Lecture 3: Optimization and Generalization Challenges in Deep Learning

Lecturer: Aleksander Mądry

Scribes: Nadiia Chepurko, Irina Degtiar, Nishanth Dikkala Pritish Kamath, Molei Liu, Devendra Shelar
(Revised by Andrew Ilyas and Dimitris Tsipras)

1 Issues with Applying Gradient Descent Methods in Deep Learning Setting

Last time we talked about different types of gradient descent methods, including the ones used in deep learning. Now, it is time to discuss several issues that may arise when we use first-order methods to train deep neural networks.

Just to build some intuition, consider a fully connected neural network (with a softmax layer on top). Let X^0 denote the data matrix, i.e., the m-by-n matrix whose each of n columns corresponds to one of the data points. One can view this neural network as a sequence of layer-wise transformations of that data matrix. That is, the "data matrix" after passing through i-th layer of that network is given by

$$X^i = \sigma(W^i X^{i-1} + B^i),$$

where W^i is the weight matrix of the i-th layer, B^i is the matrix of added (and learnable) biases, and X^{i-1} is the "data matrix" after passing the previous layers. Here, σ denotes the non-linear activations applied coordinate-wise. A typical choice of the non-linear activation σ is the ReLU activation function given as $\sigma(a) = \max\{a, 0\}$.

Now, to get a feel for how the information about the gradient updates propagate during training, let us ignore the bias matrices and focus on some specific data point j and its representation X_j^1 after passing through the first layer. In that case, the gradient of the change $\nabla_{X_j^1} X_j^\ell$ of the representation X_j^ℓ of that data point after the final (ℓ-th) layer with respect to change in X_j^1 is given by:

$$\nabla_{X_j^1} X_j^\ell = D^\ell W^\ell \cdots D^2 W^2, \tag{1}$$

where each D^i is a diagonal matrix with $D^i_{kk} = \mathbf{1}\{(X_j^i)_k \geq 0\}$, where $(X_j^i)_k$ is the k-th coordinate of the representation X_j^i. (Note that, strictly speaking, in training we can't directly modify the representation vector X_j^1—we can do it only indirectly via modifying W^1.)

Vanishing/exploding gradients. The formula (1) should already hint that training neural networks can be tricky. In particular, the gradient signal can exhibit bad behavior.

For example, suppose each weight matrix W^i is a random orthogonal matrix (i.e. a rotation)—then, due to the fact that each matrix D^i is "killing" half of the dimensions on average, the gradient $\nabla_{X_j^1} X_j^\ell$ would be of magnitude $\exp(-\ell)$ (if not 0), decreasing exponentially with the number of layers in the network. This would be an instance of a broader phenomenon known as the *vanishing gradient* effect.

Another illustration of what can go wrong is the situation if the entries of each W^i are larger than 1. Then, the "signal" in each layer could build up uncontrollably, making the gradient $\nabla_{X_j^1} X_j^\ell$ be of magnitude $\exp(\ell)$. This corresponds to the *exploding gradients* effect.[1]

Issues at initialization. The above discussion should already have hinted that one needs to be careful about how to initialize deep neural networks. Indeed, issues with initialization extend to far more complex cases than this, and there have been a multitude of recent studies showing that initialization can actually cause large issues for network training (see, for example [HR18]).

[1] Note, however, that both these two examples are very cartoonish/unrealistic. An interesting question is though: why do we observe vanishing/exploding gradients in practice?

1.1 Proposed solutions to issues in DNN training

The above problems were plaguing early (modern) deep neural network architectures. Fortunately, over time, we managed to come up with architectural changes (as well as corresponding initialization schemes) that improved the situation considerably. We briefly describe the two most important ones below.

Batch normalization [IS15] Batch normalization was originally proposed to address the hypothesized problem of *internal covariate shift*. Specifically, internal covariate shift refers to the fact that the "data matrix" X^i that the optimization/learning process at the corresponding layer i has as an input constantly changes due to updates to the lower layers of the network. Ioffe and Szegedy viewed this effect as very detrimental to network training and proposed to alleviate it via so-called *batch normalization* layers.

