Yale University CPSC 683
Stability in Machine Learning: Generalization, Privacy & Replicability

Lecture 3: Stability of SGD and Randomization Tests

Instructor: Alkis Kalavasis, alkis.kalavasis@yale.edu

In Lecture 3, we will first continue our discussion on algorithmic stability. We will study the work of Hardt et al. (2016) on the stability of Stochastic Gradient Descent, probably the most applied ML algorithm. Next, we will depart from the framework of stability and discuss the experimental work of Zhang et al. (2021) on questioning the theories for generalization in modern deep learning.

1 Stochastic Gradient Descent

The most widely used optimization method in machine learning practice is stochastic gradient method (SGM). Stochastic gradient methods aim to minimize the empirical risk of a model by repeatedly computing the gradient of a loss function on a single training example, or a batch of few examples, and updating the model parameters accordingly. SGM is scalable, robust, and performs well across many different domains ranging from smooth and strongly convex problems to complex non-convex objectives.

- Let $L(w) = \frac{1}{n}\sum_i L_i(w) = \frac{1}{n}\sum_i \ell(w; x_i)$.
- Standard Gradient Update: $w \leftarrow w - \eta \nabla_w L(w)$.

In contrast, SGD performs updates as follows:

- Set $w_0 = 0$.
- Randomly shuffle samples in the training set.
- for $i = 1, ..., n : w \leftarrow w - \eta \nabla_w L_i(w)$.

More generally, SGD requires an unbiased estimate of the true gradient.

Insight 1: An advantage of SGD compared to GD

SGD is in many cases crucial for *computational efficiency* compared to vanilla gradient descent on the empirical loss. As an illustration, consider the truncated statistics setting in high dimensions (Daskalakis et al., 2018). Here, there is an unknown normal distribution $\mathcal{N}(\mu^*)$ (with identity covariance matrix for simplicity) in d dimensions and a truncation set $S \subseteq \mathbb{R}^d$, which we can access only with a membership oracle $1_S : \mathbb{R}^d \to \{0, 1\}$. We assume that the set has no particular structure except that is satisfies that $\int_S \mathcal{N}(\mu^\star; y)dy = \alpha > 1\%$.

The goal is to estimate μ^* in poly(d) time using i.i.d. samples from the truncated distribution $\mathcal{N}_S(\mu^\star)$, where $\mathcal{N}(\mu^\star; x) = \mathcal{N}(\mu^\star; x)1_S(x) / \int_S \mathcal{N}(\mu^\star; y)dy$.

A natural strategy to solve this problem is to run gradient descent on the log-likelihood objective (for any exponential family, the truncated negative log-likelihood is convex). Writing down the (negative) log-likelihood for a single sample x at our current guess μ, we can observe that

$$-\log \mathcal{N}(\mu; x) = \frac{1}{2}\|x - \mu\|^2 + \log\left(\int_S \exp\left(-\frac{1}{2}\|z - \mu\|^2\right)dz\right) .$$

Note that running gradient descent on this objective is intractable in high dimensions, since it in-

volves integrating over this high dimensional set S (and we only have membership access to the set). However, if we compute the gradient, we can see that

$$\nabla_\mu(-\log \mathcal{N}(\mu; x)) = -x + \mathop{\mathbf{E}}_{z\sim\mathcal{N}_S(\mu)}[z]\,.$$

This means that even if we cannot compute this gradient efficiently, we can compute efficiently an unbiased estimate of it (since we can sample from $\mathcal{N}_S(\mu)$, where μ is our current guess) given that the normal centered at μ places a lot of mass to S. This allows us to run SGD efficiently.

2 Stability of Stochastic Gradient Descent

2.1 Some Preliminaries from Lecture 2

We will need some notation from Lecture 2. Given algorithm A and training set S, define the population loss on a fresh sample z as

$$R(A, S) = \mathop{\mathbf{E}}_{z}\ell(A(S), z)\,.$$

Also, let the empirical loss on $S = \{z_1 = (x_1, y_1), ..., z_n = (x_n, y_n)\}$ be

$$R_{emp}(A, S) = \frac{1}{n}\sum_{i=1}^{n}\ell(A(S), z_i)\,.$$

Problem 1: High-Probability Generalization Bounds for specific algorithm A

For any ϵ, upper bound the tail probability

$$\mathop{\mathbf{Pr}}_{S}\left[|R_{emp}(A, S) - R(A, S)| > \epsilon\right]\,.$$

We remind the reader that even if an algorithm A generalizes in the above sense, it does not mean that it is a good learning algorithm. If one additionally manages to show that A achieves small training error, we can guarantee that it also gets small generalization error.

