(x^i\right),\ \text{and } M^{i,j}\left(x^i\right) = J_{x^i}^{i,j}x^i.$$

It will also be useful to note that for a matrix B, $\|B\|_F$ is the Frobenius norm, $\|B\|_2$ is the spectral norm and $\frac{\|B\|_F^2}{\|B\|_2^2}$ is its stable rank.

We now formalize the notion of compression and give some preliminary results.

Definition 5.1 *Let f be a classifier and $G_{\mathcal{A}} = \{g_A : A \in \mathcal{A}\}$ be a class of classifiers. Say that f is (γ, S)-*

compressible via $G_{\mathcal{A}}$ if there exists $A \in \mathcal{A}$ such that for any $x \in S$,

$$|f(x)[y] - g_A(x)[y]| \leq \gamma$$

for all y.

Definition 5.2 *Suppose $G_{\mathcal{A},s} = \{g_{A,s} : A \in \mathcal{A}\}$ is a class of classifiers indexed by trainable parameters A and fixed string s. A classifier f is (γ, S)-compressible with respect to $G_{\mathcal{A},s}$ using helper string s if there exists $A \in \mathcal{A}$ such that for any $x \in S$,*

$$|f(x)[y] - g_{A,s}(x)[y]| \leq \gamma$$

for all y.

Theorem 5.3 *Suppose $G_{\mathcal{A},s} = \{g_{A,s} : A \in \mathcal{A}\}$ where A is a set of q parameters each of which has at most r discrete values and s is a helper string. Let S be a training set with m samples. If the trained classifier f is (γ, S)-compressible via $G_{\mathcal{A},s}$ with helper string s, then there exists $A \in \mathcal{A}$ with high probably such that*

$$L_0(g_A) \leq \hat{L}_\gamma(f) + O\left(\sqrt{\frac{q\log(r)}{m}}\right)$$

over the training set.

Remark 5.4 *The theorem only gives a bound for g_A which is the compression of f. However, there are no requirements of a hypothesis class, assumptions are only made on f and its properties on a finite training set.*

Corollary 5.5 *If the compression works for $1 - \xi$ fraction of the training sample, then with a high probability*

$$L_0(g_A) \leq \hat{L}_\gamma(f) + \xi + O\left(\sqrt{\frac{q\log r}{m}}\right).$$

5.2 Compression of a Linear Classifier

In this section, we illustrate the compression framework by compressing the decision vector of a linear classifier. Let $c \in \mathbb{R}^h$ be a decision vector that classifies an input x as $\operatorname{sgn}(c^\top x)$. Let $\mathcal{D}$ be a distribution on the inputs (x, y) where $\|x\| = 1$ and $y = \{\pm 1\}$. Define the margin, γ, of c to be such that $y\,(c^\top x) \geq \gamma$ for all (x, y) in the training set. To the vector c apply Algorithm 3.

Lemma 5.6 *Algorithm 3 (γ, c) returns a vector $\hat{c}$ such that for any fixed u, with probability $1 - \eta$, $|\hat{c}^\top - c^\top u| \leq \gamma$. The vector $\hat{c}$ has at most $O\left(\frac{\log h}{\eta\gamma^2}\right)$ non-zero entries with high probability.*

In the discrete case, we have a similar result.

Algorithm 3 (γ, c)

Require: vector c with $\|c\| \leq 1$, η.

Ensure: vector $\hat{c}$ such that for any fixed vector $\|u\| \leq 1$, with probability at least $1-\eta$, $|c^\top u - \hat{c}^\top u| \leq \gamma$.

Vector $\hat{c}$ has $O\left(\frac{\log h}{\eta\gamma^2}\right)$ non-zero entries.

for $i = 1 \to d$ **do**

Let $z_i = 1$ with probability $p_i = \frac{2c_i^2}{\eta\gamma^2}$ and 0 otherwise.

Let $\hat{c}_i = \frac{z_i c_i}{p_i}$.

end for

Return $\hat{c}$

Lemma 5.7 *Let Algorithm 3 $\left(\frac{\gamma}{2}, c\right)$ return vector $\tilde{c}$. Let,*

$$\hat{c}_i = \begin{cases} 0 & |\tilde{c}_i| \geq 2\eta\gamma\sqrt{h} \\ \textit{rounding to nearest multiple of } \frac{\gamma}{2\sqrt{h}} & \textit{Otherwise.} \end{cases}$$

Then for any fixed u with probability at least $1-\eta$, $|\hat{c}^\top u - c^\top u| \leq \gamma$.

Using Lemmas 5.6 and 5.7 along with Corollary 5.5 we have the following.

Lemma 5.8 *For any number of samples m, there is an efficient algorithm to generate a compressed vector $\hat{c}$, such that*

$$L(\hat{c}) \leq \tilde{O}\left(\left(\frac{1}{\gamma^2 m}\right)^{\frac{1}{3}}\right)$$

Remark 5.9 *The rate is not optimal as it depends on $m^{1/3}$ and not $\sqrt{m}$. To resolve this we can appeal to helper strings.*

Algorithm 4 (γ, c)

Require: vector c with $\|c\| \leq 1$, η.

Ensure: vector $\hat{c}$ such that for any fixed vector $\|u\| \leq 1$, with probability at least $1-\eta$, $|c^\top u - \hat{c}^\top u| \leq \gamma$.

Let $k = \frac{16\log\left(\frac{1}{\eta}\right)}{\gamma^2}$.

Sample the random vectors $v_1, \ldots, v_k \sim \mathcal{N}(0, I)$.

Let $z_i = \langle v_i, c\rangle$.

(In Discrete Case) Round z_i to closes multiple of $\frac{\gamma}{2\sqrt{hk}}$.

Return $\hat{c} = \frac{1}{k}\sum_{i=1}^{k} z_i v_i$

Remark 5.10 *The vectors v_i are the helper string.*

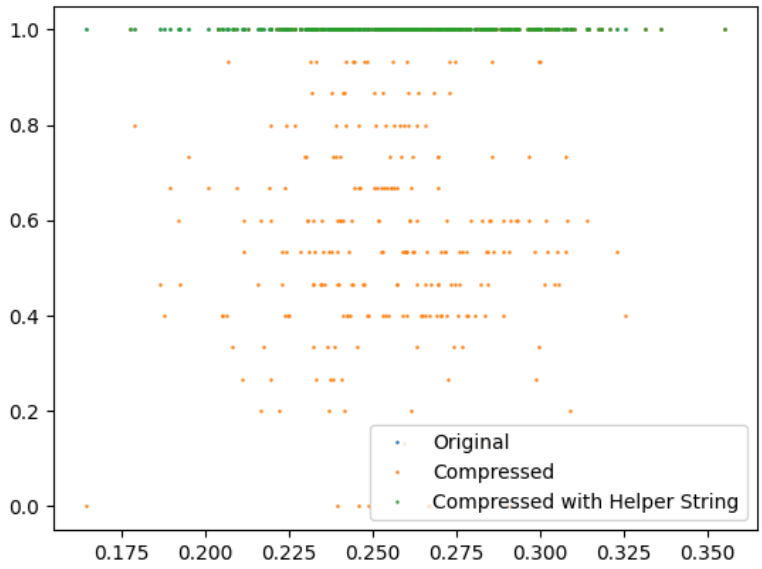

Figure 1:

Lemma 5.11 *For any fixed vector u, Algorithm 4 (γ, c) returns a vector $\hat{c}$ such that with probability at least $1-\eta$, we have $|\hat{c}^\top - c^\top u| \leq \gamma$.*

Similarly, we can conclude the following.

Lemma 5.12 *For any number of sample m, there is an efficient algorithm with helper string to generate a compressed vector $\hat{c}$, such that*

$$L(\hat{c}) \leq \tilde{O}\left(\sqrt{\frac{1}{\gamma^2 m}}\right).$$

Code that explores these examples can be found here.

5.3 Compression of a Fully Connected Network

In a similar way, we can compress the layer matrices of a fully connected network in such a way that we maintain a reasonable level of performance. A compression method is detailed in the algorithm below:

Algorithm 5 (A, ϵ, η)

Require: Layer matrix $A \in \mathbb{R}^{h_1 \times h_2}$, error parameters ϵ, η.
Ensure: Returns $\hat{A}$ such that for all vectors u, v we have that

$$\mathbb{P}\left(\left|u^\top \hat{A} v - u^\top A v\right| \geq \epsilon \|A\|_F \|u\| \|v\|\right) \leq \eta$$

Sample $k = \frac{\log(\frac{1}{\eta})}{\epsilon^2}$ random matrices $M_1, \ldots, M_k$ with i.i.d entries ± 1.
for $k' = 1 \to k$ **do**
  Let $Z_{k'} = \langle A, M_{k'} \rangle M_{k'}$
end for
Let $\hat{A} = \frac{1}{k} \sum_{k'=1}^{k} Z_{k'}$

Definition 5.13 *If M is a mapping from real-valued vectors to real-valued vectors, and $\mathcal{N}$ is a noise distribution. Then the noise sensitivity of M at x with respect to $\mathcal{N}$ is*

$$\psi_{\mathcal{N}}(M, x) = \mathbb{E}\left(\frac{\|M(x + \eta\|x\|) - M(x)\|^2}{\|M(x)\|^2}\right),$$

and let $\psi_{\mathcal{N},S}(M) = \max_{x \in S} \psi_{\mathcal{N}}(M, x)$.

Remark 5.14 *When $x \neq 0$ and the noise distribution is the Gaussian distribution $\mathcal{N}(0, I)$ then the noise sensitivity of matrix M is exactly $\frac{\|M\|_F^2}{\|Mx\|^2}$. Hence, it is at most the stable rank of M.*

We will now provide some definitions that will help formulate some results that will go toward showing that this algorithm defines a suitable compression procedure.

Definition 5.15 *The layer cushion of layer i is defined as the largest μ_i such that for any $x \in S$*

$$\mu_i \left\|A^i\right\|_F \left\|\phi\left(x^{i-1}\right)\right\| \leq \left\|A^i \phi\left(x^{i-1}\right)\right\|.$$

Remark 5.16 *Note that $\frac{1}{\mu_i^2}$ is equal to the noise sensitivity of A^i at $\phi\left(x^{i-1}\right)$ with respect to noise $\eta \sim \mathcal{N}(0, I)$.*

Definition 5.17 *For layers $i \leq j$ the inter-layer cushion $\mu_{i,j}$ is the largest number such that*

$$\mu_{i,j} \left\|J_{x^i}^{i,j}\right\|_F \left\|x^i\right\| \leq \left\|J_{x^i}^{i,j} x^i\right\|$$

for any $x \in S$. Furthermore, let $\mu_{i\to} = \min_{i \leq j \leq d} \mu_{i,j}$.

Remark 5.18 *Note that $J_{x^i}^{i,i} = I$ so that*

$$\frac{\left\|J_{x^i}^{i,i} x^i\right\|}{\left\|J_{x^i}^{i,j}\right\|_F \left\|x^i\right\|} = \frac{1}{\sqrt{h^i}}.$$

Furthermore, $\frac{1}{\mu_{i,j}^2}$ is the noise sensitivity of $J_x^{i,j}$ with respect to noise $\eta \sim \mathcal{N}(0, I)$.

Definition 5.19 *The activation contraction c is defined as the smallest number such that for any layer i*

$$\left\|\phi\left(x^i\right)\right\| \geq \frac{\left\|x^i\right\|}{c}$$

for any $x \in S$.

Definition 5.20 *Let η be the noise generated as a result of applying Algorithm 5 to some of the layers before layer i. Define the inter-layer smoothness ρ_δ to be the smallest number such that with probability $1-\delta$ we have that for layers $i < j$*

$$\left\|M^{i,j}\left(x^i+\eta\right) - J^{i,j}_{x^i}\left(x^i+\eta\right)\right\| \leq \frac{\|\eta\|\left\|x^j\right\|}{\rho_\delta\left\|x^i\right\|}$$

for any $x \in S$.