The goal of batch normalization is to ensure that the distribution of activation in each data matrix is more stable. This is achieved by normalizing the distributions to have the same first and second moments for the distributions in each layer. In particular, consider a size-k minibatch of inputs to layer i (i.e., the subset of k columns of the matrix X^i), which we denote as $\{x^{(1)}, \ldots, x^{(k)}\}$, and let $\{y^{(1)}, \ldots, y^{(k)}\}$ be the corresponding vectors of activations after leaving the layer i. Batch normalization involves adding a new layer after layer i whose goal is to reparametrize activations so as they become $\{\hat{y}^{(1)}, \ldots, \hat{y}^{(k)}\}$, where each coordinate $\hat{y}_r^{(b)}$ of $\hat{y}^{(b)}$ is given by:

$$\hat{y}_r^{(b)} = \frac{y_r^{(b)} - \frac{1}{k}\sum_{s=1}^{k} y_r^{(s)}}{\sqrt{\frac{1}{k}\sum_{s=1}^{k}\left(y_r^{(s)} - \frac{1}{k}\sum_{l=1}^{k} y_r^{(l)}\right)^2}}.$$

Intuitively, this transformation whitens the "distribution" $\{y_r^{(1)}, \ldots, y_r^{(k)}\}$ so as it has zero mean and unit variance. (In practice, one sometimes allows this whitened mean and variance to be a (trainable) parameter as well.)

It turns out that adding such batch normalizing layers after each original layer—and, crucially, allowing the training to back-propagate through these layers—leads to dramatic improvement of training reliability. In particular, vanishing and exploding gradients become much less of an issue. (And generalization tends to be somewhat improved as well.)

But, does it mean that the internal covariance shift was indeed the root of the problem here? Recent work [STIM18] has cast doubt that it is the case. Specifically, it demonstrated that for a number of natural ways of measuring internal covariate shifts, batch norm does not seem to have visible effect on reducing that measure. Still, it has an important beneficial effect: it makes the loss landscape be much more navigable for first order methods.

Residual Neural Networks [HZRS16]. Residual networks were proposed as a way to try and address a number of issues related to training networks with large depth (think 30+ layers). An intriguing problem observed with networks of such a large depth is that addition of layers after a certain point start to degrade the performance (and this is not due to overfitting as even the training loss degrades). The authors who proposed residual networks hypothesized that this is potentially due to the signal "learnt" by the lower layers getting lost by the time we reach the higher layers. To overcome this, they propose that each layer learns a residual function instead. That is, to have our activation matrices X^i evolve as

$$X^i = \sigma(W^i X^{i-1} + B^i) + X^{i-1}.$$

That is, instead of a set of stacked layers learning a function $H(x)$ the authors propose to let them learn the "residual" function $F(x) = H(x) - x$ instead. (This assumes that the input x and output $H(x)$ are of the same dimension which roughly holds for deep networks.) Architecturally, this corresponds to adding "shortcut" (referred to as residual) connections bypassing each one of the layers. The resulting architectures indeed perform significantly better in training, making ResNets (the architectures implementing residual connections) a very popular class of models.

2 Generalization in Deep Learning

Deep neural nets (DNNs) tend to generalize well, but we don't yet have a good understanding as to why. Let's recall the general setup for supervised learning: we are sampling m samples from a distribution $\mathcal{D}$, $S_m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, and we have a learning algorithm A that given S_m tries to find the best classifier f_θ out of the class $\mathcal{F}$ to fit S_m, e.g:

$$\arg\min_\theta \; L_{S_m}(\theta)$$

$$L_{S_m}(\theta) = \frac{1}{m} \sum_i loss(f_\theta(x_i), y_i)$$

where $L_{S_m}(\theta)$ is the *empirical* loss, $f_\theta(x_i)$ the prediction, and y_i the true label. In practice, however, we want to minimize the *population* loss, where we sample (x, y) from the true distribution $\mathcal{D}$:

$$L_{\mathcal{D}}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[loss(f_\theta(x), y)].$$

The difference between the empirical and population loss is called the *generalization gap*. The generalization gap is therefore the difference in loss between optimizing over the entire true data distribution and optimizing over our sampled data, for a training algorithm A:

$$\Delta_m(A) = L_{\mathcal{D}}(\theta) - L_{S_m}(\theta).$$

Ideally, by ensuring a small sample loss and a small generalization gap we can obtain a small population loss, which is the ultimate goal. The next sections outline several approaches to proving generalization bounds.

3 Rademacher Complexity

Recall that in Lecture 3 we defined Rademacher complexity as:

$$\mathscr{R}_m(\mathcal{F}) = \mathop{\mathbb{E}}_{S_m\sim\mathcal{D}}[\hat{\mathscr{R}}_S(\mathcal{F})]$$

where $\mathcal{F}$ is a family of classifiers, with $f \in \mathcal{F}$, and $\hat{\mathscr{R}}_S(\mathcal{F})$ is the empirical Rademacher complexity:

$$\hat{\mathscr{R}}_S(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f\in\mathcal{F}} \frac{1}{m} \sum_i \sigma_i f_\theta(x_i) \right],$$

where σ is a random sign vector, and the maximization over θ is performed with respect to $f \in \mathcal{F}$. Conceptually, it captures how well our family of classifiers fits random noise when parametrized by θ. If our classifier does well at fitting random noise, then we probably need to see a lot of samples before we can actually trust it. Formally, the generalization gap of such an algorithm A can be upper bounded by the Rademacher complexity:

$$\Delta_m(A) \leq 2\mathscr{R}_m(\mathcal{F}) + 3\sqrt{\frac{\log\frac{1}{\delta}}{m}},$$

with probability $1 - \delta$.

