Appealing to Gradients to Investigate Neural Network Generalization

Thomas Walker

Summer 2023

Contents

1 Introduction

Gradients are at the core 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 parameters of the network. Therefore, information on the learnable features within a training sample is present within these gradients, and in particular how they evolve and interact with other 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 to refine the training process. In this report, I will indicate some promising exploitation of these gradients in understanding the learning process of neural networks. Although this report will not be an exhaustive list of the application of gradients to investigating neural network generalization, it will provide an overview of some of the enlightening techniques.

2 Introduction

2.1 Notations and Definitions

We will first introduce some basic notation that will remain constant throughout the report. Along the way, we will need to introduce some more specialized notation for the different sections. The problems we will concern ourselves most with will be supervised classification tasks. This means we have a feature space $\mathcal{X}$ and a label space $\mathcal{Y}$ which combine to form the data space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ for which some unknown $\mathcal{D}$ is defined on. The challenge now is to learn a classifier $h : \mathcal{X} \to \mathcal{Y}$ that correctly labels samples from $\mathcal{X}$ according to $\mathcal{D}$. The training data $S = \{(x_i, y_i)\}_{i=1}^m$ consists of m i.i.d samples from $\mathcal{D}$. As we are considering neural networks, a classifier will be parameterized by a weight vector $\mathbf{w}$ which we will denote $h_{\mathbf{w}}$. Let $\mathcal{W}$ denote the set of possible weights for a classifier and the set of all possible classifiers $\mathcal{H}$ will sometimes be referred to as the hypothesis set. We will often denote the set of probability distributions over $\mathcal{W}$ as $\mathcal{M}(\mathcal{W})$. To assess the quality of a classifier we define a measurable function $l : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ called the loss function. This function defines a loss landscape on the distribution parameter space $\mathcal{W}$. At a point $\mathbf{w} \in \mathcal{W}$ the value of the loss landscape is given by the true error of the classifier defined by

$$L(\mathbf{w}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(l(h_{\mathbf{w}}(x), y)).$$

As distribution $\mathcal{D}$ is unknown we work with the training error defined by

$$\hat{L}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} l(h_{\mathbf{w}}(x_i), y_i).$$

It will be useful to use the short-hand $l_i(\mathbf{w}) = l(h_{\mathbf{w}}(x_i), y_i)$. As our training data is just a sample from the underlying (unknown) distribution $\mathcal{D}$ there is the possibility that our classifier performs well on the training data, but performs poorly on the true distribution. Neural network generalization refers to the phenomenon that classifiers trained to have low empirical risk are seen in practice to also have a low true risk. Despite having sufficient capacity to memorize the training data by optimizing to a complex classifier, neural networks are instead observed to optimize to classifiers which learn meta-features of the training sample the generalize well to the underlying distribution. We will now explore techniques that aim to shed light on this phenomenon by appealing to gradients.

2.2 Stochastic Gradient Descent

The architecture of deep neural networks was proposed long before they manifested as a useful machine learning technique. This delay was due in part to the difficulty in training the large architectures stably and effectively. Learning algorithms such as Stochastic Gradient Descent (SGD) have extracted remarkable properties from deep neural network architectures. Many of the properties are still mysterious to researchers, and these architectures seem to have greater potential than what was previously thought. To try and grapple with this it is important to understand the precise mechanisms of SGD as this has instantiated the networks with the majority of these properties. A learning algorithm aims to alter the parameters of the network to try and reach the global minimum of the loss landscape. Gradient descent methods attempt to achieve this by observing how the loss changes with respect to perturbations in the parameters at different training examples to determine the direction in which the parameters should be manipulated to move down the loss landscape. Computing the gradients at each individual training example is computationally expensive, and so instead gradients are calculated for batches of examples. These batches reduce computational costs but also introduce some noise into the algorithm that can be useful in helping the algorithm escape local minima of landscape and find more stable regions. This procedure is known as SGD and has a central role in endowing neural networks with their generalization capacities. It has been observed that SGD can train these networks in the over-parameterized setting such that they converge to global minima of the loss landscape. In [2] an attempt is made to explain this using dynamic stability. This approach illustrates how the randomness induced by SGD is vital, and why regular gradient descent (GD) is not an effective learning algorithm for training neural networks. As is the case with most machine learning scenarios, one is trying