In this Lecture, we will show that parametric models trained by a stochastic gradient method (SGM) with few iterations have vanishing generalization error. We prove our results by arguing that SGD is algorithmically stable in the sense of Lecture 2.

Definition 1: Uniform Stability (Bousquet and Elisseeff, 2002)

A randomized algorithm A has **uniform stability** ϵ_{stab} with respect to ℓ if for all datasets $S, S' \in Z^n$ that differ in at most one example, we have

$$\sup_z \mathop{\mathbf{E}}_{A}|\ell(A(S), z) - \ell(A(S'), z)| \le \epsilon_{stab}\,,$$

where the expectation is is taken only over the internal randomness of A.

In words, an algorithm is stable if the training error it achieves varies only slightly if we change any single training data point. The precise notion of stability we use is known as uniform stability. It states that a randomized algorithm A is uniformly stable if for all data sets differing in only one element, the learned models produce nearly the same predictions.

Observe that the two executions are *coupled* since the randomness is shared across them. This will be crucial for the upcoming analysis. (Think of it as sharing the random seed across the two executions.)

In what follows the algorithm $A : (\mathcal{X} \times \mathcal{Y})^n \to W$ takes as input a training set and runs T steps of SGD on a parametric model (e.g., weights of a neural network) to output parameters w_T.

2.2 Setup for SGD Analysis

We start with the work of Hardt et al. (2016). Given n samples $S = \{z_1, ..., z_n\}$, consider a decomposable function

$$f(w) = \frac{1}{n} \sum_i f(w; z_i) \,,$$

where $f(w; z_i)$ is the loss of w on the example z_i. The SGD update is

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t; z_{i_t}) \,.$$

The important thing here is the index i_t : The SGD algorithm is obtained by performing the SGD update T times where the indices $i_1, ..., i_T$ are *randomly and independently chosen*. There are two popular schemes for choosing the examples' indices. One is to pick it uniformly at random in $[n]$ at each step. The other is to choose a random permutation over $[n]$ and cycle through the examples repeatedly in the order determined by the permutation. The results hold for both variants.

2.3 The Main Result: SGD is Stable

In order to prove that the stochastic gradient method is stable, we will analyze the output of the algorithm on two data sets that differ in precisely one location.

Step 1. If the loss function is L-Lipschitz with respect to w for every example z, we have

$$\mathbf{E} \left| f(w; z) - f(w'; z) \right| \leq L \, \mathbf{E} \left\| w - w' \right\|$$

for all w and w'. This means that $\|\nabla_w f(w; z)\| \leq L$ for any z. Hence, it suffices to analyze how w_t and w'_t diverge in the domain as a function of time t.

Step 2. Because of the recursive structure of SGD, our goal is to control $\delta_t = \|w_t - w'_t\|$ in expectation as a function of δ_{t-1}.

Step 3. Based on the random index choice and the fact that S, S' differ in one element, there are two cases to consider.

- In the first case, SGD selects the index of an example at step t which is identical in S and S'. Unfortunately, it could still be the case that δ_t grows, since w_t and w'_t differ and so the gradients at these two points may still differ.
- The second case to consider is when SGD selects the one example to update in which S and S' differ. Note that this happens only with probability $1/n$ if examples are selected randomly. In that case,

  $$\delta_t = \|w_{t-1} - \alpha_{t-1} \nabla_w f(w_{t-1}; z) - w'_{t-1} + \alpha_{t-1} \nabla_w f(w'_{t-1}; z')\| \leq \delta_{t-1} + 2\alpha_{t-1} L \,,$$

  since f is L-Lipschitz.

It remains to combine the two cases to complete the proof.

Step 4. Let us begin with the simple case where f is convex.

Definition 2

A function $f : X \to \mathbb{R}$ is convex if for all $x, y \in X$, it holds

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \,.$$

A function $f : X \to \mathbb{R}$ is β-smooth if for all $x, y \in X$, it holds

$$\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\| \,.$$

Theorem 1: Convex Case (Hardt et al., 2016)

Assume that the loss function $f(\cdot; z)$ is β-smooth, convex and L-Lipschitz for any z. Suppose that we run SGD (with either the random index resampling scheme or the permutation one) with step sizes $\alpha_t \leq 2/\beta$ for T iterations with the same initialization. Then, SGD satisfies uniform stability with $\epsilon_{stab} \leq \frac{2L^2}{n} \sum \alpha_t$.