Lemma 5.21 *For any $0 < \delta, \epsilon \leq 1$ let $G = \left\{(U^i, x^i)\right\}_{i=1}^m$ be a set of matrix-vector pairs of size m where $U \in \mathbb{R}^{n\times h_1}$ and $x \in \mathbb{R}^{h_2}$, let $\hat{A} \in \mathbb{R}^{h_1\times h_2}$ be the output of Algorithm 5 $\left(A, \epsilon, \eta = \frac{\delta}{mn}\right)$. With probability at least $1-\delta$ we have for any $(U,x) \in G$ that $\left\|U(\hat{A}-A)x\right\| \leq \epsilon\|A\|_F\|U\|_F\|x\|$.*

Remark 5.22 *We can apply this to a neural network. For any layer let x be the input, A be the layer matrix and U the Jacobian of the network output with respect to the layer input. Then the network output before compression is given by UAx and after compression the output is given by $U\hat{A}x$.*

We can use Lemma 5.21 to prove the following.

Lemma 5.23 *For any fully connected network f_A with $\rho_\delta \geq 3d$, any probability $0 < \delta \leq 1$ and any $0 < \epsilon \leq 1$, Algorithm 5 can generate weights $\tilde{A}$ for a network with*

$$\frac{72c^2d^2\log\left(\frac{mdh}{\delta}\right)}{\epsilon^2}\cdot\sum_{i=1}^{d}\frac{1}{\mu_i^2\mu_{i\rightarrow}^2}$$

total parameters such that with probability $1-\frac{\delta}{2}$ over the generated weights $\tilde{A}$, for any $x \in S$

$$\left\|f_A(x) - f_{\tilde{A}}(x)\right\| \leq \epsilon\|f_A(x)\|,$$

where $\mu_i, \mu_{i\rightarrow}, c$ and ρ_δ are the layer cushion, inter-layer cushion, activation contraction and inter-layer smoother for the network.

Using Lemma 5.23 we get the next result.

Lemma 5.24 *For any fully connected network f_A with $\rho_\delta \geq 3d$, any probability $0 < \delta \leq 1$ and any margin $\gamma > 0$, f_A can be compressed (with respect to a random string) to another fully connected network $f_{\tilde{A}}$ such that for $x \in S$, $\hat{L}_0(f_{\hat{A}}) \leq \hat{L}_\gamma(f_A)$ and the number of parameters in $f_{\tilde{A}}$ is at most*

$$\tilde{O}\left(\frac{c^2d^2\max_{x\in S}\|f_A(x)\|_2^2}{\gamma^2}\sum_{i=1}^{d}\frac{1}{\mu_i^2\mu_{i\rightarrow}^2}\right).$$

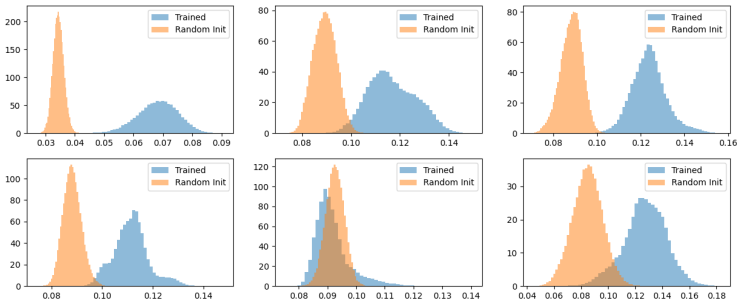

Figure 2:

Lemma 5.25 *Let f_A be a d-layer network with weights $A = \{A^1, \dots, A^d\}$. Then for any input x, weights A and $\hat{A}$, if for any layer i, $\left\|A^i - \hat{A}^i\right\| \leq \frac{1}{d}\|A^i\|$, then*

$$\|f_A(x) - f_{\hat{A}}(x)\| \leq e\|x\| \left(\prod_{i=1}^{d} \|A^i\|_2\right) \sum_{i=1}^{d} \frac{\left\|A^i - \hat{A}^i\right\|_2}{\|A^i\|_2}$$

Now using Lemmas 5.23,5.24 and 5.25 we are able to show that Algorithm 5 provides a generalization bound for suitable fully connected neural networks.

Theorem 5.26 *For any fully connected network f_A with $\rho_\delta \geq 3d$, any probability $0 < \delta \leq 1$ and any margin γ. Algorithm 5 generates weights $\tilde{A}$ such that with probability $1 - \delta$ over the training set,*

$$L_0(f_{\tilde{A}}) \leq \hat{L}_\gamma(f_A) + \tilde{O}\left(\sqrt{\frac{c^2 d^2 \max_{x \in S} \|f_A(x)\|_2^2 \sum_{i=1}^{d} \frac{1}{\mu_i^2 \mu_{i\rightarrow}^2}}{\gamma^2 m}}\right).$$

Code that explores this example can be found here.

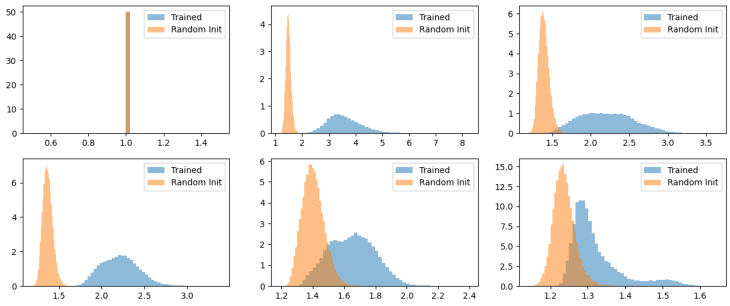

Figure 3:

6 PAC Compression Bounds

In this section, we explore the work of [10], which uses compression ideas to tighten PAC-Bayes bounds on generalization error. The work evaluates generalization bounds by measuring the effective compressed size of networks and then substituting this into the bounds. For example, we have just seen that neural networks tend to be robust to perturbations of their weights, which lead to compression algorithms. Accounting for this we can reduce effective complexity. Intuitively, if a model tends to overfit then there is a limit as to how much it can be compressed, thus affecting the tightness of the bound.

6.1 Setup

Consider learning a classifier using examples (x, y) where $x \in \mathcal{X}$ is a feature and $y \in \mathcal{Y}$ is a label. We assume that $(X_i, Y_i) \overset{\text{i.i.d}}{\sim} \mathcal{D}$ for some distribution $\mathcal{D}$. The aim is to choose a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the risk,

$$L(h) = \mathbb{E}_{(X,Y)\sim\mathcal{D}}(L(h(X), Y)).$$

Generally, $\mathcal{D}$ is unknown and so instead using an i.i.d sample $S_m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ one can minimize the empirical risk

$$\hat{L}(h) = \frac{1}{m} \sum_{(x,y)\in S} L(h(x), y).$$

However, suppose that $\hat{h}$ minimizes the empirical risk, there is the possibility that it overfits the training sample, $\hat{L}(\hat{h}) \ll L(\hat{h})$. The generalization error is given by $L(\hat{h}) - \hat{L}(\hat{h})$ and measures the degree to which our hypothesis overfits.

Theorem 6.1 *Let L be a $\{0,1\}$-valued loss function. Let π be a probability measure on the hypothesis class, and let $\alpha > 1, \epsilon > 0$. Then, with probability at least $1-\epsilon$ over the distribution of the sample:*

$$L(\rho) = \mathbb{E}_{h\sim\rho}(L(h)) \leq \inf_{\lambda>1} \Phi^{-1}_{\lambda/n}\left(\hat{L}(\rho) + \frac{\alpha}{\lambda}\left(\mathrm{KL}(\rho,\pi) - \log(\epsilon) + 2\log\left(\frac{\log\left(\alpha^2\lambda\right)}{\log(\alpha)}\right)\right)\right)$$

where

$$\Phi^{-1}(x) = \frac{1-e^{-\gamma x}}{1-e^{-\gamma}}.$$

Remark 6.2 *We refer to π as the prior and ρ as the posterior. The challenge is then to find a π for which $\mathrm{KL}(\rho,\pi)$ is small and computable. The choice of π can be motivated by ideas of compressibility.*

6.2 Using Compression to Tighten PAC Bounds

The idea now is to choose the prior π such that a greater probability mass is associated with models with short code length. Consider a non-random classifier by taking the posterior ρ to be a point mass at $\hat{h}$, that is the output of the training plus compression procedure.

Theorem 6.3 *Let $|h|_c$ denote the number of bits required to represent hypothesis h using some pre-specified coding c. Let ρ denote the point mass at the compressed model $\hat{h}$. Let m denote any probability measure on the positive integers. There exists a prior π_c such that*

$$\mathrm{KL}(\rho,\pi_c) \leq \left|\hat{h}\right|_c \log(2) - \log\left(m\left(\left|\hat{h}\right|_c\right)\right).$$

We now formalise compression schemes to enable us to make use of Theorem 6.3. Denote a compression procedure by a triple (S,C,Q) where

- $S = \{s_1,\ldots,s_k\} \subseteq \{1,\ldots,p\}$ is the location of the non-zero weights,
- $C = \{c_1,\ldots,c_r\} \subseteq \mathbb{R}$, is a codebook, and
- $Q = (q_1,\ldots,q_k)$ for $q_i \in \{1,\ldots,r\}$ are the quantized values.

Define the corresponding weights $w(S,Q,C) \in \mathbb{R}^p$ as,

$$w_i(S,Q,C) = \begin{cases} = c_{q_j} & i = s_j \\ 0 & \text{otherwise.} \end{cases}$$

Let $\rho \sim \mathcal{N}\left(w,\sigma^2 J\right)$, with J a diagonal matrix,

Theorem 6.4 *Let (S,C,Q) be the output of a compression scheme, and let $\rho_{S,C,Q}$ be the stochastic estimator given by the weights decoded from the triplet and variance σ^2. Let c denote an arbitrarily fixed coding scheme*

and let m denote an arbitrary distribution on the positive integers. Then for any $\tau > 0$, there is a PAC-Bayes prior π such that

$$\begin{aligned}\mathrm{KL}(\rho_{S,C,Q}, \pi) \leq & (k\lceil \log(r) \rceil + |S|_c + |C|_c)\log(2) - \log(m)(k\lceil \log(r) \rceil + |S|_c + |C|_c) \\ & + \sum_{i=1}^{k} \mathrm{KL}\left(\mathcal{N}\left(c_{q_i}, \sigma^2\right), \sum_{j=1}^{r} \mathcal{N}\left(c_j, \tau^2\right) \right).\end{aligned}$$

Choosing the prior alluded to by Theorem 6.4 and utilizing Theorem 6.1 one can obtain a PAC generalization bounds that exploits notions of compressibility.

6.3 Extension

[My own work] Using the framework for defining compression mechanisms, we can formalize Algorithm 3 from the previous section. Recall, that the algorithm compresses a vector $c \in \mathbb{R}^d$ to a vector $\hat{c}$. Under the framework defined above we want to define S, Q and C such that $\hat{c}_i = w_i(S, Q, C)$. As $\hat{c}$ is a random variable there must be some randomness to either S, Q or C. For $i \in \{1, \ldots, d\}$ let $i \in S$ with probability $\frac{2c_i^2}{\eta\gamma^2}$. Now let $C_i = \frac{c_i}{p_i}$ and $q_i = s_i$. Recalling, that

$$w_i(S, Q, C) = \begin{cases} c_{q_j} & i = s_j \\ 0 & \text{otherwise,} \end{cases}$$

it follows that $\hat{c}_i = 0$ when $i \notin S$ which happens with probability $1 - \frac{2c_i^2}{\eta\gamma^2}$. On the other hand, $\hat{c}_i \neq 0$ only when $i \in S$ which happens with probability $\frac{2c_i^2}{\eta\gamma^2}$, in which case $\hat{c}_i = C_i = \frac{c_i}{p_i}$. Note that in this compression framework, there is no quantization and thus the codebook will have a length equal to the number of non-zero entries in $\hat{c}$. In this using a Huffman encoding will provide no benefit, as entries occur singly. However, we can appeal to the bound of Theorem 6.2 to get a bound on the KL divergence that can then be inserted into Theorem 6.1 to get a bound on the generalization error.

Part III
Information Theoretic Approaches

7 Introduction

It has been shown that layered neural networks form a Markov chain, which suggests studying them from the perspective of mutual information. That is, we let each layer be a single random variable and consider its mutual information with the input X and the output Y. The properties of neural network training are explored using this approach in [3].