to minimize the training error $\hat{L}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} l_i(\mathbf{w})$ over the parameter space. A general optimizer for this problem can be written as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - G(\mathbf{w}_t; \xi_t), \tag{1}$$

where ξ_t is a random variable independent of $\mathbf{w}_t$ and each ξ_t are i.i.d. For GD,

$$G(\mathbf{w}_t; \xi_t) = \frac{\eta}{m}\sum_{i=1}^{m} \nabla l_i(\mathbf{w}_t),$$

and for SGD,

$$G(\mathbf{w}_t; \xi_t) = \eta \nabla l_i \xi_t(\mathbf{w}_t).$$

Definition 2.1. *Call $\mathbf{w}^*$ a fixed point (1) if for any ξ it follows that $G(\mathbf{w}^*, \xi) = 0$.*

Definition 2.2. *Let $\mathbf{w}^*$ be a fixed point of (1). For the linearized dynamical system,*

$$\tilde{\mathbf{w}}_{t+1} = \tilde{\mathbf{w}}_t - A_{\xi_t}(\tilde{\mathbf{w}}_t - \mathbf{w}^*)$$

where $A_\xi = \nabla_{\mathbf{w}} G(\mathbf{w}^*, \xi_t)$. The fixed point $\mathbf{w}^*$ is linearly stable if there exists a constant C such that

$$\mathbb{E}\left(\|\tilde{\mathbf{w}}_t\|^2\right) \leq C \|\tilde{\mathbf{w}}_0\|^2$$

for all $t > 0$.

If it is assume that $\hat{L}(\mathbf{w}^*) = 0$, and the approximation

$$\hat{L}(\mathbf{w}) \approx \frac{1}{2m}\sum_{i=1}^{m} (\mathbf{w} - \mathbf{w}^*)^\top H_i (\mathbf{w} - \mathbf{w}^*)$$

with $H_i = \nabla^2 l_i(\mathbf{w}^*)$, is used then the linearized SGD is given by

$$\mathbf{w}_{t+1} = x_t - \frac{\eta}{B}\sum_{j=1}^{B} H_{\xi_j}(\mathbf{w}_t - \mathbf{w}^*).$$

Where B is the batch size and $\xi = \{\xi_1, \ldots, \xi_B\}$ is a uniform, non-replaceable random sampling of size B on $\{1, \ldots, m\}$.

Definition 2.3. *Let $H = \frac{1}{m}\sum_{i=1}^{m} H_i$ and $\Sigma = \frac{1}{m}\sum_{i=1}^{m} H_i^2 - H^2$. Let $a = \lambda_{\max}(H)$ be the sharpness, and $s = \lambda_{\max}\left(\Sigma^{\frac{1}{2}}\right)$ be the non-uniformity.*

Theorem 2.4. *The global minimum $\mathbf{w}^*$ is linearly stable for SGD with learning rate η and batch size B if the following is satisfied*

$$\lambda_{\max}\left((1 - \eta H)^2 + \frac{\eta^2(n - B)}{B(n - 1)}\Sigma\right) \leq 1.$$

For $d = 1$ this is a necessary and sufficient condition.

Remark 2.5. *A simpler necessary condition is*

$$0 \leq a \leq \frac{2}{\eta}, \text{ and } 0 \leq s \leq \frac{1}{\eta}\sqrt{\frac{B(n-1)}{n-B}}. \tag{2}$$

For a fixed learning rate η,

1. GD can converge to minima satisfying $a \leq \frac{2}{\eta}$, and
2. SGD can converge to minima satisfying (2).

Hence, SGD can filter out minima with large non-uniformity. The difference between GD and SGD is that SGD must converge to solutions that fit the data uniformly well. This provides a basic demonstration of how the properties of neural networks are a manifestation of the properties of the learning algorithm. It is this investigation that motivates future work to determine more sophisticated probing methods to understand why certain properties of neural networks emerge.

2.3 Complexity Measures

It is generally accepted that having a more straightforward function that correctly classifies a dataset is more likely to generalize well to unseen data. Generalization bounds can often be derived by developing measures of the complexity of the hypothesis class. An important complexity measure is Rademacher complexity, which forms the basis of one of the main results in generalization theory. Recall, that we have the space $\mathcal{Z}$ on which a distribution $\mathcal{D}$ is defined from which we draw an i.i.d sampled $S = \{(x_i, y_i)\}_{i=1}^m$. Suppose we have a class of functions $\mathcal{F} = \{f : \mathcal{Z} \to \mathbb{R}\}$.