Proof. It suffices to bound $\mathbf{E}[\delta_T]$. Observe that at step t, with probability $1 - 1/n$, the example selected by SGD is the same in both S and S'. This means that

$$\mathbf{E}[\delta_{t+1}] \leq (1 - 1/n) \cdot S_t^{(1)} + 1/n \cdot S_t^{(2)}$$

where $S_t^{(1)}$ corresponds to the case where the method selects a common point of S, S' :

$$S_t^{(1)} = \mathbf{E} \, \|w_t - \alpha_t \nabla_w f(w_t; z) - w_t' + \alpha_t \nabla_w f(w_t'; z)\|$$

and $S_t^{(2)}$ corresponds to picking the index of the differing point:

$$S_t^{(2)} = \mathbf{E} \, \|w_t - \alpha_t \nabla_w f(w_t; z) - w_t' + \alpha_t \nabla_w f(w_t'; z')\| \,.$$

Analysis for $S_t^{(1)}$. We will now use convexity and smoothness. An update rule G is η-expansive if for all $w_1, w_2 \in \Omega$, $\|G(w_1) - G(w_2)\| \leq \eta \|w_1 - w_2\|$. It is not difficult to see that the SGD update rule is 1-expansive if f is β-smooth and $\alpha \leq 2/\beta$ (see Lemma 3.7.2 in Hardt et al. (2016)). Note that if we only use smoothness we get that G is $(1 + \alpha\beta)$-expansive. But, convexity and smoothness imply that the gradients are co-coercive, namely

$$\langle \nabla g(w_1) - \nabla g(w_2), w_1 - w_2 \rangle \geq \frac{1}{\beta} \|\nabla g(w_1) - \nabla g(w_2)\|^2 \,.$$

Exercise 1

Prove that ∇f is co-coersive if f is convex and smooth.
(It suffices to show that $f(y) - f(x) - \nabla f(x)^\top (y - x) \geq \frac{1}{2\beta} \|\nabla f(x) - \nabla f(y)\|^2$.)

This now gives that

$$\|G(w_1) - G(w_2)\|^2 = \|w_1 - w_2\|^2 + \alpha^2 \|\nabla g(w_1) - \nabla g(w_2)\|^2 - 2\alpha \langle \nabla g(w_1) - \nabla g(w_2), w_1 - w_2 \rangle \leq \|w_1 - w_2\|^2 \,.$$

This means that $S_t^{(1)} \leq \mathbf{E} \, \delta_t$.

Analysis for $S_t^{(2)}$. Based on the discussion in Step 3, we will get that $S_t^{(2)} \leq \mathbf{E} \, \delta_t + 2\alpha_t L$.

Combining the two and using recursion, we get that $\mathbf{E} \, \delta_T \leq 2L \sum_t \frac{\alpha_t}{n}$. This means that SGD is uniformly stable with parameter $2L^2 \sum_t \frac{\alpha_t}{n}$ (here we used the fact that we started from the same initial condition). □

Non-Convex Regime The above extends to non-convex objectives. The main idea is to observe that SGD typically makes several steps before it even encounters the one example on which two data sets in the stability analysis differ.

Lemma 1: (Hardt et al., 2016)

Assume that the loss function $f(\cdot; z)$ is nonnegative and L-Lipschitz for all z. Let S and S' be two samples of size n differing in only a single example. Denote by w_T and w'_T the output of T steps of SGD on S and S'. respectively. Then, for every $z \in Z$ and every $t_0 \in \{0, 1, ..., n\}$, under both the random update rule and the random permutation rule, we have

$$\mathbf{E}\left|f(w_T; z) - f(w'_T; z)\right| \leq \frac{t_0}{n} \sup_{w,z} f(w, z) + L\,\mathbf{E}[\delta_T | \delta_{t_0} = 0]\,.$$

This can be used to show that

Theorem 2: Stability of SGD (Hardt et al., 2016)

Assume that $f(\cdot; z) \in [0, 1]$ is an L-Lipschitz and β-smooth loss function for every z. Suppose that we run SGD for T steps with monotonically non-increasing step sizes $\alpha_t \leq c/t$. Then, SGD satisfies uniform stability with

$$\epsilon_{stab} = \frac{T^{1-\frac{1}{\beta c+1}}}{n}\,.$$

This shows that the method generalizes provided that (i) the steps are sufficiently small α_t is small) and (ii) the number of iterations is not too large (T is small). More specifically,the number of steps of stochastic gradient can grow as n^C for a small $C > 1$. This can be used to provide some explanation on why neural networks can be trained for multiple epochs of stochastic gradient descent (multiple passes of the training set) and still exhibit good generalization.