4 PAC-Bayesian

Another approach we might consider using to prove generalization for DNNs uses PAC-Bayesian bounds. In particular, the PAC-Bayes theorem bounds the expected error rate of a classifier chosen from a distribution Q in terms of the KL-divergence from some a priori fixed distribution P. If the volume of equally good solutions is large and not too far from the mass of P, we will obtain a nonvacuous bound.

We assume that we have some prior distribution P over the initial set of θ before seeing the data (e.g., the initialization parameters for a neural net). In this Bayesian approach, learning using algorithm A uses data to modify the prior distribution to get the posterior distribution Q, from which the classifier θ is sampled. The corresponding expected loss is: $\mathop{\mathbb{E}}_{\theta\sim Q}[L_D(\theta)]$. The generalization bounds we get are instance-specific (they depend on the i.i.d set S of size m). It can be proven that:

$$\Delta_m(A) = \mathop{\mathbb{E}}_{\theta\sim Q}[L_D(\theta)] - \mathop{\mathbb{E}}_{\theta\sim Q}[L_{S_m}(\theta)] \leq \sqrt{\frac{D_{KL}(P,Q) + ln\frac{m}{\delta}}{2(m-1)}}.$$

This allows upperbounds on generalization error (specifically, upperbounds on number of samples that guarantee such an upperbound). The major component in this bound is the KL divergence between our prior and posterior distributions. KL divergence is defined as:

$$D_{KL}(P,Q) = \int \log\frac{q(x)}{p(x)} q(x)dx$$

where q and p are densities for Q and P. A smaller KL divergence implies better generalization. So if the initial parameters didn't change much between the prior and posterior, the algorithm generalizes well. Thus, in order to minimize the error on the real distribution, we should try to simultaneously minimize the empirical error as well as the KL-divergence between the posterior and the prior [Aroa].

5 Meta-theorem of generalization

One way to show that a model is generalizable is to show that it does not have too many degrees of freedom. The meta theorem states that a generalization gap is bounded by the square root of (effective) degrees of freedom of a model, $\mathscr{N}(\theta)$, over m:

$$\Delta_m(A) \leq \sqrt{\frac{\mathscr{N}(\theta)}{m}}.$$

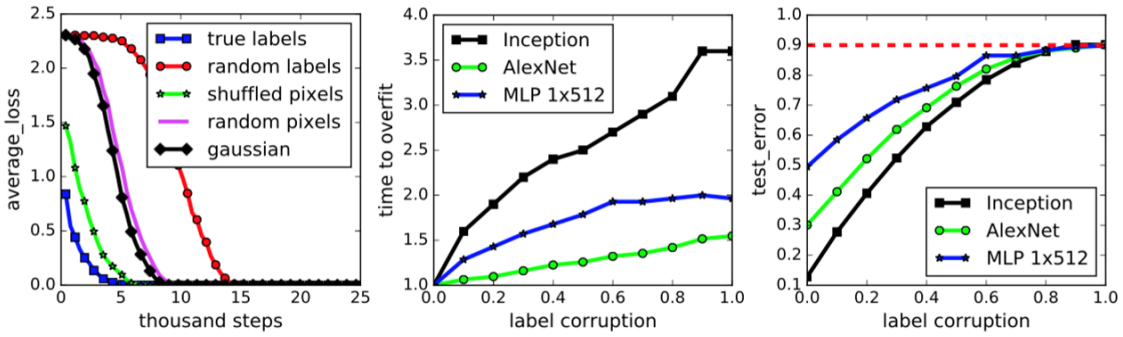
One example is Massart's lemma for the empirical Rademacher complexity, $\hat{\mathscr{R}}_m$, with finite $\mathcal{F}$:

$$\hat{\mathscr{R}}_{m,S}(\mathcal{F}) \propto \sqrt{\frac{log\mathcal{F}}{m}}.$$

However, in the case of deep learning, the number of model parameters is usually larger than the sample size, which provides a useless generalization gap bound of greater than 1.