8 Setup

In [3] learning is formulated as finding a good representation $T(X)$ of the input patterns $x \in X$ that generates good predictions for label $y \in Y$. Training is conducted through stochastic gradient descent (SGD), where are each step we aim to minimize the empirical error over the weights of the network. As this minimization occurs layer-by-layer we can treat a whole layer as a single representation T which is characterized by the encoder $P(T|X)$ and the decoder $P(Y|T)$ distributions.

9 Mutual Information

Definition 9.1 *For two random variables X and Y, with joint distribution $p(x, y)$, their Mutual Information is defined as,*

$$\begin{aligned} I(X;Y) = \mathrm{KL}(p(x,y), p(x)p(y)) &= \sum_{x\in X, y\in Y} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) \\ &= H(X) - H(X|Y), \end{aligned}$$

where $H(X)$ and $H(X|Y)$ are the entropy and conditional entropy of X and Y.

Mutual information quantifies the number of relevant bits that the input variable X contains about the label Y on average.

Theorem 9.2 *For any invertible functions ϕ and ψ,*

$$I(X;Y) = I(\psi(X), \phi(Y)).$$

Theorem 9.3 *For any three variables that form a Markov chain $X \to Y \to Z$,*

$$I(X;Y) \geq I(X,Z).$$

10 Information Plane

Given $P(X;Y)$, any representation T corresponds to a unique point in the information plane with coordinates $(I(X;T), I(T;Y))$. Consider a K-layered deep neural network, with T_i denoting the representation of the i^{th} layer then there is a unique information path which satisfies Theorem 9.3,

$$\begin{aligned} I(X;Y) \geq I(T_1;Y) \geq \cdots \geq I(T_k;Y) \geq I\left(\hat{Y};Y\right) \\ H(X) \geq I(X;T_1) \geq \cdots \geq I(X;T_k) \geq I\left(X;\hat{Y}\right). \end{aligned}$$

Due to Theorem 9.2 it is possible that many different deep neural networks correspond to an information path.

11 Information Bottleneck

Definition 11.1 *Let X and Y be two random variables. A sufficient statistic $S(X)$ is map or partition of X that captures all the information that X has on Y,*

$$I(S(X);Y) = I(X,Y).$$

A minimal sufficient statistic, $T(X)$, induces the coarsest such partition on X. Consequently, one can form the Markov Chain

$$Y \to X \to S(X) \to T(X).$$

Using Theorem 9.3 finding T can be formulated as the optimization problem,

$$T(X) = \arg \min_{S(X);I(S(X);Y)=I(X;Y)} I(S(X);X).$$

The Information Bottleneck trade-off enables a framework for finding approximate minimal sufficient statistics. Let $t \in T$ be the compressed representation of $x \in X$, so that x is represented as $p(t|x)$. The Information Bottleneck trade-off is captured in the optimization problem

$$\min_{p(t|x),p(y|t),p(t)} (I(X;T) - \beta I(T;Y)).$$

Where β determines the level of relevant information captured by T. The solution to this problem is given by

$$\begin{cases} p(t|x) = \frac{p(t)}{Z(x;\beta)} \exp\left(-\beta \mathrm{KL}(p(y|x), p(y|t))\right) \\ p(t) = \sum_x p(t|x)p(x), \\ p(y|t) = \sum_x p(y|x)p(x|t), \end{cases} \tag{3}$$

where $Z(x;\beta)$ is the normalized function.

12 Empirical Results

13 Implementation

We now appeal to work done by [7], which develops on the work done by [3].

13.1 Student-Teacher Setup

In this scenario, there is a linear teacher network that generates training examples for a deep linear student network to learn. For the teacher network, we have an input size of N_i and an output size of 1. The input is a multi-variate normal, $X \sim \mathcal{N}(0, \frac{1}{N_i} I_{N_i})$, and the weights of the network, W_o, are sampled independently from $\mathcal{N}(0, \sigma_2^2)$. The output for a given input is given by $Y = W_o X + \epsilon_o$ where $\epsilon_o \sim \mathcal{N}(0, \sigma_o^2)$. We take P samples from this teacher network to train a student network using gradient descent to minimize the mean squared error. The student network consists of an input layer, hidden layers and a single output neuron. The activation function on the hidden layer neurons is just the identity function. This setup can be represented as $\hat{Y} = W_{D+1} \dots W_1 X$ when the network has depth D. From this, we see that the activity of the i^{th} hidden layer is given by $T = \bar{W} X = W_i \dots W_1 X$. The true generalization error at training step t is given by

$$E_g(t) = \|W_o - W_{\text{tot}}(t)\|_F^2 + \sigma_o^2.$$

To calculate the mutual information we need to add some noise otherwise, it would be infinite. Hence, we suppose that $T = \bar{W} X + \epsilon_{MI}$ for $\epsilon_{MI} \sim \mathcal{N}(0, \sigma_{MI}^2)$. With these assumptions, it follows that

$$I(T; X) = \log |\bar{W}\bar{W}^\top + \sigma_{MI}^2 I_{N_h}| - \log |\sigma_{MI}^2 I_{N_h}|,$$

where $|\cdot|$ denotes the determinant of a matrix and N_h is the number of hidden units in the layer. Similarly, the mutual information with the output Y can be calculated as follows

$$\begin{aligned}
H(Y) &= \frac{N_o}{2} \log(2\pi e) + \frac{1}{2} \log |W_o W_o^\top + \sigma_o^2 I_{N_o}| \\
H(T) &= \frac{N_h}{2} \log(2\pi e) + \frac{1}{2} \log |\bar{W}\bar{W}^\top + \sigma_{MI}^2 I_{N_h}| \\
H(Y; T) &= \frac{N_o + N_h}{2} \log(2\pi e) + \frac{1}{2} \log \begin{vmatrix} \bar{W}\bar{W}^\top + \sigma_{MI}^2 I_{N_h} & \bar{W} W_o^\top \\ W_o \bar{W}^\top & W_o W_o^\top + \sigma_o^2 I_{N_o} \end{vmatrix} \\
I(Y; T) &= H(Y) + H(T) - H(Y; T),
\end{aligned}$$

where N_o is the size of the input. For our implementation, we will focus on the case where we have only a single hidden layer so that $\bar{W} = W_1$, refer to Figure 4 for the results.

13.2 Non-Linear Setting via KDE

To investigate the mutual information in the case of non-linear neural networks we appeal to Kernel Density Estimation. To apply KDE we assume that the hidden activity is distributed as a mixture

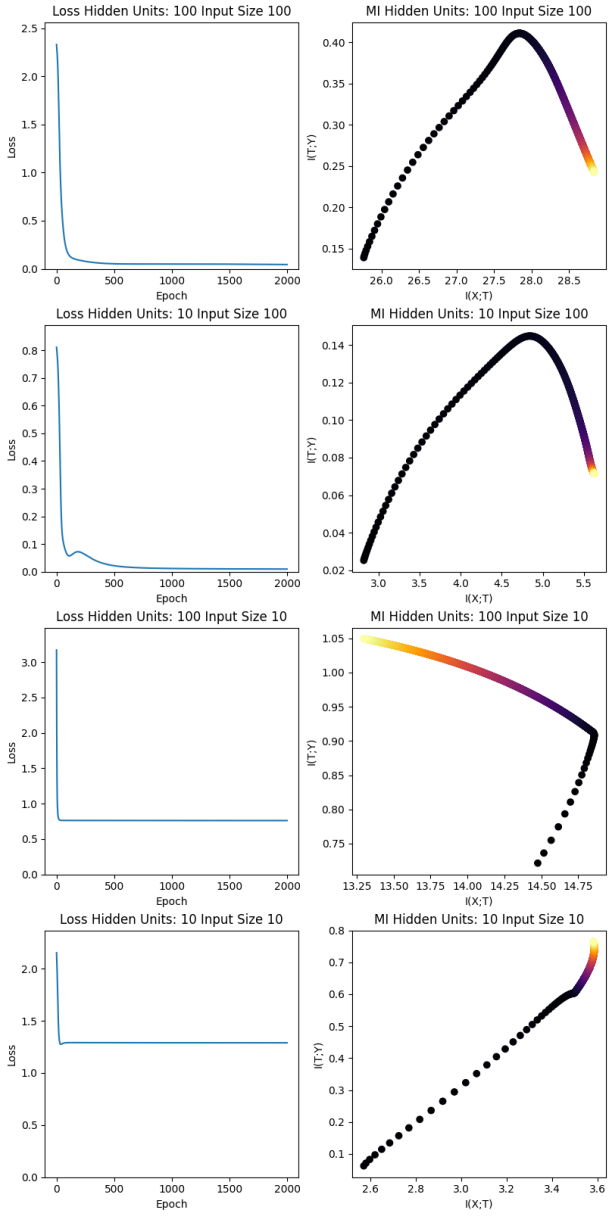

Figure 4: Explores the Mutual Information in the teacher-student scenario with different architectures and training data.

of Gaussians. As the layer activity is a deterministic function of the input we need to add some noise to get finite mutual information. Hence, we let $T = h + \epsilon$ where h is the activity of the hidden layer and $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$. Under these assumptions, it follows that

$$I(T;X) \leq -\frac{1}{P}\sum_i \log\left(\frac{1}{P}\sum_j \exp\left(-\frac{1}{2}\frac{\|h_i - h_j\|_2^2}{\sigma^2}\right)\right)$$

$$I(T;Y) \leq -\frac{1}{P}\sum_i \log\left(\frac{1}{P}\sum_j \exp\left(-\frac{1}{2}\frac{\|h_i - h_j\|_2^2}{\sigma^2}\right)\right) - \sum_{l=1}^{L} p_l \left(-\frac{1}{P_l}\sum_{Y_i=l} \log\left(\frac{1}{P_l}\sum_{Y_j=l} \exp\left(-\frac{1}{2}\frac{\|h_i - h_j\|_2^2}{\sigma^2}\right)\right)\right),$$

where

- P is the number of training samples,
- h_i is the hidden activity in response to sample i,
- L is the number of output labels,
- P_l is the number of samples with label l,
- $p_l = \frac{P_l}{P}$, and
- $Y_i = l$ is the sum over examples with output l.

See the results of this implementation in Figure 5. We train a network with layers $784 \to 1024 \to 20 \to 20 \to 20 \to 10$ on 1000 images from the MNIST dataset. We observe the activations of the third hidden layer and use these to estimate the mutual information between the inputs and outputs. The orange line corresponds to the entropy of a uniform distribution across the 1000 samples ($\log_2(1000)$).

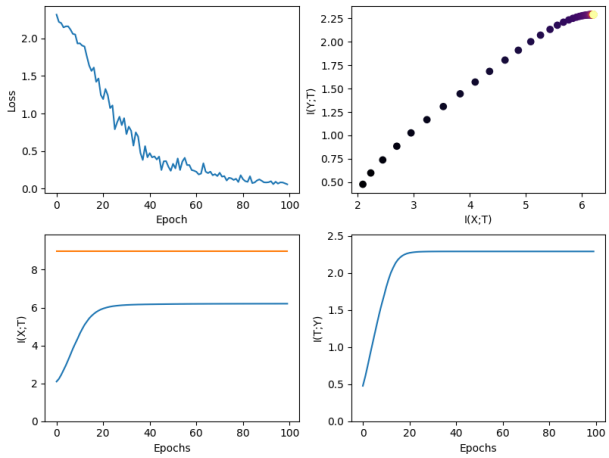

Figure 5:

Part IV
Appealing to Gradients

14 Introduction

Gradients form the foundation of implementing deep neural network architectures. It is through gradient descent methods that one is able to train these large networks to perform well on data sets. These methods work by understanding the gradient of a loss function at training examples with respect to the training examples. Therefore, any of the learned features of the network are discovered within these gradients. The properties of the trained network are a product of the evolution of these gradients. Designing tools to investigate how the learning algorithm manipulates these gradients can provide a lot of insight into the features learned by the network, the processes underlying the training process, the network's capacity to generalize, and areas of exploitation to refine the training process.