Insight 2

The above bounds are algorithm specific. They specifically apply to SGD. Since the number of iterations we allow can be larger than the sample size, an arbitrary algorithm could easily achieve small training error by *memorizing all training data* with no generalization ability. In contrast, if the stochastic gradient method manages to fit the training data in a reasonable number of iterations, it is guaranteed to generalize.

We remind the reader that the above results do not focus on algorithms that yield low training error, but provide insights on algorithms that yield low generalization error. If, in addition, one can achieve low training error quickly on a non-convex problem with stochastic gradient, the above results guarantee that the resulting model generalizes well.

Exercise 2

Complete the proofs of the above results.

3 Rethinking Generalization in Deep Learning

There are a variety of theories proposed to explain generalization: Uniform convergence (recall Lecture 1), and algorithmic stability (Lectures 2,3) are but a few of the important conceptual tools to reason about generalization. The work of Zhang et al. (2021) aims to interrogate these different theories of generalization via simple experiments.

3.1 The Randomization Test

The first experiment is very simple and is based on a long statistical literature (Edgington and Onghena, 2007). They create a copy of the training data where they replace each label independently by a random label chosen from the set of valid labels.

A dog picture labeled "dog" in the true dataset might thus become a dog picture labeled "cat". The crucial point is that *the randomization breaks any relationship between the instance, for example, the image, and the label.* Then they run the learning algorithm both on the natural data and on the randomized data with identical settings and model choice. By design, no generalization is possible on the randomized data (it is information-theoretically impossible to get better than $1/2$ even with 2 labels).

For any purported measure of generalization, it is now possible to compare how it fares on the natural data versus the randomized data. If it turns out to be the same in both cases, it could not possibly be a good measure of generalization for it cannot even distinguish learning from natural data (where generalization is possible) from learning on randomized data (where no generalization is possible). The main observation is:

Deep neural networks can easily fit random labels.

More precisely, when trained on a completely random labeling of the true data, neural networks achieve zero training error. The test error, of course, is no better than random chance as there is no correlation between the training labels and the test labels.

- The effective capacity[1] of neural networks is sufficient for memorizing the entire data set. This is interesting since one may suspect that the reason that neural networks could fit the training data is that they make use of abstractions between the images and the labels. However, since the training loss on the randomized dataset is also zero, this is not the case.
- Plot (b) deals with the training overhead: On the one side, it is important to see that even the optimization part on random labels remains 'easy'. In fact, training time increases only by a small constant factor compared with training on the true labels (see Plot (b) for the training overhead for different models). Hence what drives generalization cannot be identical to what makes optimization of deep neural networks easy in practice. On the other side, fitting random labels takes 3x time compared to the true ones, which is an important training overhead.

Connection to Hardt et al. (2016) and Uniform Stability Uniform stability of a learning algorithm is *independent of the labeling of the training data*. Hence, the concept is not strong enough to distinguish between the models trained on the true labels (small generalization error) and models trained on random labels (high generalization error). This also highlights why the analysis of Hardt et al. (2016) for non-convex optimization was rather pessimistic, allowing only a very few passes over the data (recall Theorem 2). The randomization test essentially shows that training neural networks is not uniformly stable for many passes over the data. However, it seems that there exists something reassuring about the pessimistic analysis of Theorem 2: when the number of steps are sufficiently small, then SGD is actually stable and the training error is close to the population error; however, in the randomized dataset, the training error is pretty high

[1] By effective capacity, the authors of Zhang et al. (2021) informally refer to the size of the subset of models that is effectively achievable by the learning algorithm.