This phenomenon was also demonstrated in the work of Zhang et al. [ZBH+16]. They train a DNN on standard datasets and on the same datasets with random labels[2]. In both cases, the *same* classifier and training algorithm performed equally well with enough iterations, achieving 0% error on the training set (Figure 1). As the amount of label corruption was varied smoothly, generalization deteriorated with more noise. These findings imply that we cannot hope to prove generalization bounds for DNNs similar to those of Lecture 3. Traditional generalization theory (VC dimentionality, Rademacher complexity, and uniform stability) cannot distinguish between models that generalize well and those that don't, and explicit regularization such as dropout, data augmentation, and weight decay cannot sufficiently control generalization error. Since there exists a distribution where the generalization gap is greater than 1, there do not exist non-vacuous generalization bounds that are independent of the data distribution.

[2]The inputs x remain the same, but the labels y are chosen randomly, having no relationship to x. This is a hopeless task for any classifier.



(a) learning curves (b) convergence slowdown (c) generalization error growth

Figure 1: DNN classifiers are expressive enough to fit random labels in CIFAR10. (a) Under various settings, training loss reached zero after a sufficient number of steps. (b) With more label corruption, convergence time increased by only a small constant factor. (c) With more label corruption, test error (which equals generalization error since there is no training error) increased [ZBH+16].

6 Flatness of Local Minima (Speculative)

Given that the empirical loss function is often non-convex, there may be multiple local minima of similar (near-optimal) values. How can we identify the ones that generalize well? Intuitively, one would expect that flat minima generalize better than sharp minima (Figure 2). Flat minima require less information to describe them (Occam's razor) since there are multiple (almost) equally good points in their neighborhood. Keskar et al. [KMN+16] explain the generalization properties of flat minima as follows: a function increases more rapidly around sharp minima than around flat minima; thus, more precision is needed in describing sharp minima. This makes them more sensitive to deviations in the parameter estimates and could impact the model's ability to generalize. The theory of minimum description length (MDL) states that statistical models that can be described in fewer bits (with lower precision) generalize better. When adding random Gaussian noise $\mathrm{N}(0, \sigma^2)$ to the local minimum θ^*, a solution in a flat minimal will remain at the minimum while a solution in a sharp minimum might jump outside of the minimum. Since we expect the "true" classifier to be robust to small amount of noise in its parameters, we expect flat minima to be more suitable for classification.

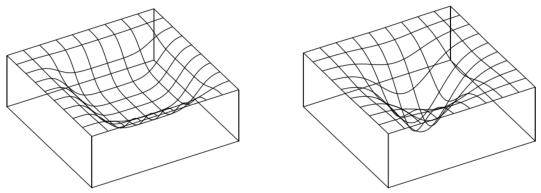


Figure 2: Flat minima v.s. sharp minima.

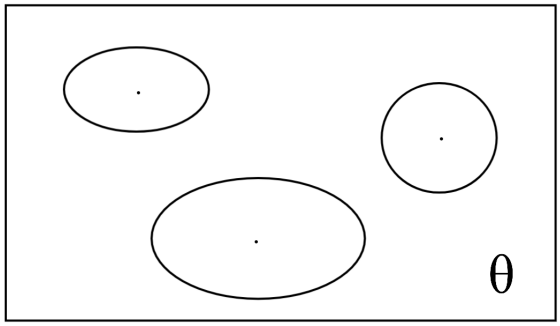


Figure 3: Parameter space

When there are more flat minima in a parameter space, fewer minima "fit" into the space and so an algorithm will have fewer degrees of freedom and will be more likely to converge to one of the mimima (Figure 3). Alternative explanations as to why flat minima generalize better include the Bayesian view of learning [Mac91] and the free Gibbs energy (entropy) explanation [CCS+16].

Evidence exists that stochastic gradient descent (SGD) tends to converge to flat minima more so than other optimization functions [DR17]. The noise due to SGD's stochasticity is therefore often viewed as a feature and not a bug, resulting in regularization.

So far we have not rigorously defined the notion of flatness. There are multiple ways to do so, however most of them cannot be directly related to generalization. Note that we can re-parametrize a DNN by (say) multiplying a layer by a constant and dividing the next layer by the same constant. This transformation has no effect on its generalization performance (it is the same network after all) but can greatly change flatness under multiple definitions of it. We explore this concept further in the next section.

6.1 Sharp minima can generalize for deep nets

As we discussed in the previous section, flat minima tend to generalize better, but Dinh et al. [DPBB17] claim that having sharp minima may not necessarily reduce generalizability for DNNs with certain properties. Specifically, when focusing on deep networks with rectifier units, we can exploit the particular geometry of the parameter space induced by the inherent symmetries that these architectures exhibit to build equivalent models corresponding to arbitrarily sharper minima. Furthermore, if we allow for function reparametrization, the geometry of its parameters can change drastically without affecting its generalization properties [DPBB17].