15 Stiffness

We first look at an idea introduced in [8]. Where a notion of stiffness is used to

1. Explain network generalization,
2. Uncover semantically meaningful groups of data points,
3. Explore the effects of learning rates on the function learned by the network, and

4. Defining and measuring so-called dynamical critical length, ξ.

It will turn out that the definition of stiffness will

1. relate directly to network generalization,
2. will be sensitive to the semantically meaningful content of inputs, and
3. capture information about the loss landscape.

15.1 Sign and Cosine Stiffness

Consider a functional approximation f, parameterized by a trainable parameter W. Suppose we are conducting classification tasks, with data points X and true labels y. Let $\mathcal{L}(f_W(X), y))$ be the loss function, and let the gradient of this loss function with respect to the parameters be

$$\bar{g} = \nabla_W \mathcal{L}(f_W(X), y).$$

Gradient descent uses this quantity to update the weights of the functional approximation. We will now give two definitions of stiffness, which both capture the underlying principle of the phenomena we are trying to describe with stiffness but do it in slightly different ways.

Definition 15.1 *For two data points (X_1, y_1) and (X_2, y_2) define the sign stiffness to be*

$$S_{\text{sign}}\left((X_1, y_1), (X_2, y_2); f\right) = \mathbb{E}\left(\text{sign}\left(\bar{g}_1 \cdot \bar{g}_2\right)\right).$$

Definition 15.2 *For two data points (X_1, y_1) and (X_2, y_2) define the cosine stiffness to be*

$$S_{\cos}\left((X_1, y_1), (X_2, y_2); f\right) = \mathbb{E}\left(\cos\left(\bar{g}_1 \cdot \bar{g}_2\right)\right),$$

where

$$\cos(\bar{g}_1 \cdot \bar{g}_2) = \frac{\bar{g}_1 \cdot \bar{g}_2}{|\bar{g}_1||\bar{g}_2|}.$$

Remark 15.3 *Note the stiffness is dependent on the underlying distributions of the dataset from which X_1 and X_2 are drawn.*

15.2 Intuition of Stiffness

To understand what these definitions mean intuitively we consider two data points and their ground truth labels (X_1, y_1) and (X_2, y_2). We can compute the gradient of the loss function with respect to the parameters at these data points as

$$\bar{g}_1 \nabla_W \mathcal{L}(f_W(X_1), y_1) \text{ and } \bar{g}_2 = \nabla_W \mathcal{L}(f_W(X_2), y_2).$$

Now we consider

$$\Delta\mathcal{L}_1 = -\epsilon\nabla_\epsilon\mathcal{L}(f_{W-\epsilon\bar{g}_1}(X_1), y_1) = -\epsilon\bar{g}_1 \cdot \bar{g}_1 + O(\epsilon^2).$$

By construction as $\epsilon \to 0$ we have that $\Delta\mathcal{L}_1 < 0$. However, stiffness concerns itself with the behaviour of

$$\Delta\mathcal{L}_2 = -\epsilon\nabla_\epsilon\mathcal{L}(f_{W-\epsilon\bar{g}_1}(X_2), y_2) = -\epsilon\bar{g}_1 \cdot \bar{g}_2 + O(\epsilon^2),$$

as ϵ (the same as that used in $\Delta\mathcal{L}_1$) tends to 0. A positive stiffness means that in this limit $\Delta\mathcal{L}_2 < 0$, and so the parameter updates caused by SGD for these two data points support each other. Hence, the degree to which they support each other is directly related to the quantity $\bar{g}_1 \cdot \bar{g}_2$.

15.3 Class Membership Stiffness

It turns out that sign stiffness will be more suitable for identifying stiffness between classes, whereas cosine stiffness will be more useful for identifying within-class stiffness. Stiffness can be evaluated in different ways. Firstly, there is the train-train method where the two points are chosen from the train set. In the train-val evaluation, one point is taken from the train set and the other from the validation set. Then, in the val-val evaluation, both points are taken from the validation set. Clearly, generalization is directly related to train-val stiffness. Suppose that our set of data points X contains semantic classes $c_1, c_2, \ldots, c_n$, then naturally we would like to consider the stiffness of data points between the classes.

Definition 15.4 *Define the class stiffness, C, to be a matrix with entries*

$$[C]_{ij} = \mathbb{E}_{X_1 \in c_i, X_2 \in c_j, X_1 \neq X_2}(S((X_1, y_1), (X_2, y_2))).$$

We can interpret the entries of this matrix as follows,

- The on-diagonal elements correspond to the suitability of the gradient update to the members of the class itself. Hence, these entries give us an idea of within-class generalization.
- The off-diagonal elements express the amount of improvement transferred from one class to another.

Using these interpretations we make the following definitions.

Definition 15.5 *For a set of data points X with n semantic classes we summarize the between-class stiffness by*

$$S_{\text{between classes}} = \frac{1}{n(n-1)} \sum_i \sum_{i \neq j} [C]_{ij}.$$

Definition 15.6 *For a set of data points X with n semantic classes we summarize the within-class stiffness by*

$$S_{\text{within classes}} = \frac{1}{n} \sum_i [C]_{ii}.$$

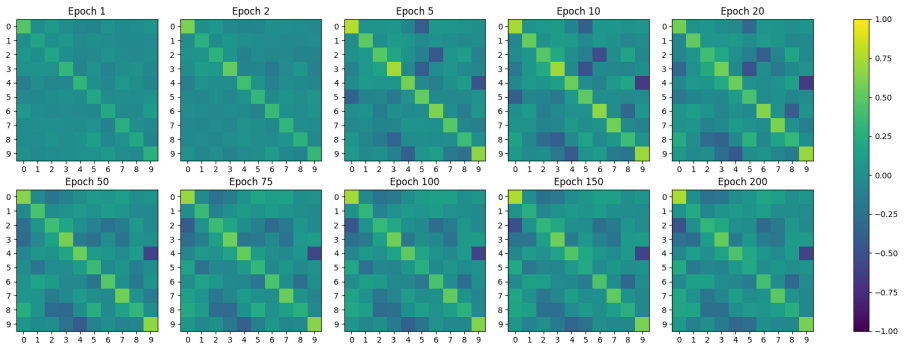

Figure 6: Evolution of the stiffness between the classes of MNIST data sets during training.

Intuitively, when within-class stiffness drops below 1, the generality of the features improved does not extend to even the class itself, suggesting that overfitting may be occurring.

15.4 Super-Classes

From our dataset, using the context available we can categorise classes into groups that share more meta semantic properties. We call these categories super-classes, and similarly, we call categories of super-classes super-super-classes. Empirically, it has been observed that stiffness between these clustered groups is higher than expected, suggesting that stiffness can be used as a measure of related semantic content between classes.

15.5 Stiffness and Distance

We can also investigate the relationship between stiffness and absolute distance between the points in the input space. If we normalize our data to the unit sphere we can use the metric

$$\text{distance}\left(\bar{X}_1, \bar{X}_2\right) = 1 - \frac{\bar{X}_1 \cdot \bar{X}_2}{|\bar{X}_1||\bar{X}_2|}.$$

We note that at a distance of zero, the stiffness is one. We define a threshold distance ξ to be the point at which on average the stiffness goes to zero. Observing ξ as a function of distance and learning rate enables one to estimate the size of the stiff regions of a neural network.

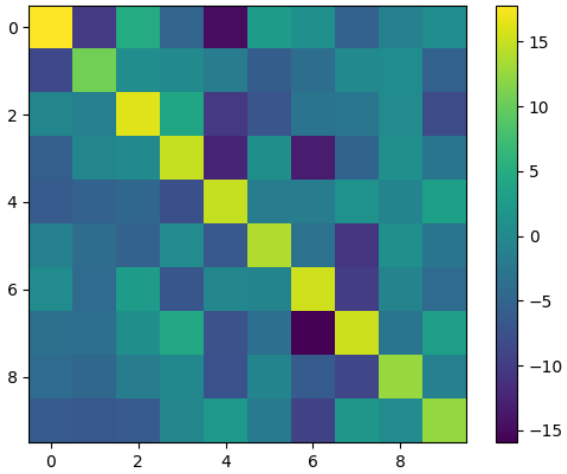

Figure 7: The Last layer activity of a neural network trained on MNIST.

16 Neural Tangent Kernels

The idea of tangent kernels is to understand what function the neural networks are learning. As we've emphasised before, neural networks improve performance by altering their parameters in response to how a loss function evaluated at different data points changes with respect to those parameters. Therefore, to discern the features captured by the network we ought to look at how the gradients of the functions represented by the network interact at different points. Intuitively, at data points with similar features, these functions should have gradients that interact strongly in the parameter space as these are reinforced by the learning algorithm. In [11] this intuition is formalized and then also applied to give of neural network complexity.

16.1 Tangent Features and Kernels

We let $\mathcal{F}$ be a class of scalar functions parameterised by $\mathbf{w} \in \mathbb{R}^P$, $f_{\mathbf{w}} : \mathcal{X} \to \mathbb{R}$.

Definition 16.1 *The tangent features of a scalar function $f_{\mathbf{w}} : \mathcal{X} \to \mathbb{R}$ is the gradient with the respect to the parameters,*

$$\Phi_{\mathbf{w}}(\mathbf{x}) = \nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x}) \in \mathbb{R}^P.$$

With the corresponding tangent kernel given by

$$k_{\mathbf{w}}(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi_{\mathbf{w}}(\mathbf{x}), \Phi_{\mathbf{w}}(\tilde{\mathbf{x}}) \rangle$$

Using these definitions we can quantify the changes in the output of the function as a result of perturbations in the weights as

$$\delta f_{\mathbf{w}}(\mathbf{x}) = \langle \delta \mathbf{w}, \Phi_{\mathbf{w}}(\mathbf{x}) \rangle + O\left(\|\delta \mathbf{w}\|^2 \right).$$

To characterise the prominent directions in parameter space we look at the eigenvalue decomposition of the covariance matrix

$$g_{\mathbf{w}} = \mathbb{E}_{\mathbf{x}\sim\rho}\left(\Phi_{\mathbf{w}}(\mathbf{x}) \Phi_{\mathbf{w}}(\mathbf{x})^\top \right)$$

where ρ is the distribution of the input. That is, we consider

$$g_{\mathbf{w}} = \sum_{j=1}^{P} \lambda_{\mathbf{w}j} \mathbf{v}_{\mathbf{w}j} \mathbf{v}_{\mathbf{w}j}^\top.$$

Given inputs $\mathbf{x}_i$, and the corresponding vector outputs $f_{\mathbf{w}}(\mathbf{x}_i)$ for $i = 1, \ldots, n$, the gradient descent update on the weights is given by

$$\delta \mathbf{w}_{\mathrm{GD}} = -\eta \nabla_{\mathbf{w}} L$$

for some loss function L. The resulting function updates can be linearly approximated by

$$\delta f_{\mathrm{GD}}(\mathbf{x}_i) = \langle \delta \mathbf{w}_{\mathrm{GD}}, \Phi_{\mathbf{w}}(\mathbf{x}_i) \rangle,$$

which can be decomposed in the eigenbasis of the tangent kernel,

$$u_{\mathbf{w}j}(\mathbf{x}) = \frac{1}{\sqrt{\lambda_{\mathbf{w}j}}} \langle \mathbf{v}_{\mathbf{w}j}, \Phi_{\mathbf{w}}(\mathbf{x}) \rangle.$$

Lemma 16.2 *The function updates decompose as* $\delta f_{GD} = \sum_{j=1}^{P} \delta f_j u_{\mathbf{w}j}(\mathbf{x})$ *with*

$$\delta f_j = -\eta \lambda_{\mathbf{w}j} (\mathbf{u}_{\mathbf{w}j})^\top (\nabla_{f_{\mathbf{w}}} L),$$

where $\mathbf{u}_{\mathbf{w}j} = (u_{\mathbf{w}j}(\mathbf{x}_1) \ldots u_{\mathbf{w}j}(\mathbf{x}_n))^\top \in \mathbb{R}^n$ *and* $\nabla_{f_{\mathbf{w}}}$ *is the gradient with respect to sample outputs.*

This illustrates how the eigenvalues of the tangent kernel act as mode-specific re-scaling of the learning rate. Introducing a local bias to prioritize learning functions within the top eigenspaces of the kernel. Hence, understanding the properties of the tangent kernel should enable us to understand what the network has learned. An implementation of the tangent kernel to do just that can be found here.