Definition 2.6 ([1]). *The empirical Rademacher complexity of $\mathcal{F}$ is*

$$\hat{\mathfrak{R}}(\mathcal{F}) = \mathbb{E}_{\sigma \in \{\pm 1\}} \left(\sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^m \sigma_i f((x_i, y_i)) \right) \right),$$

where each σ_i is an independent random variable uniformly distribution on $\{\pm 1\}$.

Definition 2.7 ([1]). *The Rademacher complexity of $\mathcal{F}$ is*

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

Theorem 2.8 ([1]). *For a parameter $\delta \in (0, 1)$ if $\mathcal{F} \subseteq \{f : \mathcal{Z} \to [0, 1]\}$ then*

$$\mathbb{P}_{S \sim \mathcal{D}^m} \left(\mathbb{E}_{z \sim \mathcal{D}} \left(f(z) \right) \leq \frac{1}{m} \sum_{i=1}^m f(z_i) + 2\mathfrak{R}(\mathcal{F}) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{m}} \right) \geq 1 - \delta,$$

and

$$\mathbb{P}_{S \sim \mathcal{D}^m} \left(\mathbb{E}_{z \sim \mathcal{D}} \left(f(z) \right) \leq \frac{1}{m} \sum_{i=1}^m f(z_i) + 2\hat{\mathfrak{R}}(\mathcal{F}) + 3\sqrt{\frac{\log\left(\frac{2}{\delta}\right)}{m}} \right) \geq 1 - \delta.$$

If we let $\mathcal{F} = \{(x, y) \mapsto l(h_{\mathbf{w}}(x), y) : \mathbf{w} \in \mathcal{W}\}$ then for any $\delta \in (0, 1)$ and $\mathbf{w} \in \mathcal{W}$ we have that

$$\mathbb{P}_{S \sim \mathcal{D}^m} \left(L\left(\mathbf{w}\right) \leq \hat{L}(\mathbf{w}) + 2\hat{\mathfrak{R}}(\mathcal{F}) + 3\sqrt{\frac{\log\left(\frac{2}{\delta}\right)}{m}} \right) \geq 1 - \delta.$$

We will later see develop measures that either bound the Rademacher complexity directly, or act as a heuristic for the generalization capacity of neural networks.

3 Stiffness

3.1 Intuition for Stiffness

Stiffness was an idea introduced in [3] with the intention to explain network generalization, uncover semantically meaningful groups of data points, explore the effects of learning rates on the function learned by the network, and formalise a notion of dynamical critical length. The definitions of stiffness that we will soon introduce will be shown to relate directly to network generalization, will be sensitive to the semantically meaningful content of inputs, and capture information about the loss landscape. The idea is the following, consider two data points (x_1, y_1) and (x_2, y_2). Compute the gradient of the loss function with respect to the parameters as these points as

$$\bar{g}_1 = \nabla_{\mathbf{w}} l(h_{\mathbf{w}}(x_1), y_1), \text{ and } \bar{g}_2 = \nabla_{\mathbf{w}} l(h_{\mathbf{w}}(x_2), y_2).$$

Now we want to consider how the loss on the data points changes as we move the parameter in the direction of $-\bar{g}_1$. So let,

$$\Delta l_1 = l(h_{\mathbf{w}-\epsilon\bar{g}_1}(x_1), y_1) - l(h_{\mathbf{w}}, y_1) = -\epsilon\bar{g}_1 \cdot \bar{g}_1 + O\left(\epsilon^2\right),$$

and

$$\Delta l_2 = -\epsilon\bar{g}_1 \cdot \bar{g}_2 + O\left(\epsilon^2\right).$$

By construction as $\epsilon \to 0$ it follows that $\Delta l_1 < 0$, however, stiffness concerns itself with the behaviour of Δl_1 as $\epsilon \to 0$. We define a positive stiffness to mean that $\Delta l_2 > 0$ in the limit, which intuitively means that in the parameter update of SGD, these two data points support each other. The degree to which they support each other is directly related to the quantity $\bar{g}_1 \cdot \bar{g}_2$.

3.2 Sign and Cosine Stiffness

We now formalise the intuition of the previous section with the following definitions.