Let us define a non-negative homogeneity property and see its implications for rectifier activation function. For more details refer to [DPBB17].

Definition 1 *A given function ϕ is non-negative homogeneous if* $\forall(z, \alpha) \in \mathbb{R} \times \mathbb{R}^+, \phi(\alpha z) = \alpha\phi(z)$

The parameter space of our model relates a unit of change in the behavior of the model to the equivalent change in the parameters. This metric is related to non-linear curvature due to model's non-linearity properties. There are possibly a large number of symmetric configurations that result in similar model behavior due to the non-negative homogeneity property of rectifier activation (see 4).

Theorem 2 *The rectifier function $\phi_{rect}(x) = max(x, 0)$ is non-negative homogeneous which implies*

$$\phi_{rect}(x \cdot (\alpha\theta_1)) \cdot \theta_2 = \phi_{rect}(x \cdot \theta_1) \cdot (\alpha\theta_2)$$

This results in the parameters $(\alpha\theta_1, \theta_2)$ being observationally equivalent to $(\theta_1, \alpha\theta_2)$

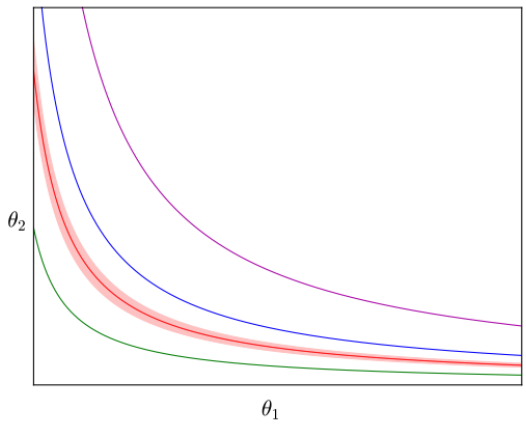


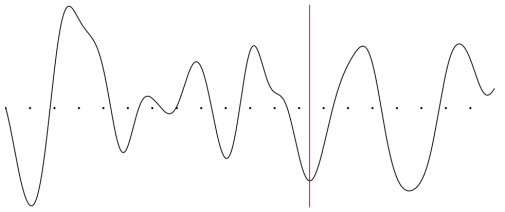
Figure 4: An illustration of the effects of non-negative homogeneity. The graph depicts level curves of the behavior of the loss embedded into the two dimensional parameter space with the axis given by θ_1 and θ_2. Specifically, each line of a given color corresponds to the parameter assignments (θ_1, θ_2) that result in the same prediction function.

In what follows, we rely on α-scale transformations that will not affect the generalization, as the behavior of the function is identical.

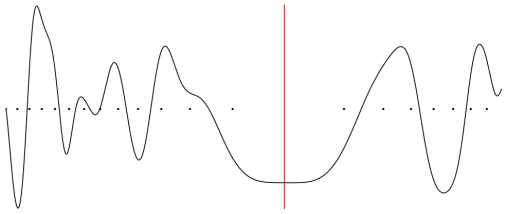
Definition 3 *For a single hidden layer rectifier feed-forward network, the family of alpha-scale transformations is defined as follows:*

$$T_\alpha : (\theta_1, \theta_2) \mapsto (\alpha\theta_1, \alpha^{-1}\theta_2)$$

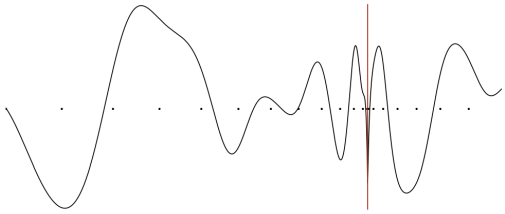
Dinh et al. [DPBB17] exploit the resulting strong non-identifiability to showcase a few shortcomings of some of the definitions of flatness. Although an α-scale transformation does not affect the function represented, it allows us to manipulate the flatness of our minima. For another definition of flatness, the α-scale transformation shows that all minima are equally flat. Similarly, if we are allowed to change the parametrization of some function f, we can obtain arbitrarily different geometries without affecting how the function evaluates on unseen data. The same holds for re-parametrization of the input space. This implies that the correlation between the geometry of the parameter space (and hence the error surface) and the behavior of a given function is meaningless if not preconditioned on the specific parametrization of the model.