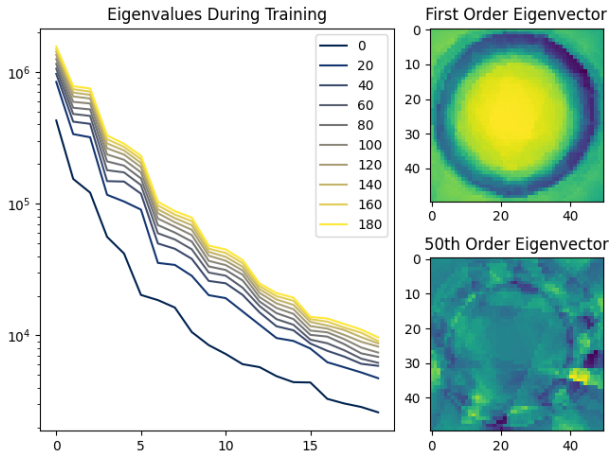

Figure 8: **Left:** Eigenvalues of tangent kernel during training. **Right:** Eigenvectors of the tangent kernel at the end of training.

16.2 Application as a Complexity Measure

We are still considering scalar functions $f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ that are linearly parameterised by $\mathbf{w} \in \mathbb{R}^p$. When given n inputs, the tangent features $\Phi(\mathbf{x}_i) \in \mathbb{R}^P$ yield a feature matrix $\Phi \in \mathbb{R}^{n \times P}$. Recall, that the empirical Rademacher complexity for a sample set S and a class of functions $\mathcal{F}$ is given by

$$\hat{\mathfrak{R}}_S(\mathcal{F}) = \frac{1}{m} \mathbb{E}_{\xi \in \{\pm 1\}^m} \left(\sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \xi_i f(\mathbf{x}_i) \right).$$

Therefore, by controlling the capacity of our networks can yield tighter generalization bounds as the Rademacher complexity depends on the size of the class $\mathcal{F}$. Common methods of restricting capacity appeal to bounding the norm of the weight vector. However, here we consider the implicit bias induced by the training algorithm we have established using the tangent kernel.

Definition 16.3 *Let $A \in \mathbb{R}^{P \times P}$ be an invertible matrix. The vector norm of $\mathbf{w} \in \mathbb{R}^P$ induced by A is given by $\|\mathbf{w}\|_A := \sqrt{\mathbf{w}^\top g_A \mathbf{w}}$ where $g_A = AA^\top$.*

Proposition 16.4 *For the restricted class of functions $\mathcal{F}^A_{M_A} = \{f_{\mathbf{w}} : \mathbf{x} \mapsto \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle : \|\mathbf{w}\|_A \leq M_A\}$ and sample set S,*

$$\hat{\mathfrak{R}}_S \left(\mathcal{F}^A_{M_A} \right) \leq \frac{M_A}{n} \left\| A^{-1} \Phi^\top \right\|_F,$$

where $\left\| A^{-1} \Phi^\top \right\|_F$ is the Frobenius norm of the re-scaled feature matrix.

As we are applying an interactive algorithm we can consider $f_{\mathbf{w}} = \sum_t \delta f_{\mathbf{w}_t}$ with $\delta f_{\mathbf{w}_t} = \langle \delta \mathbf{w}_t, \Phi(\mathbf{x}) \rangle$. Hence, we can apply a local constraint on the parameter updates by considering the restricted class of functions

$$\mathcal{F}_{\mathbf{m}}^{\mathbf{A}} = \left\{ f_{\mathbf{w}} : \mathbf{x} \mapsto \sum_t \langle \delta \mathbf{w}_t, \Phi(\mathbf{x}) \rangle : \|\delta \mathbf{w}_t\|_{A_t} \leq m_t \right\}.$$

Theorem 16.5 *Given any sequences* $\mathbf{A}$ *and* $\mathbf{m}$ *of invertible matrices* $A_t \in \mathbb{R}^{P \times P}$ *and positive numbers* $m_t > 0$ *we have the bound*

$$\hat{\mathfrak{R}}\left(\mathcal{F}_{\mathbf{m}}^{\mathbf{A}}\right) \leq \sum_t \frac{m_t}{n} \left\| A_t^{-1} \Phi^\top \right\|_F.$$

Remark 16.6 *By the linear re-parameterisation* $\mathbf{w} \mapsto A^\top \mathbf{w}$ *and* $\Phi \mapsto A^{-1}\Phi$ *we can equivalently consider the restricted function class*

$$\mathcal{F}_{\mathbf{m}}^{\Phi} = \left\{ f_{\mathbf{w}} : \mathbf{x} \mapsto \sum_t \left\langle \tilde{\delta} \mathbf{w}_t, \Phi_t(\mathbf{x}) \right\rangle : \left\| \tilde{\delta} \mathbf{w}_t \right\|_2 \leq m_t \right\}.$$

From this, we get the bound

$$\hat{\mathfrak{R}}_S \leq \sum_t \frac{m_t}{n} \|\Phi_t\|_F.$$

These bounds are for fixed sequences of feature maps. However, the results can be extrapolated to the non-deterministic setting for which [11] proposes the following heuristic measure for the complexity of neural networks,

$$C(f_{\mathbf{w}}) = \sum_t \|\delta \mathbf{w}_t\| \|\Phi_t\|_F.$$

Where Φ_t is the learned tangent feature matrix at iteration t, and $\|\delta \mathbf{w}_t\|_2$ is the norm of the SGD update.

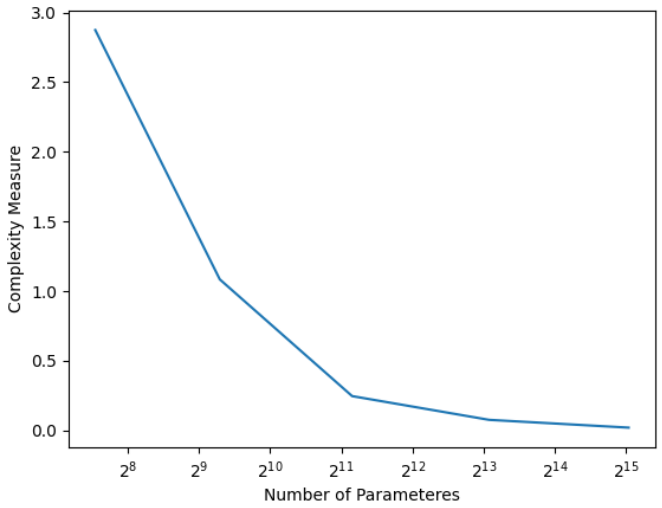

Figure 9: Complexity of fully connected networks with a certain number of parameters trained up to the point of 0.01 Cross Entropy Loss on a synthetic data set.

17 Geometric Complexity Measure

We previously saw that Rademacher complexity is a measure of complexity that focuses on the entire hypothesis space rather than focusing on a function. Similarly, complexity notions such as VC dimension and parameter counting focus on quantifying complexity for the entire hypothesis space. On the other hand, measures such as counting the number of linear pieces of ReLU networks and matrix norms, measure the complexity of the function independently from the task at hand. The work of [13] focuses on capturing function complexity over proposed datasets through geometrical arguments with the following proposed complexity measure.

Definition 17.1 *Let $g_\theta : \mathbb{R}^d \to \mathbb{R}^k$ be a neural network parameterised by θ. We can write $g_\theta(x) = \alpha(f_\theta(x))$ where α is the last layer activation, and f_θ is its logit network. The Geometric Complexity (GC) of the network over a dataset D is the discrete Dirichlet energy of its logit network,*

$$\langle f_\theta, D\rangle = \frac{1}{|D|}\sum_{x\in D} \|\nabla_x f_\theta(x)\|_F^2,$$

where $\|\nabla_x f - \theta(x)\|_F$ is the Frobenius norm of the network Jacobian.

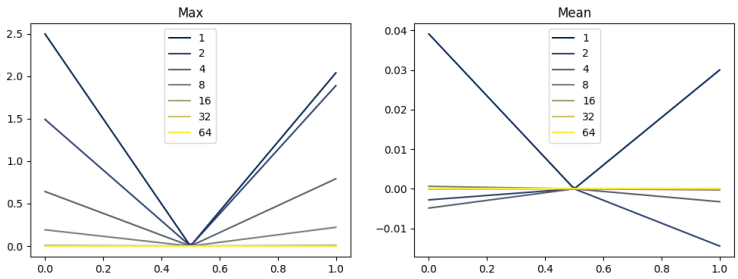

Figure 10: Determining the function of a ReLU network with a variable number of layers when initialized using the Glorot Initialization scheme.

17.1 GC and ReLU Networks

Let $g_\theta : \mathbb{R}^d \to \mathbb{R}^k$ be a ReLU network. As g_θ parameterises piece-wise linear function, the domain can be partitioned by subsets $X_i \subset \mathbb{R}^d$ such that f_θ is an affine map $A_i x + b_i$. Let $D_i = D \cap X_i$, as for every $x \in X_i$ it follows that $\|\nabla_x f_\theta(x)\|_F^2 = \|A_i\|_F^2$ it follows that

$$\langle f_\theta, D \rangle_G = \sum_i \left(\frac{n_i}{|D|} \right) \|A_i\|_F^2,$$

where $n_i = |D_i|$. Hence, a stratified batch $B \subset D$ has a GC that coincides with the network. Hence, the GC on large batches can be used as a proxy for the GC of a network, making this a tractable measure of complexity to attain during training.

18 Algorithmic Stability

We continue our focus on the learning algorithm to understand network generalization by looking at the work of [12]. When taking an algorithmic perspective it is important to understand how gradient descent exploits patterns in the training data as networks are usually trained through gradient-based methods. The only point at which this exploitation could occur is in the parameter update step, and so this is the main focus of the section. A general update rule takes the form

$$w_{t+1} = w_t - \eta \frac{1}{m} \sum_{i=1}^{m} g_i(w_t),$$

where $g_i(w_t)$ is the gradient of the loss function at example i evaluated at the parameter w_t, and η is the learning rate. From this, we observe the following:

1. The average gradient $g(w_t)$ is strong in directions where per-example gradients are "similar".
2. Parameters that correspond to stronger directions change more.

When per-example gradients reinforce each other they are said to be coherent. We call a learning process stable if changing an example in the training set doesn't affect the learned model in drastic ways. From above we've observed that strong directions in the average gradient are stable, as we have multiple examples reinforcing each other. Hence, we can relate the stability of the learning process to the ability to generalize. That is, stable updates in the parameters should lead to good generalization. Therefore, to approach generalization from the perspective of learning algorithms we need to quantify the notion of coherence.

18.1 Motivating Example

Consider the linear model $\hat{y} = w \cdot x = \sum_{i=1}^{6} w_i x_i$ which is fitted using the square loss $l(w) = \frac{1}{2}(y - \hat{y})$. Consider the datasets

$$\begin{bmatrix} i & & & x_i & & & & y_i \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 2 & 0 & -1 & 0 & 0 & 0 & -1 & -1 \\ 3 & 0 & 0 & -1 & 0 & 0 & -1 & -1 \\ 4 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 5 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} i & & & x_i & & & & y_i \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\ 3 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ 4 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 5 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \end{bmatrix},$$

where the difference between the two only arises in the last entry of the x_i. Refer to the first one as "real" data and the second as "random" data. Use the first 4 rows for training and the last for testing.