Definition 3.1. *The sign stiffness is defined to be*

$$S_{\text{sign}}(h_{\mathbf{w}}) = \mathbb{E}_{(x_1,y_1),(x_2,y_2)\sim\mathcal{D}}\left(\text{sign}\left(\bar{g}_1 \cdot \bar{g}_2\right)\right).$$

Definition 3.2. *Define the cosine stiffness to be*

$$S_{\cos}\left(h_{\mathbf{w}}\right) = \mathbb{E}_{(x_1,y_1),(x_2,y_2)\sim\mathcal{D}}\left(\cos\left(\bar{g}_1 \cdot \bar{g}_2\right)\right),$$

where

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

Remark 3.3. *Note the stiffness is dependent on the underlying distributions $\mathcal{D}$.*

3.3 Class Membership Stiffness

Empirically, we see that sign stiffness is more suitable for identifying the relationship between data points between classes, whereas cosine stiffness better highlights the relationship between data points within classes. As stiffness as defined is a theoretical quantity we must develop a proxy. Recall that in the machine learning paradigm, we have our training set and a held-out validation set that is used to evaluate the trained model. This leads to three different methods of evaluating stiffness. We can either evaluate when both points are taken from the train set, when both points are taken from the validation set, or when one point is taken from each set. Suppose that our set has semantic classes $c_1, \ldots, c_n$.

Definition 3.4. *Define the class stiffness matrix, C, to be the matrix with entries*

$$[C]_{ij} = \mathbb{E}_{X_1\in c_i, x_2\in c_j, x_1\neq X_2}(S((x_1, y_1), (x_2, y_2))).$$

The entries of this matrix can be interpreted 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 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 3.5. *For a set of data points X with n semantic classes summarize the between-class stiffness by*

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

Definition 3.6. *For a set of data points X with n semantic classes summarize the within-class stiffness by*

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

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.

3.4 Stiffness and Distance

We can use stiffness to develop a notion of distance. Intuitively, data points that support each other strongly through SGD should be related in some semantic classes. Therefore, one could imagine that they are close in some space. Defining the distance metric

$$d(x_1, x_2) = 1 - \frac{\bar{g}_1 \cdot \bar{g}_2}{|\bar{g}_1||\bar{g}_2|}$$

tries to encapsulate this intuition. Note that at a distance of zero, the stiffness is one.

4 Neural Tangent Kernel

Tangent kernels are used to understand what function the neural networks are learning. To discern the features captured by the network we 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 [4] this intuition is formalized and then also applied to give of neural network complexity.

4.1 Tangent Features and Kernel

Consider the class of scalar functions

$$\mathcal{H} = \left\{ h_{\mathbf{w}} : \mathcal{X} \to \mathbb{R} : \mathbf{w} \in \mathbb{R}^d \right\}.$$

Definition 4.1. *The tangent features of the scalar function $h_{\mathbf{w}} : \mathcal{X} \to \mathbb{R}$ is the gradient of the function with respect to the parameters,*

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

With the corresponding tangent kernel given by

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

One can quantify the changes in the output of the function as a result of perturbations of the weights using these concepts as,

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

To characterise the prominent directions in parameter space as a result of the training we can look at the eigenvalue decomposition of the covariance matrix

$$\Sigma_{\mathbf{w}} = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left(\Phi_{\mathbf{w}}(x) \Phi_{\mathbf{w}}(x)^\top \right)$$

where $\mathcal{D}$ is the distribution of the input. That is, consider

$$\Sigma_{\mathbf{w}} = \sum_{j=1}^{d} \lambda_{\mathbf{w}j} v_{\mathbf{w}j} v_{\mathbf{w}j}^\top.$$

For each data point (x_i, y_i) in the the training set, and the corresponding outputs $h_{\mathbf{w}}(x_i)$ the gradient descent update on the weights is given by

$$\delta \mathbf{w}_{\mathrm{GD}} = -\eta \nabla_{\mathbf{w}} l(h_{\mathbf{w}}(x_i), y_i)$$

for a loss function l. The resulting function updates can be linearly approximated by

$$\delta h_{\mathrm{GD}}(x_i) = \langle \delta \mathbf{w}_{\mathrm{GD}}, \Phi_{\mathbf{w}}(x_i) \rangle,$$

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

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

Lemma 4.2. *The function updates decompose as $\delta f_{GD}(x) = \sum_{j=1}^{d} \delta f_j u_{\mathbf{w}j}(x)$ with*

$$\delta h_j = -\eta \lambda_{\mathbf{w}j} (\mathbf{u}_{\mathbf{w}j})^\top (\nabla_{h_{\mathbf{w}}} l),$$

where $\mathbf{u}_{\mathbf{w}j} = (u_{\mathbf{w}j}(x_1) \dots u_{\mathbf{w}j}(x_m))^\top \in \mathbb{R}^n$ and $\nabla_{h_{\mathbf{w}}}$ is the gradient with respect to sample outputs.