(a) Loss function with default parametrization



(b) Loss function with reparametrization



(c) Loss function with another reparametrization

Figure 5: A one-dimensional example on how much the geometry of the loss function depends on the parameter space chosen. The x-axis is the parameter value and the y-axis is the loss. The points correspond to a regular grid in the default parametrization. In the default parametrization, all minima have roughly the same curvature but with a careful choice of re-parametrization, it is possible to turn a minimum significantly flatter or sharper than the others.

7 Compression

As we already discussed, classifiers with a large number of parameters have many degrees of freedom. This prevents us from proving meaningful generalization bounds for them. A natural idea is to compress the function parameterized by θ, f_θ to $\hat{f}_{\hat{\theta}}$, where $\hat{f}_{\hat{\theta}}$ is significantly more restricted in its expressive power. If the compressed classifier $\hat{f}_{\hat{\theta}} \approx_\epsilon f_\theta$ also fits the training data well, we can use its limited expressivity to prove generalization bounds.

This suggests an intriguing approach for obtaining classifiers with provable generalization. First, optimize training error over an *over-parametrized* family of classifiers that is easy to train. Then, compress the resulting classifier to an (almost) equivalent one from a significantly more restricted family that we can prove generalization bounds for.

See [Arob] for the explanation of compression and the motivation behind it.

Briefly, DNNs weight matrices can be compressed using a truncated SVD method. The compression algorithm applies a randomized transformation to each layer's matrix that relies on the low stable rank condition at each layer. This compression introduces error in the layer's output, but the vector describing this error is Gaussian-like due to the use of randomness in the compression. Thus, this error gets attenuated by higher layers. The process can be broken down as follows: take higher level matrices for trained DNNs, perform SVD, compress each layer by zero-ing out singular values less than some threshold $t|A|$ where A is our original matrix. A simple computation shows that the number of

remaining singular values is at most the stable rank divided by t^2. The truncation introduces error in the layer's computation, which gets propagated through the higher layers and magnified at most by the Lipschitz constant. We want to make this propagated error small, which can be done by making t inversely proportional to the Lipschitz constant. The next sections describe how compression can be used to obtain the generalizability of classification models.

7.1 Linear Regression

We will first consider the case of linear classifiers. Let $c \in \mathbf{R}^d$, $\|c\| = 1$ be the classifier of (x, y) in training dataset S, where $x \in \mathbf{R}^d$, $\|x\| = 1$, and $y \in \{-1, 1\}$. The prediction for datapoint x is $\text{sign}(c^T x)$. Assume for all $x \in S$, we have $|c^T x| > \gamma$ (the confidence margin). We compress $c = [c_1, c_2, \cdots, c_d]$ to a sparse vector $\hat{c} = [\hat{c}_1, \hat{c}_2, \cdots, \hat{c}_d]$ in the following way:

$$\hat{c}_i = \begin{cases} \frac{\gamma^2}{8c_i} & \text{w.p. } \min\left\{\frac{8c_i^2}{\gamma^2}, 1\right\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

This is possible since the classification margin makes the prediction noise resistant. This compression/discretization makes $\mathrm{E}|supp(\hat{c})| = O(1/\gamma^2)$, where $supp(\cdot)$ denotes the support of a vector, i.e. the number of non-zero coordinates:

$$\begin{aligned} \mathrm{E}|supp(\hat{c})| &= \mathrm{E}\sum_{i=1}^{d} \mathbf{1}_{\hat{c}_i \neq 0} = \sum_{i=1}^{d} \mathrm{P}(\hat{c}_i \neq 0) \\ &\leq \sum_{i=1}^{d} \frac{8c_i^2}{\gamma^2} = \frac{8}{\gamma^2}\|c\|^2 = O(1/\gamma^2). \end{aligned} \tag{3}$$

Also, we have that

$$\mathrm{E}\hat{c}_i = c_i \mathbf{1}_{8c_i^2/\gamma^2 > 1} + \frac{\gamma^2}{8c_i} \cdot \frac{8c_i^2}{\gamma^2} \mathbf{1}_{8c_i^2/\gamma^2 \leq 1} = c_i \tag{4}$$

and thus, for fixed x: $\mathrm{E}[\hat{c}^T x] = c^T x$. We can also bound $\mathrm{Var}(\hat{c}^T x)$:

$$\begin{aligned} \mathrm{Var}(\hat{c}^T x) &= \sum_{i=1}^{n} x_i^2 \mathrm{Var}(\hat{c}_i) \\ &= \sum_{i=1}^{n} x_i^2 [0 \cdot \mathbf{1}_{8c_i^2/\gamma^2 > 1} + \mathrm{Var}(\hat{c}_i) \mathbf{1}_{8c_i^2/\gamma^2 \leq 1}] \\ &\leq \sum_{i=1}^{n} x_i^2 \left(\frac{\gamma^2}{8c_i}\right)^2 \cdot \frac{8c_i^2}{\gamma^2} = \frac{\gamma^2}{8}\|x\|^2 = \frac{\gamma^2}{8} \end{aligned} \tag{5}$$

and obtain generalization bounds for the compressed classifier. Combine this with the meta theorem:

$$\Delta_m(A) \leq \sqrt{\frac{\mathscr{N}(\theta)}{m}},$$

where the degree of freedom $\mathscr{N}(\theta) \leq \mathrm{E}|supp(\hat{c})| = O(1/\gamma^2)$. We have that $\Delta_m(A) = O(1/\sqrt{m})$

7.2 Compressing matrices

We can apply similar ideas to compress matrices. One approach uses the stable rank of a matrix M: $\frac{\|M\|_F}{\|M\|}$, where $\|\cdot\|_F$ corresponds to the Frobenius norm and $\|\cdot\|$ to the spectral norm of the matrix.

Recently, Arora et al. [AGNZ18] showed that matrices typically learned in DNN classifiers have few large eigenvalues. Using this observation, in combination with properties of the matrix's sensitivity to

noise, they show that these matrices can easily be compressed. Choosing a small eigenvalue as the threshold, all values below it can be set to 0; the remaining large eigenvalues correspond to the largest signals (Figure 6).

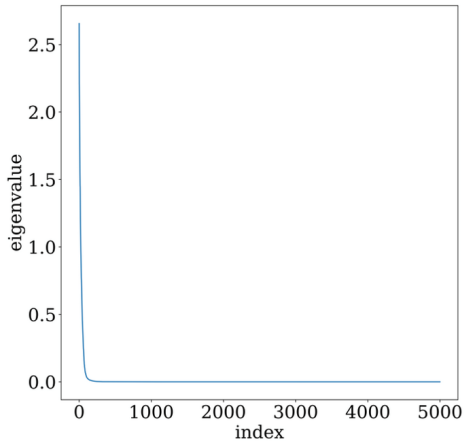


Figure 6: Spectrum of a typical matrix learned through training a deep learning classifier [AGNZ18]. We can observe that only a few eigenvalues are significant.

8 Stability

An alternative way to bound the generalization gap of learning algorithms is through the notion of stability. In their seminal work, Bousquet and Elisseeff [BE02] showed that algorithms that are *uniformly stable* have small expected generalization gaps. For training set $S = \{z_i = (x_i, y_i) : i = 1, 2, \cdots, n\}$, let $R_S[A(S)] = \frac{1}{n}\sum_{i=1}^{n} loss(A(S), z_i)$, $R[A(S)] = \mathrm{E}_{z\sim D} loss(A(S), z)$, and denote the generalization gap as

$$\epsilon_{gen} = \mathrm{E}_{S,A}[R_S[A(S)] - R[A(S)]]. \tag{6}$$

An algorithm $A(\cdot)$ is uniformly stable if changing any single training example does not change the training loss by more than ϵ. That is, if the training set S_1 differs from the training set S by one datapoint:

$$\sup_z \mathrm{E}_A[loss(A(S), z)] - \mathrm{E}_A[loss(A(S_1), z)] \leq \epsilon,$$

where the expectation is over the randomness of the algorithm. The proof is as follows. Denote $S' = \{z'_1, z'_2, \cdots, z'_n\}$, $S^{(i)} = \{z_1, z_2, \cdots, z_{i-1}, z'_i, z_{i+1}, \cdots, z_n\}$, which differs from S only on z_i. Then we have:

$$\begin{aligned}
\mathrm{E}_{S,A}[R[A(S)]] &= \mathrm{E}_{S,A}\left[\frac{1}{n}\sum_{i=1}^{n} loss(A(S), z_i)\right] \\
&= \mathrm{E}_{S,S',A}\left[\frac{1}{n}\sum_{i=1}^{n} loss(A(S^{(i)}), z'_i)\right] \\
&\leq \mathrm{E}_{S,S',A}\left[\frac{1}{n}\sum_{i=1}^{n} [loss(A(S), z'_i) + \epsilon]\right] \\
&= \mathrm{E}_{S,S',A} R_S[A(S')] + n \cdot \frac{1}{n}\epsilon \\
&= \mathrm{E}_{S,A} R[A(S)] + \epsilon,
\end{aligned} \tag{7}$$