Consider the average gradient of the loss function on each of these datasets. For the real data set the 6^{th} is reinforced by each example, on the other hand, in the random dataset this component is not enforced. Hence, during gradient descent, the real dataset causes a greater change in the 6^{th} component of the parameter compared to the random data set. We know that the 6^{th} is what distinguishes the data set and it is the learnable feature of the real dataset. It is reassuring then that the gradients of the data points reinforce this feature so that the model can learn this. Refer to Figure 11

18.2 Coherence

Let $\mathcal{D}(z)$, the distribution of examples z from finite set Z. Suppose our network has d trainable parameters. Let $l_z(w)$ be the loss of example $z \sim \mathcal{D}$ for parameter vector $w \in \mathbb{R}^d$ and let $l(w) = \mathbb{E}_{z\sim\mathcal{D}}(l_z(w))$ be the expected loss. We denote the gradient of the loss at example z by $g_z := (\nabla l_z)(w)$ and the average gradient by $g := (\nabla l)(w)$. Throughout we will use η to denote the learning rate.
The learning problem is trying to minimize the expected loss, often achieved through a gradient descent algorithm. Suppose we take a small descent step $h = -\eta g$. Using Taylor approximations

$$l(w+h) - l(w) \approx g \cdot h = -\eta g \cdot g = -\eta \mathbb{E}_{z\sim\mathcal{D}}(g_z) \cdot \mathbb{E}_{z\sim\mathcal{D}}(g_z) = -\eta \mathbb{E}_{z,z'\sim\mathcal{D}}(g_z \cdot g_{z'}).$$

Which motivates using the average pairwise dot product as a metric for coherence. With this setup, there are clear directions for improving the generalization of networks by modifying the process of gradient descent.

1. Make gradient descent more stable by eliminating weak direction through combining per-example gradients using robust mean estimation techniques (i.e. use the median rather than sample mean).

2. Using l^2 regularization as a means to reduce movement in weak directions.

3. As training progresses the gradients of fitted examples become negligible. Therefore, earlier stopping may prevent the fewer examples in the latter parts of training from dominating the average gradient and triggering over-fitting.

To use the average pairwise dot product as an interpretable metric it has to be normalized. For an individual loss l_z consider a step h_z down its gradient g_z at a parameter point w,

$$l_z(w+h_z) - l_z(w).$$

Using Taylor approximations and taking expectations over z yields

$$\mathbb{E}_{z\sim\mathcal{D}}\left(l_z(w+h_z) - l_z(w)\right) = -\eta \mathbb{E}_{z\sim\mathcal{D}}(g_z \cdot g_z).$$

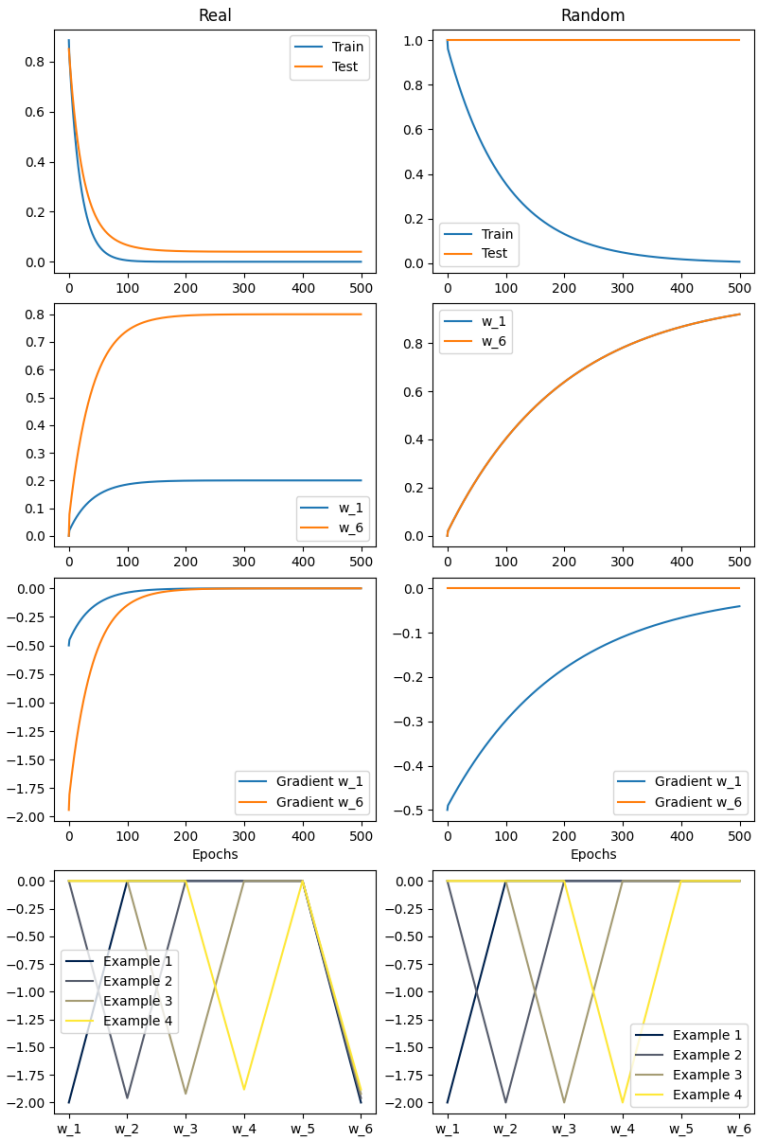

Figure 11: Motivating Example Plots

This is the idealized reduction in loss, as per-example examples tend not to be aligned so steps are usually not taken down an individual losses gradient. Therefore, it serves as a reasonable scaling factor for our metric. Let α be the normalized metric for coherence defined as

$$\alpha(\mathcal{D}) := \frac{\mathbb{E}_{z,z' \sim \mathcal{D}}(g_z \cdot g_{z'})}{\mathbb{E}_{z \sim \mathcal{D}(g_z \cdot g_z)}}.$$

Theorem 18.1 *Let $\mathcal{V}$ be a probability distribution on a collection of m vectors in Euclidean space. Then, $0 \leq \alpha(\mathcal{V}) \leq 1$, where*

1. *$\alpha(\mathcal{V}) = 0$ if and only if $\mathbb{E}_{v \sim \mathcal{V}}(v) = 0$, and*
2. *$\alpha(\mathcal{V}) = 1$ if and only if all vectors are equal.*

Furthermore, for $0 \neq k \in \mathbb{R}$ we have

$$\alpha(k\mathcal{V}) = \alpha(\mathcal{V})$$

where $k\mathcal{V}$ is the distribution of random variables kv for $v \sim \mathcal{V}$.

Remark 18.2 *Typically one does not have access to the underlying distribution of the samples. So we sample our data and let α_m denote the coherence of a sample S where $|S| = m$.*

Example 18.3 *Let S be a sample of size m, whose gradients g_i ($1 \leq i \leq m$) are pairwise orthogonal. Then,*

$$\alpha_m = \frac{\frac{1}{m}\mathbb{E}_i(g_i \cdot g_i)}{\mathbb{E}_i(g_i \cdot g_i)} = \frac{1}{m}.$$

Therefore, in this case, α_m is independent of the g_i. Call $\frac{1}{m}$ the orthogonal limit of a sample of size m and denote it $\alpha_m^\perp$.

From here on we consider the quantity $\alpha_m/\alpha_m^\perp$ as it has a simple interpretation.

1. For an orthogonal sample $\alpha_m/\alpha_m^\perp = 1$. Examples are fitted independently.
2. For a perfectly coherent sample $\alpha_m/\alpha_m^\perp = m$. Examples help fit all other examples.
3. When $\alpha_m/\alpha_m^\perp = 0$, we have $\alpha_m = 0$ so on average no example helps fit any other example.
4. Consider the under-parameterized setting $d \ll m$ and a sample of $k \gg 1$ copies of orthogonal gradients in d-dimensional space (i.e. $m = kd \gg d$). In this case $\alpha_m = \frac{1}{d}$ and $\alpha_m/\alpha_m^\perp = k$. Agreeing with the notion that each example helps k other examples in the sample.

18.3 Application to the Generalization Gap

It has been shown that small batch stochastic gradient descent is stable as each individual example is looked at so rarely when training is not run for too long. However, this is a dataset-independent argument for stability. In [12] a dataset-dependent argument for stability is given where coherence implies stability which in turn implies generalization. The expected generalization gap is the expected difference between training and test loss over samples of size m from $\mathcal{D}$, $\text{gap}(\mathcal{D}, m)$.

Theorem 18.4 *If stochastic gradient descent is run for T steps on a training set consisting of m examples drawn from distribution $\mathcal{D}$, then,*

$$|\text{gap}(\mathcal{D}, m)| \leq \frac{L^2}{m} \sum_{t=1}^{T} (\eta_k \beta)_{k=t+1}^{T} \cdot \eta_t \cdot \sqrt{2(1 - \alpha(w_{t-1}))}$$

where

- *$\alpha(w)$ denotes coherence at point w in parameter space,*
- *w_t is the parameter value seen at step t of gradient descent,*
- *η_t is the learning rate a step t of gradient descent,*
- *$(\eta_k \beta)_{k=t_0}^{t_1} = \prod_{k=t_0}^{t_1} (1 + \eta_k \beta)$, and*
- *L and β are Lipschitz constants.*

Remark 18.5

- *The bound is dependent on the training length, and size of the training set.*
- *High coherence early on in training is better than high coherence later on.*
- *The bound applied uniformly to stochastic and full-batch cases.*
- *The bound is only useful in a qualitative sense and is loose.*

Theorem 18.4 is stated with little context, in the following we develop the context of the theorem more formally. First, recall some notation, we have a domain of samples Z with $\mathcal{D}(Z)$ a distribution on Z from which training and test examples are sampled. Let $l(w, z)$ be the loss on example z with model parameters w. Machine learning then aims to minimize the population risk $R(w) = \mathbb{E}_{z \sim \mathcal{D}} l(w, z)$. In practice when equipped with training samples $S = (z_1, \ldots, z_m)$ we instead minimize

$$\hat{R}(w, S) := \frac{1}{m} \sum_{i \in [m]} l(w, z_i).$$

By appealing to ideas in algorithmic stability we want a bound on the expected generalization bound

$$\text{gap}(\mathcal{D}, m) := \mathbb{E}_{S\sim\mathcal{D}^m}\mathbb{E}_\theta\left(R(A_\theta(S)) - \hat{R}(A_\theta(S), S)\right)$$

where $A_\theta(S)$ is the model obtained by running the randomized training algorithm A on S, with θ being the sequence of decisions made by the stochastic processes of A. If we have a second sample $S' := (z'_1, \ldots, z'_m)$ then the expected stability of A is given by

$$\text{stab}(\mathcal{D}, m) := \mathbb{E}_{S\sim\mathcal{D}^m}\mathbb{E}_{S'\sim\mathcal{D}^m}\mathbb{E}_\theta\left(\frac{1}{m}\sum_{i\in[m]}\left(l\left(A_\theta\left(S^{(i)}\right), z_i\right) - l(A_\theta(S), z_i)\right)\right)$$

where $S^{(i)} = \left(z_1, \ldots, z_{i-1}, z'_i, z_{i+1}, \ldots, z_m\right)$.

Theorem 18.6 *When training with m samples from a distribution $\mathcal{D}$, we have,*

$$\text{gap}(\mathcal{D}, m) = \text{stab}(\mathcal{D}, m).$$

To proceed we make some assumptions on the smoothness of the loss function. With $g(w, z) = \nabla_w l(w, z)$ we assume that for all $z \in Z$ we have,

- $|l(w, z) - l(w', z)| \leq L\|w - w'\|$
- $\|g(w, z)\| \leq L$, and
- $\|g(w, z) - g(w', z)\| \leq \beta\|w - w'\|$.

Suppose during T steps of gradient descent training on the datasets S and $S^{(i)}$ we observe the weight parameters $w_0, \ldots, w_T$ and $w'_0, \ldots, w'_T$ respectively. In the context of stochastic gradient descent suppose that we choose the mini-batches of size b using a random without-replacement sample technique. Use $I_t(\theta)$ to denote the indicator random variable for the event that the i^{th} training example is selected in the mini-batch of time-step t. Now starting from the update rule

$$w_t = w_{t-1} - \eta_t\frac{1}{b}\sum_{j\in[b]} g(w_{t-1}, z_{tj})$$

we deduce that

$$\delta_t \leq \delta_{t-1} + \eta_t\|\Delta''_t(b)\|$$

where

- $\delta_t = \|w_t - w'_t\|$, and
- $\Delta''_t(b) = \frac{1}{b}\sum_{j\in[b]}\left(g\left(w_{t-1}, z_{tj}\right) - g\left(w'_{t-1}, z'_{tj}\right)\right)$.