4.2 Application as a Complexity Measure

Let us consider the set of scalar functions

$$\mathcal{F} = \left\{ h_{\mathbf{w}}(x) = \langle \mathbf{w}, \Phi(x) \rangle : \tilde{\lesssim} \in \mathbb{R}^d \right\}.$$

When given m inputs, the tangent features $\Phi(x_i) \in \mathbb{R}^d$ yield a feature matrix $\Phi \in \mathbb{R}^{m \times d}$. Recall that the Rademacher complexity depends on the size of the class $\mathcal{F}$. A common method for controlling the size of our set of functions is to bound the norm of the weight vector. However, the implicit bias induced by the training algorithm identified by the tangent kernel can also be used.

Definition 4.3. *Let $A \in \mathbb{R}^{d \times d}$ be an invertible matrix. The vector norm of $\mathbf{w} \in \mathbb{R}^d$ induced by A is given by $\|\mathbf{w}\|_A := \sqrt{\mathbf{w}^\top (AA^\top)\, \mathbf{w}}$.*

Proposition 4.4. *For the restricted class of functions $\mathcal{F}^A_{M_A} = \{h_{\mathbf{w}} : x \mapsto \langle \mathbf{w}, \Phi(x) \rangle : \|\mathbf{w}\|_A \leq M_A\}$ and sample set S, we have that*

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

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

As the training algorithm is iterative we can consider the decomposition $h_{\mathbf{w}} = \sum_t \delta h_{\mathbf{w}_t}$ where $\delta h_{\mathbf{w}_t} = \langle \delta \mathbf{w}_t, \Phi(\mathbf{x}) \rangle$. Applying a local constraint on the parameter updates we can considering the restricted class of functions

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

Theorem 4.5. *Given any sequences $\mathbf{A}$ and $\mathbf{m}$ of invertible matrices $A_t \in \mathbb{R}^{d \times d}$ and positive numbers $m_t > 0$ the following bound holds*

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

Remark 4.6. *Using the linear re-parameterisation $\mathbf{w} \mapsto A^\top \mathbf{w}$ and $\Phi \mapsto A^{-1}\Phi$ we can consider the restricted function class*

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

In this case a similar bound holds,

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

These bounds are for fixed sequences of feature maps. However, the results can be extrapolated to the non-deterministic setting for which [4] proposes the following heuristic measure for the complexity of neural networks,

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

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.

5 Algorithmic Stability

The work of [5] focuses on the learning algorithm to understand network generalization. A general update rule of GD takes the form

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{m}\sum_{i=1}^{m} g_i(\mathbf{w}_t),$$

where where $g_i(\mathbf{w}_t)$ is the gradient of the loss function at example i evaluated at the parameter $\mathbf{w}_t$, and η is the learning rate. Our intuition says that the average gradient $g(\mathbf{w})$ is strong in directions where per-example gradients are "similar" and parameters that correspond to these stronger directions should change more. When per-example gradients reinforce each other they are said to be coherent. A learning process is stable if changing an example in the training set doesn't affect the learned model in drastic ways. Strong directions in the average gradient are stable, as multiple examples reinforce each other. The stability of the learning process relates 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 require a quantitative notion of coherence.

5.1 Motivating Example

Consider the linear model $\hat{y} = \mathbf{w} \cdot x = \sum_{i=1}^{6} w_i x_i$ which is fitted using the square loss $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$. 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, as with the first data set the last entry of the x_i correlates with the label y_i. However, with the other dataset, there is no correlation between any one entry of the x_i with the labels y_i and hence appears "random". 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} entry is reinforced by each example, on the other hand, in the random dataset this component is not reinforced. 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. It is reassuring then that the gradients of the data points reinforce the learnable feature so that the model can learn this.