by the uniform stability of A. This indicates that the generalization error

$$\epsilon_{gen} = \mathrm{E}_{S,A}[R_S[A(S)] - R[A(S)]] \leq \epsilon.$$

Gradient descent is not stable since a change in a single datapoint affects every step of training and can lead to different classifiers, especially when the loss landscape is non-convex (Figure 7). SGD, on the other hand, is uniformly stable and quickly obtains small generalization errors [HRS15] because updates are usually performed according to the same objective function for S and S' and only approximately every m steps will the two diverge. For a finite number of steps (when the number of iterations is linear in the number of data points), the divergence will not be too great, providing further justification for early stopping as a means to improve generalizability. Generalization error will then be bounded even for large numbers of parameters and when no other regularization is used (although further regularization improves generalizability).

Stability bounds can be formally derived under Lipschitz and smoothness assumptions [HRS15]. For strongly convex problems, S and S' will never diverge by more than a fixed amount since the two solutions will move closer together on most steps. For convex functions, when encountering the same training samples, S and S' will not diverge, and they will only diverge a little when encountering different samples. For non-convex functions, generalization can be achieved when one uses small steps and few iterations, $O(m^c),\ c > 1$, so that when the differing point is observed, the divergence is not too great. Empirically, the derived bounds are too pessimistic and far from the stability observed during actual experiments.

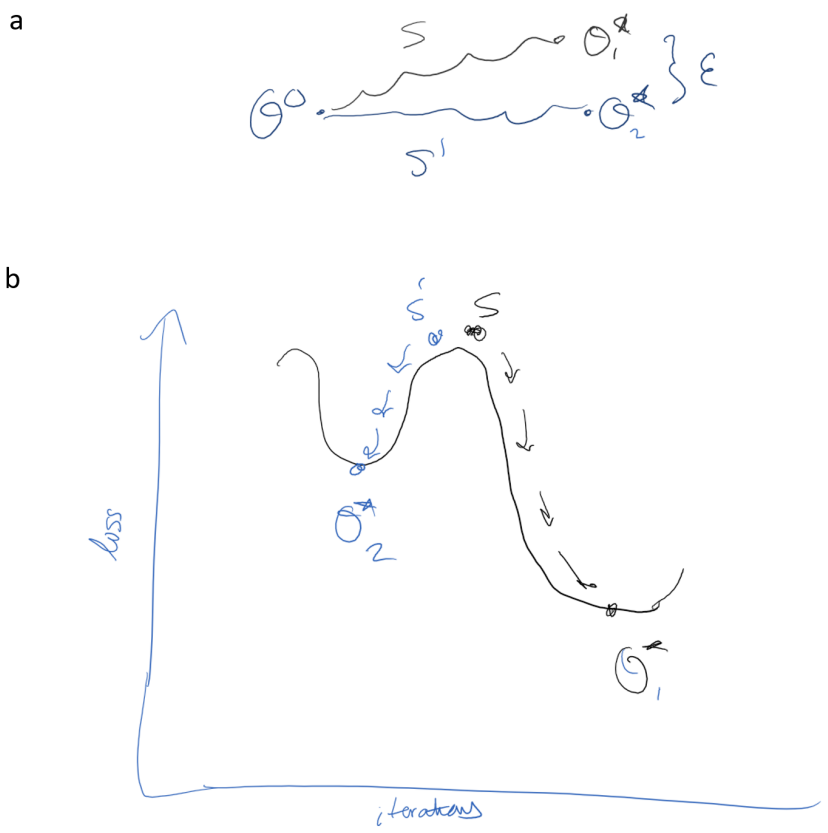


Figure 7: Difference in training error for training sets S and S', which differ by one data point. (a) For stochastic gradient descent, results diverge every 1/m steps on average so resulting solutions are within a loss of ϵ of each other with early stopping. (b) For non-convex functions, with gradient descent, the resulting solutions can differ substantially.

References

[AGNZ18] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach, 2018.

[Aroa] Sanjeev Arora. Cos597a: Machine learning seminar 2017: Generalization bounds.

[Arob] Sanjeev Arora. Proving generalization of deep nets via compression.

[BE02] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar):499–526, 2002.

[CCS+16] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys, 2016.

[DPBB17] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets, 2017.

[DR17] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data, 2017.

[HR18] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. *arXiv preprint arXiv:1803.01719*, 2018.

[HRS15] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[KMN+16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016.

[Mac91] David J.C. MacKay. Bayesian interpolation. *NEURAL COMPUTATION*, 4:415–447, 1991.

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?(no, it is not about internal covariate shift). *arXiv preprint arXiv:1805.11604*, 2018.

[ZBH+16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2016.