Lemma 18.7 *For $t \in [T]$, we have*

$$\delta_t \leq (1 + \eta_t \beta) \cdot \delta_{t-1} + \frac{\eta_t I_t(\theta)}{b} \|g(w_{t-1}, z_i) - g(w_{t-1}, z_i')\|.$$

Lemma 18.8 *We have,*

$$\delta_T \leq \sum_{t \in [T]} \left(\frac{\eta_t I_t(\theta)}{b} \|g(w_{t-1}, z_i) - g(w_{t-1}, z_i')\| \prod_{k=t+1}^{T} (1 + \eta_k \beta) \right).$$

Lemma 18.9 *We have,*

$$\mathbb{E}_{S \sim \mathcal{D}^m} \mathbb{E}_{z_i' \sim \mathcal{D}} \mathbb{E}_\theta \left(\left| l\left(A_\theta\left(S^{(i)}\right), z_i\right) - l(A_\theta(S), z_i) \right| \right) \leq \frac{L^2}{m} \sum_{t \in [T]} (\eta_k \beta)_{k=t+1}^{T} \eta \sqrt{2(1 - \alpha(w_{t-1})}.$$

From these Lemmas we get the following theorem.

Theorem 18.10

$$|\text{stab}(\mathcal{D}, m)| \leq \frac{L^2}{m} \sum_{t=1}^{T} (\eta_k \beta)_{k=t+1}^{T} \cdot \eta_t \cdot \sqrt{2(1 - \alpha(w_{t-1}))}.$$

Theorem 18.4 is a direct consequence by applying Theorem 18.6. We can apply Theorem 18.4 in different learning regimes to yield the following corollaries.

Corollary 18.11 *If the steps are fixed, $\eta_t = \eta$, then*

$$|\text{gap}(\mathcal{D}, m)| \leq \frac{L^2 \eta}{m} \sum_{t=1}^{T} \exp((T - t)\eta\beta) \cdot \sqrt{2(1 - \alpha(w_{t-1}))}.$$

Corollary 18.12 *If the step size decay linearly, that is for some $\eta > 0$ we have $\eta_t \leq \frac{\eta}{t}$, then*

$$|\text{gap}(\mathcal{D}, m)| \leq \frac{L^2 \eta T^{\eta\beta}}{m} \sum_{t=1}^{T} \sqrt{2(1 - \alpha(w_{t-1}))}.$$

18.4 Coherence of Fully Connected Neural Networks

For this, we follow one of the approaches investigated in [12]. We train a fully-connected one layer neural network with 2048 neurons using SGD and a learning rate of 0.1. We investigate how $\alpha_m / \alpha_m^\perp$ evolves during training when the network is applied to classifying the MNIST dataset. We then compare this to the evolution $\alpha_m / \alpha_m^\perp$ on the same network but trained on random data. The random data is generated by assigning random labels to the MNIST dataset. There are several issues that inhibit the ability to compute coherence through training:

1. Batch normalization means that it is not possible to recover per-example gradients.

2. Large training sets mean it would be impractical to consider computing the coherence even if we had access to per-example gradients.

To alleviate these issues we employ a method to approximate the $\alpha_m/\alpha_m^\perp$ from the batch data.

Theorem 18.13 *Let $v_1, \ldots, v_k$ be k* i.i.d *variables drawn from $\mathcal{V}$. Let $\mathcal{W}$ denote the distribution of the random variable $w = \frac{1}{k}\sum_{i=1}^{k} v_i$. We have,*

$$\alpha(\mathcal{W}) = \alpha(k\mathcal{W}) = \frac{k\alpha(\mathcal{V})}{1 + (k-1)\alpha(\mathcal{V})}$$

Furthermore, $\alpha(\mathcal{W}) \geq \alpha(\mathcal{V})$ with equality if and only if $\alpha(\mathcal{V}) = 0$ or $\alpha(\mathcal{V}) = 1$.

Using this Theorem 18.13 we can construct a tractable method for approximating coherence from gradients computed at the batch level rather than the per-example level.

1. Computer the gradients for each batch.
2. Computer coherence of the $\frac{m}{k}$ batch in the usual way, $\alpha(\mathcal{W})$.
3. Re-arrange the identity of Theorem 18.13,

$$\alpha(\mathcal{V}) = \frac{\alpha(\mathcal{W})}{k - (k-1)\alpha(\mathcal{W})}$$

   and use this to get a value for $\alpha(\mathcal{V})$.
4. Computer $\alpha_m/\alpha_m^\perp$ as $m\alpha(\mathcal{V})$.

The code for this implementation can be founder here.

18.5 Suppressing Weak Gradients

In theory, suppressing weak directions in the average gradient should lead to less over-fitting and better generalization. Indeed current regularization techniques perform this implicitly, however, winsorized gradient descent (WGD) aims to do it explicitly.

18.5.1 Winsorized Gradient Descent Let $w_t^{(j)}$ be the j^{th} component of the trainable parameter w_t and let $g_i^{(j)}(w_t)$ be the j^{th} component of the gradient of the i^{th} example at w_t. Let $c \in [0, 50]$ be the level of winsorization. Define $l^{(j)}$ and $u^{(j)}$ be the c^{th} and $(100-c)^{\text{th}}$ percentile of $g_i^{(j)}(w_t)$ respectively. Then the update rule for WGD is

$$w_{t+1}^{(j)} = w_t^{(j)} - \frac{\eta}{m}\sum_{i=1}^{m} \text{clip}\left(g_i^{(j)}(w_t), l^{(j)}, u^{(j)}\right)$$

where $\text{clip}\left(g_i^{(j)}(w_t), l^{(j)}, u^{(j)}\right) := \min(\max(x, l), u)$. The parameter c acts as a threshold as to what to consider an outlier. A similar stochastic version, winsorized stochastic gradient descent (WSGD), can

also be employed to reduce the computational costs. In any case WGD incurs greater computational costs as it necessitates storing all per-example (batch) gradients to perform the update. Furthermore, higher winsorization values lead to optimization instability as training accuracy is observed to fall after a certain point. To address these issues one can use the median of means algorithm:

1. Divide the sample into k groups,
2. Compute sample mean of each group,
3. Return the median of these k means.

Part V
Other Approaches

19 Unit-Wise Capacity Measures

19.1 Introduction

It is generally accepted that having a more straightforward function that correctly classifies a dataset is more likely to generalize well to unseen data. Indeed the generalization bounds we have discussed so far reward simpler models, where simpler is dependent on the setting we are operating in. In the case of Bayesian machine learning, we designed priors to assign mass to more straightforward functions by assigning greater mass to functions with short code representations. In the section on compressibility, we were able to bound the generalization error of compressed models by their uncompressed counterparts and actively tried to find these models by developing compression algorithms. In this scenario, we similarly took simple to mean represented by fewer parameters. In both these cases the notion of complexity was data-independent, in this section we consider bounds that use a notion of complexity dependent on the relationship between the training data and the hypothesis class of function. In this case, the ideas of complexity follow from theoretical foundations, however, in subsequent sections, we will identify empirical heuristics for model complexity that provide a good indication of the generalization capacity of the model.

19.2 Setup

In this section, we follow the work of [6] and look at two-layer fully connected ReLU networks. Where the inputs have dimension d, outputs have dimension c, and layers have h hidden units. The function of the network can be represented as $f_{\mathbf{V},\mathbf{U}}(\mathbf{x}) = \mathbf{V}(\mathbf{U}\mathbf{x})_+$ where $\mathbf{x} \in \mathbb{R}^d, \mathbf{U} \in \mathbb{R}^{h\times d}$ and $\mathbf{V} \in \mathbb{R}^{c\times h}$. We let $\mathbf{u}_i$ denote the incoming weights to hidden unit i and $\mathbf{v}_i$, the outgoing weights to hidden unit i. Typically, one applies a learning algorithm to some initialized weight values, hence, we let $\mathbf{U}_0$ and $\mathbf{V}_0$ be the initialized weights and have $\mathbf{u}_i^0$ and $\mathbf{v}_i^0$ denote be the corresponding weights as defined above. The network will be used to perform c-class classification, where the maximum output of a score function gives the label. We define this score function to be the margin operator $\mu : \mathbb{R}^c \times [c] \to \mathbb{R}$ which scores an output $f(\mathbf{x})$ for each label $y \in [c]$ according to $\mu(f(\mathbf{x}), y) = f(\mathbf{x})[y] - \max_{i\neq y} f(\mathbf{x})[i]$. To train the network we use the ramp loss,

$$l_\gamma(f(\mathbf{x}), y) = \begin{cases} 0 & \mu(f(\mathbf{x}), y) > \gamma \\ \frac{\mu(f(\mathbf{x}),y)}{\gamma} & \mu(f(\mathbf{x}), y) \in [0, \gamma] \\ 1 & \mu(f(\mathbf{x}), y) < 0. \end{cases}$$

Hence, we have the following definitions of error,

- $L_\gamma(f) = \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}(l_\gamma(f(\mathbf{x}, y))$, expected margin loss of a predictor $f(\cdot)$, distribution $\mathcal{D}$ and margin $\gamma > 0$.
- $\hat{L}_\gamma(f)$, empirical margin loss.
- $L_0(f)$, expected risk.
- $\hat{L}_0(f)$, expected training error.

19.3 Generalization Bounds

Suppose we have a hypothesis class $\mathcal{H}$, and a training set S. Let $l_\gamma \circ \mathcal{H} := \{l_\gamma \circ h : h \in \mathcal{H}\}$.

Definition 19.1 *Let $\mathfrak{R}_S(\mathcal{H})$ be the Rademacher complexity of a class of functions $\mathcal{H}$ with respect to the training set S of size m, defined to be*

$$\mathfrak{R}_S(\mathcal{H}) = \frac{1}{m}\mathbb{E}_{\xi\in\{\pm1\}^m}\left(\sup_{f\in\mathcal{H}}\sum_{i=1}^{m}\xi_i f(x_i)\right).$$

Theorem 19.2 ([5]) *With probability $1-\delta$ over a training set of size m,*

$$L_0(f) \leq \hat{L}_\gamma(f) + 2\mathfrak{R}_S(l_\gamma \circ \mathcal{H}) + 3\sqrt{\frac{\log\left(\frac{2}{\delta}\right)}{2m}}$$

holds for any $f \in \mathcal{H}$.

Hence, we see that Rademacher complexity is a notion of complexity that provides theoretical bounds on the expected risk of the function of our network. We see that it is dependent on how our hypothesis class is able to separate the data points of our training set. We are exploring now the work of [6] that aims to find a tractable heuristic for Rademacher complexity to get bounds on the generalization error.