5.2 Coherence

Suppose the network has d trainable parameters. Let $l_z(\mathbf{w})$ be the loss of example $z \sim \mathcal{D}$ for weight vector $\mathbf{w} \in \mathbb{R}^d$ and let $l(\mathbf{w}) = \mathbb{E}_{z\sim\mathcal{D}}(l_z(\mathbf{w}))$ be the expected loss. Denote the gradient of the loss at example z by $g_z := (\nabla l_z)(\mathbf{w})$ and the average gradient over all examples by $g := (\nabla l)(\mathbf{w})$. Throughout η will denote the learning rate. The learning problem is trying to minimize the expected loss through a gradient descent algorithm where small descent steps $h = -\eta g$ are taken. Using Taylor approximations to get that

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

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. We could make gradient descent more stable by combining per-example gradients using robust mean estimation techniques (i.e. use the median rather than sample mean) to eliminate weak directions in the gradients.

2. We could use l^2 regularization as a means to reduce movement in weak directions.

3. As training progresses the gradients of fitted examples become negligible. Therefore, we could consider early stopping to 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 a step h_z down its gradient g_z at a parameter point w is given by,

$$l_z(\mathbf{w} + h_z) - l_z(\mathbf{w}).$$

Using Taylor approximations and taking expectations over z gives

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

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 the 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 5.1. *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 5.2. *Typically we do not have access to the underlying distribution of the samples. Therefore, we let α_m denote the coherence of a sample S where $|S| = m$.*

Example 5.3. *Let S be a sample of size m, whose gradients g_i ($1 \leq i \leq m$) are pairwise orthogonal. Then,*

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

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 $\alpha_m^\perp$.

The quantity $\alpha_m/\alpha_m^\perp$ can be interpreted as the average number of samples that a particular gradient helps fit.

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. In the under-parameterized setting $d \ll m$ with a sample of $k \gg 1$ copies of orthogonal gradients in d-dimensional space (i.e. $m = kd \gg d$). We have that $\alpha_m = \frac{1}{d}$ and $\alpha_m/\alpha_m^\perp = k$. Hence, each example helps k other examples in the sample as expected.

5.3 Application to the Generalization Gap

A dataset-independent argument for stability could be 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. In [5] a dataset-dependent argument for stability is given which uses coherence, these ideas are then applied to explain generalization. The expected generalization gap is the expected difference between training and test loss over samples of size m from $\mathcal{D}$, which we will denote $\mathrm{gap}(\mathcal{D}, m)$.

Theorem 5.4. *If stochastic gradient descent is run for T steps on a training set consisting of m examples drawn from distribution $\mathcal{D}$, then,*

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

where

- $\alpha(\mathbf{w})$ *denotes coherence at point w in parameter space,*
- $\mathbf{w}_t$ *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 5.5.

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

5.4 Coherence of Fully Connected Neural Networks

When equipped with a dataset and a neural network we can calculate an empirical value for coherence by replacing the expectations in the definition of coherence with sample means. However, when using SGD there are several issues that inhibit the ability to compute the coherence of a network during 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 access to per-example gradients was granted.

We now propose a method to approximate $\alpha_m / \alpha_m^{\perp}$ from batched data.

Theorem 5.6. *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 $w = \frac{1}{k} \sum_{i=1}^{k} v_i$. Then,*

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

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

Using Theorem 5.6 we can define a tractable method for approximating coherence by using gradients computed at the batch level rather than the per-example level.

1. Compute the gradients for each batch.

2. Compute coherence of the $\frac{m}{k}$ batch in the usual way, $\alpha(\mathcal{W})$.

3. Re-arrange the identity of Theorem 5.6,

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

   and use this to get a value for $\alpha(\mathcal{V})$.

4. Compute $\alpha_m / \alpha_m^{\perp}$ as $m\alpha(\mathcal{V})$.

5.5 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.

5.5.1 Winsorized Gradient Descent

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

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

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 the median of means algorithm is used:

1. Divide the sample into k groups,

2. Compute sample mean of each group,

3. Return the median of these k means.

6 Geometric Complexity

Rademacher complexity is a measure of complexity that focuses on the entire hypothesis space rather than focusing on a function. Complexity notions such as VC dimension and parameter counting also 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 [6] focuses on capturing function complexity over proposed datasets through geometrical arguments with the following proposed complexity measure.

Definition 6.1. *Let $h_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}^k$ be a neural network parameterised by $\mathbf{w}$. Let $g_{\mathbf{w}}(x) = \alpha(f_{\mathbf{w}}(x))$ where α is the last layer activation, and f_θ is its logit network. The Geometric Complexity (GC) of the network over a dataset S is the discrete Dirichlet energy of its logit network,*

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

where $\|\nabla_x f_{\mathbf{w}}(x)\|_F$ is the Frobenius norm of the network Jacobian.

6.1 Application to ReLU Networks

Let $h_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}^k$ be a ReLU network. As $h_{\mathbf{w}}$ parameterises piece-wise linear functions, the domain can be partitioned by subsets $X_i \subset \mathbb{R}^d$ such that $f_{\mathbf{w}}$ is an affine map $A_i x + b_i$. Let $S_i = S \cap X_i$, then for every $x \in X_i$ it follows that $\|\nabla_x f_{\mathbf{w}}(x)\|_F^2 = \|A_i\|_F^2$ so that

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

where $m_i = |D_i|$. Therefore, a stratified batch $B \subset S$ 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.

6.2 Relationship to Lipschitz Smoothness

We can measure the smoothness of a function $h : \mathbb{R}^d \to \mathbb{R}^k$ on a subset $X \subset \mathbb{R}^d$ by its Lipschitz constant h_L which is defined to be the smallest positive real number such that

$$\|h(x_1) - h(x_2)\| \leq h_L \|x_1 - x_2\|$$

for all $x_1, x_2 \in X$. Using this we can bound the GC using Lipschitz complexity as

$$\langle h, S\rangle_G = \frac{1}{|S|}\sum_{x\in S}\|\nabla_x h(x)\|_F^2 \leq \frac{1}{|S|}\sum_{x\in S}\min(k,d)\,\|\nabla_x h(x)\|_{\text{op}}^2 \leq \min(k,d)h_L^2. \tag{3}$$

6.3 Encapsulating Neural Network Phenomena

Experimentally it can be shown that common training heuristics keep the GC low, which reinforces its application as a complexity measure.

- Parameter initialization: It is found that as the number of layers increases the GC at initialization can be brought to zero. This correlates with the theoretical result that a neural network will converge to a constant function at random initialization as the number of layers increases.
- Explicit Regularization:

– L2 Regularization: The L2 norm penalties on the weight matrix coincide with an explicit GC regularization in the case of linear models. For deep ReLU networks the matrix output y for an input x is given by

$$y = P_l W_l \dots P_1 W_1 x + c$$

where c is a constant, the P_i's are diagonal matrices with entries $\{0, 1\}$ and the W_i's are weight matrices. The derivative of x is just the product matrix

$$P_l W_l \dots P_1 W_1.$$

So the GC is the Frobenius norm of this matrix. An L2 penalty penalizes large values

$$\|W_l\|_F^2 + \dots + \|W_1\|_F^2$$

which in turn encourages small

$$P_l W_l \dots P_1 W_1$$

and hence results in lower GC values.

– Lipschitz Regularization via Spectral Norm Regularization: The spectral norm of the product of the network's weight matrices is an upper bound to the Lipschitz constant of the model. Therefore, using (3) we see that any form of Lipschitz regularization also constrains the GC.

– Noise Regularization: It has been shown that the introduction of noise during training via SGD exerts a regularization pressure on $\|\nabla_{\mathbf{w}} h_{\mathbf{w}}\|$. Now it can also be shown that pressure on $\|\nabla_{\mathbf{w}} h_{\mathbf{w}}\|$ translates to pressure $\|\nabla_x h_{\mathbf{w}}\|$, therefore using label noise in SGD constrains GC.

– Flatness Regularization: This involves a penalty term of the form $\|\nabla_{\mathbf{w}} l_B\|^2$ on the loss, where l_B is the loss evaluated on a batch B. This penalizes learning trajectories that follow steep slopes. Intuitively, it follows that flatness regularization also constrains GC as learning is encouraged to follow shallower slopes in the loss landscape. This intuition can be shown theoretically.

References

[1] Maria-Florina Balcan. *Rademacher Complexity*. 2011.

[2] Lei Wu, Chao Ma, and Weinan E. "How SGD Selects the Global Minima in Over-parameterized Learning: A Dynamical Stability Perspective". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018.

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

[4] 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).

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

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