Definition 19.3 *The unique capacity of a hidden unit i is $\beta_i = \left\|\mathbf{u}_i - \mathbf{u}_i^0\right\|_2$.*

Definition 19.4 *The unit impact of a hidden unit i is $\alpha_i = \|\mathbf{v}_i\|_2$.*

Definition 19.5 *Let $\mathcal{W}$ be the restricted set of parameters*

$$\mathcal{W} = \left\{(\mathbf{V},\mathbf{U}) : \mathbf{V}\in\mathbb{R}^{c\times h}, \mathbf{U}\in\mathbb{R}^{h\times d}, \|\mathbf{v}_i\| \leq \alpha_i, \left\|\mathbf{u}_i - \mathbf{u}_i^0\right\|_2 \leq \beta_i\right\}$$

and let $\mathcal{F}_\mathcal{W}$ be the corresponding class of neural networks

$$\mathcal{F}_\mathcal{W} = \{f(\mathbf{x}) = \mathbf{V}(\mathbf{U}\mathbf{x})_+ : (\mathbf{V},\mathbf{U}) \in \mathcal{W}\}$$

Theorem 19.6 *Given a training set* $S = \{\mathbf{x}_i\}_{i=1}^m$ *and* $\gamma > 0$*, then*

$$\begin{aligned}\mathfrak{R}_S(l_\gamma \circ \mathcal{F}_\mathcal{W} &\leq \frac{2\sqrt{2c}+2}{\gamma m}\sum_{j=1}^{h}\alpha_j\left(\beta_j\|\mathbf{X}\|_F + \left\|\mathbf{u}_j^0\mathbf{X}\right\|_2\right)\\ &\leq \frac{2\sqrt{2c}+2}{\gamma m}\|\alpha\|_2\left(\|\beta\|_2\sqrt{\frac{1}{m}\sum_{i=1}^{m}\|\mathbf{x}_i\|_2^2} + \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left\|\mathbf{U}^0\mathbf{x}_i\right\|_2^2}\right).\end{aligned}$$

Theorem 19.7 *For any* $h \geq 2, \gamma > 0, \delta \in (0,1)$ *and* $\mathbf{U}^0 \in \mathbb{R}^{h\times d}$ *with probability* $1-\delta$ *over the training set* $S = \{\mathbf{x}_i\}_{i=1}^m \subset \mathbb{R}^d$*, for any function* $f(\mathbf{x}) = \mathbf{V}(\mathbf{U}\mathbf{x})_+$ *such that* $\mathbf{V} \in \mathbb{R}^{c\times h}$ *and* $\mathbf{U} \in \mathbb{R}^{h\times d}$*,*

$$\begin{aligned}L_0(f) &\leq \hat{L}_\gamma(f) + \tilde{O}\left(\frac{\sqrt{c}\|\mathbf{V}\|_F\left(\left\|\mathbf{U}-\mathbf{U}^0\right\|_F\|\mathbf{X}\|_F + \left\|\mathbf{U}^0\mathbf{X}\right\|_F\right)}{\gamma m} + \sqrt{\frac{h}{m}}\right)\\ &\leq \hat{L}_\gamma(f) + \tilde{O}\left(\frac{\sqrt{c}\|\mathbf{V}\|_F\left(\left\|\mathbf{U}-\mathbf{U}^0\right\|_F + \left\|\mathbf{U}^0\right\|_2\right)\sqrt{\frac{1}{m}\sum_{i=1}^{m}\|\mathbf{x}_i\|_2^2}}{\gamma\sqrt{m}} + \sqrt{\frac{h}{m}}\right)\end{aligned}$$

Therefore, the term $\|\mathbf{V}\|_F\left(\left\|\mathbf{U}-\mathbf{U}^0\right\|_F + \left\|\mathbf{U}^0\right\|_2\right) + \sqrt{h}$ can be used as a heuristic for complexity.

20 Validation Paradigm

20.1 Introduction

Here we investigate the work of [14] which looks into how the training-validation paradigm leads to deep neural networks that generalize well. In this paradigm, a validation set of data is held out to optimize the model architecture and hyper-parameters. Giving rise to the hypothesis that *deep neural networks can obtain good generalization error by performing a model search on the validation set*.

20.2 Setup

We again consider an input $x \in \mathcal{X}$ and a label $y \in \mathcal{Y}$. The loss function is denoted $\mathcal{L}$ and $\mathcal{R}[f] = \mathbb{E}_{x,y\sim\mathbb{P}_{(\mathcal{X},\mathcal{Y})}}(\mathcal{L}(f(x),y))$ is the expected risk of a function f for $\mathbb{P}_{(\mathcal{X},\mathcal{Y})}$ being the true distribution. Let $f_{\mathcal{A}(S)} : \mathcal{X} \to \mathcal{Y}$ denote the function learnt by a learning algorithm $\mathcal{A}$ on training set $S := S_m := \{(x_1,y_1),\dots,(x_m,y_m)\}$. The set of possible learned functions is characterized by the hypothesis space $\mathcal{F}$. Associated with this space we have the family of loss functions $\mathcal{L}_{\mathcal{F}} := \{g : g \in \mathcal{F}, g(x,y) = \mathcal{L}(f(x),y)\}$, the family of loss functions associated to $\mathcal{F}$.

Definition 20.1 ([5]) *Let $\mathcal{D}$ be a distribution from which samples are drawn. For any integer $m \geq 1$, the Rademacher complexity of a family of functions $\mathcal{G}$ is the expectation of the empirical Rademacher complexity over all samples of size m drawn from $\mathcal{D}$. That is,*

$$\mathfrak{R}_m(\mathcal{G}) = \mathbb{E}_{S\sim\mathcal{D}^m}\left(\hat{\mathfrak{R}}_S(\mathcal{G})\right).$$

20.3 Generalization Bounds

Machine learning aims to minimize $\mathcal{R}\left(f_{\mathcal{A}(S)}\right)$. However, this is **non-computable** as $\mathbb{P}_{(\mathcal{X},\mathcal{Y})}$ is unknown. Therefore, one minimizes the empirical risk

$$\mathcal{R}_S\left(f_{\mathcal{A}(S)}\right) = \frac{1}{|S|}\sum_{(x,y)\in S}\mathcal{L}(f_{\mathcal{A}(S)}(x),y),$$

where the generalization gap is defined to be $\mathcal{R}\left(f_{\mathcal{A}(S)}\right) - \mathcal{R}_S\left(f_{\mathcal{A}(S)}\right)$.

Proposition 20.2 *Let $S^{(\mathrm{val})}_{m_{\mathrm{val}}}$ be a held-out validation set, where $\left|S^{(\mathrm{val})}_{m_{\mathrm{val}}}\right| = m_{\mathrm{val}}$. Assume that m_{val} is an* i.i.d *sample from $\mathbb{P}_{(X,Y)}$. Let $\kappa_{f,i} = \mathcal{R}(f) - \mathcal{L}(f(x_i),y_i)$ for $(x_i,y_i) \in S^{(\mathrm{val})}_{m_{\mathrm{val}}}$. Suppose that $\mathbb{E}\left(\kappa^2_{f,i}\right) \leq \gamma^2$ and $|\kappa_{f,i}| \leq C$ almost surely for all $(f,i) \in \mathcal{F}_{\mathrm{val}} \times \{1,\dots,m_{\mathrm{val}}\}$. Then, for $\delta \in (0,1]$, with probability $1-\delta$*

$$\mathcal{R}(f) \leq \mathcal{R}_{S^{(\mathrm{val})}_{m_{\mathrm{val}}}}(f) + \frac{2C\log\left(\frac{|\mathcal{F}_{\mathrm{val}}|}{\delta}\right)}{3m_{\mathrm{val}}} + \sqrt{\frac{2\gamma^2\log\left(\frac{|\mathcal{F}_{\mathrm{val}}|}{\delta}\right)}{m_{\mathrm{val}}}}$$

holds for all $f \in \mathcal{F}_{\mathrm{val}}$.

Remark 20.3

- $\mathcal{F}_{\text{val}}$ *is independent of* $S_{m_{\text{val}}}^{(\text{val})}$.
- *The bound is only dependent on the validation error on* $S_{m_{\text{val}}}^{(\text{val})}$.

The dependence on $|\mathcal{F}_{\text{val}}|$ can be alleviated by utilizing the following theorem and its corollary.

Theorem 20.4 ([5]) *Let* $\mathcal{G}$ *be a family of functions with co-domain* $[0,1]$. *Then, for any* $\delta > 0$, *with probability at least* $1-\delta$ *over an* i.i.d *sample* S *of size* m,

$$\mathbb{E}(g(z)) \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 2\mathfrak{R}_m(\mathcal{F}) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2m}}$$

for all $g \in \mathcal{G}$.

Corollary 20.5 *Assume* $S_{m_{\text{val}}}^{(\text{val})}$ *is an* i.i.d *sample from* $\mathbb{P}_{(\mathcal{X},\mathcal{Y})}$. *Let* $\mathcal{L}_{\mathcal{F}_{\text{val}}} = \{g : f \in \mathcal{F}_{\text{val}}, g(x,y) := \mathcal{L}(f(x),y)\}$. *Apply the above theorem to* $\mathcal{L}_{\mathcal{F}_{\text{val}}}$ *when* $\mathcal{L}$ *has co-domain* $[0,1]$ *to deduce that*

$$\mathcal{R}(f) \leq \mathcal{R}_{S_{m_{\text{val}}}^{(\text{val})}}(f) + 2\mathfrak{R}_m\left(\mathcal{L}_{\mathcal{F}_{\text{val}}}\right) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{m_{\text{val}}}}.$$

Some code for implementing this paradigm can be found here.

References

[1] David A. McAllester. “Some PAC-Bayesian Theorems”. In: *Machine Learning* 37 (1998), pp. 355–363.

[2] Gintare Karolina Dziugaite and Daniel M. Roy. “Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data”. In: *CoRR* (2017).

[3] Ravid Shwartz-Ziv and Naftali Tishby. “Opening the Black Box of Deep Neural Networks via Information”. In: *CoRR* (2017).

[4] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. “Stronger generalization bounds for deep nets via a compression approach”. In: *CoRR* (2018).

[5] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2018.

[6] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. “Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks”. In: *CoRR* (2018).

[7] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. “On the Information Bottleneck Theory of Deep Learning”. In: *International Conference on Learning Representations*. 2018.

[8] Stanislav Fort, Pawel Krzysztof Nowak, and Srini Narayanan. “Stiffness: A New Perspective on Generalization in Neural Networks”. In: *CoRR* (2019).

[9] Benjamin Guedj. *A Primer on PAC-Bayesian Learning*. 2019.

[10] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. *Non-Vacuous Generalization Bounds at the ImageNet Scale: A PAC-Bayesian Compression Approach*. 2019.

[11] Aristide Baratin, Thomas George, César Laurent, R. Devon Hjelm, Guillaume Lajoie, Pascal Vincent, and Simon Lacoste-Julien. “Implicit Regularization via Neural Feature Alignment”. In: *CoRR* (2020).

[12] Satrajit Chatterjee and Piotr Zielinski. *On the Generalization Mystery in Deep Learning*. 2022.

[13] Benoit Dherin, Michael Munn, Mihaela Rosca, and David G. T. Barrett. *Why neural networks find simple solutions: the many regularizers of geometric complexity*. 2022.

[14] K. Kawaguchi, Y. Bengio, and L. Kaelbling. “Generalization in Deep Learning”. In: *Mathematical Aspects of Deep Learning*. Cambridge University Press, 2022, pp. 112–148.

Part VI
Appendix

21 Bayesian Machine Learning

Here we will outline an introduction to PAC-Bayesian learning given by [9], to provide context on the PAC-Bayes generalization bounds.

21.1 Framework

Throughout we will assume we have data of the form $\mathcal{D}_n = (X_i, Y_i)_{i=1}^n$ where each sample is a realisation of some random variables $(X, Y) \in \mathbb{R}^d \times \mathbb{R}$ with underlying distribution $\mathbb{P}$. PAC-Bayesian learning is a framework to determine a predictor $\hat{\phi}$ such that $\phi(X') \approx Y'$. A learning algorithm with therefore be a map from data to predictors, we will denote the set of possible predictors $\mathcal{F}$. To assess the quality of predictors we define a loss function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$, and use it to define the risk of a predictor $\hat{\phi}$ as

$$R : \hat{\phi} \mapsto \mathbb{E}_{\mathbb{P}} \left(l \left(\hat{\phi}(X), Y \right) \right).$$

However, $\mathbb{P}$ is usually unknown, therefore in practice, we use the empirical risk which is computable,

$$r_n : \hat{\phi} = \frac{1}{n} \sum_{i=1}^{n} l \left(\hat{\phi}(X_i), Y_i \right).$$

Note that $\mathbb{E} \left(r_n \left(\hat{\phi} \right) \right) = R \left(\hat{\phi} \right)$. Under the Bayesian framework, we assign a prior distribution π on $\mathcal{F}$ for $\hat{\phi}$. The posterior distribution ρ then takes the form

$$\rho \left(\hat{\phi} | \mathcal{D}_n \right) \propto \mathcal{L} \left(\mathcal{D}_n | \hat{\phi} \right) \pi \left(\hat{\phi} \right),$$

where $\mathcal{L}$ is the likelihood function. From here there are multiple methodologies of choosing $\hat{\pi}$ from the posterior distribution ρ. For example, one may take its mean, median or a random realisation to be the chosen predictor $\hat{\phi}$.

22 SGD

1. How SGD finds good solutions only if they are surrounded by relatively large volume solutions that are nearly as good.