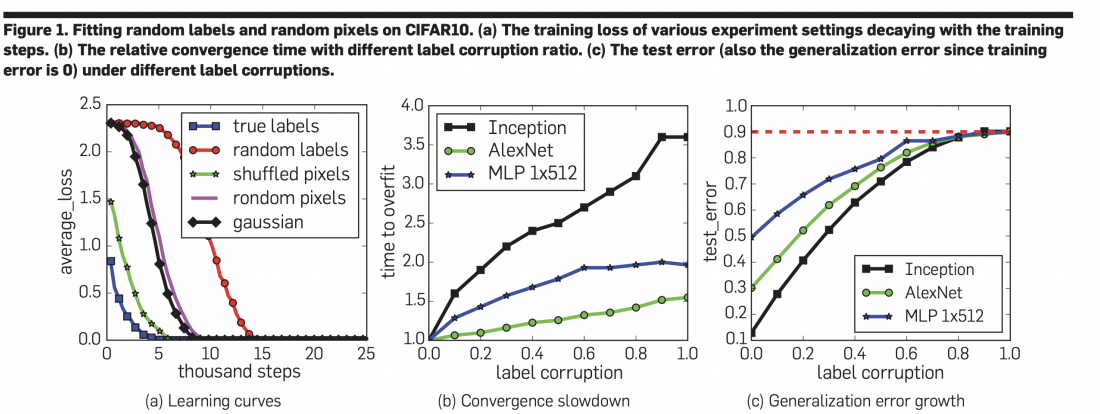

Figure 1: Taken from Zhang et al. (2021).

after a few iterations (it requires several steps to go to 0) and this is aligned to the prediction of stability theory.

Implication to Rademacher Complexity Recall that the empirical Rademacher complexity of a function class $\mathcal{F}$ on an unlabeled dataset $\{x_1, ..., x_n\}$ is defined as

$$\widehat{R}_n(\mathcal{F}) = \mathop{\mathbf{E}}_{\sigma} \sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i \in [n]} \sigma_i f(x_i) \,.$$

The Rademacher complexity measures the ability of a function class to fit random binary label assignments, which closely resemble the randomization test (the points $x_1, ..., x_n$ are drawn from the marginal distribution on features, which is not changing when applying the randomization test). Since the neural networks fit random labels, we can see that $\widehat{R}_n(\mathcal{F}) \approx 1$. Often one bounds the Rademacher complexity of a function class $\mathcal{L} = \{\ell(f(x), y) : f \in \mathcal{F}\}$. For L-Lipschitz loss functions ℓ and real-valued class $\mathcal{F}$, it holds

$$\widehat{R}_n(\mathcal{L}) \leq L\widehat{R}_n(\mathcal{F}).$$

Since the neural networks fit random labels, Zhang et al. (2021) expect that $\widehat{R}_n(\mathcal{L})$ approximately achieves the maximum for the corresponding loss class.

Note that a trivial bound on the Rademacher complexity that does not lead to useful generalization bounds in realistic settings, since roughly speaking $\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h) \leq 2R_m(\mathcal{H}) + \sqrt{\log(1/\delta)/m)}$, where $R_m(\mathcal{H}) = \mathbf{E}_S[\widehat{R}_m(\mathcal{H})]$. Hence, the claim of Zhang et al. (2021) is that the effective capacity of neural networks trained by SGD cannot be small enough to explain the success of the model on generalizing on the natural data.[2]

3.2 The Role of Regularization

The second part of Zhang et al. (2021) studies regularization. Apart from the effective capacity of the model family, conventional wisdom attributes small generalization error to regularization techniques used during training.

Regularizers are the standard tool in theory and practice to mitigate overfitting in the regime when there are more parameters than data points. The basic idea is that although the original hypothesis is too large to generalize well, regularizers help confine learning to a subset of the hypothesis space with manageable complexity. By adding an explicit regularizer, say by penalizing the norm of the optimal solution, the effective Rademacher complexity of the possible solutions is dramatically reduced.

[2]Thought: However the effective size of the class of models explored by SGD is likely to depend on the training distribution, which changes between the two experiments.

- Data augmentation: augment the training set via domain-specific transformations. For image data, commonly used transformations include random cropping, random perturbation of brightness, saturation, hue and contrast.
- Weight decay: equivalent to a ℓ_2 regularizer on the weights; also equivalent to a hard constrain of the weights to an Euclidean ball, with the radius decided by the amount of weight decay.

Table 1. The training and test accuracy (in %) of various models on the CIFAR10 dataset.

Model	# params	Random crop	Weight decay	Train accuracy	Test accuracy
Inception	1,649,402	Yes	Yes	100.0	89.05
		Yes	No	100.0	89.31
		No	Yes	100.0	86.03
		No	No	100.0	85.75
(fitting random labels)		No	No	100.0	9.78
Inception w/o BatchNorm	1,649,402	No	Yes	100.0	83.00
		No	No	100.0	82.00
(fitting random labels)		No	No	100.0	10.12
Alexnet	1,387,786	Yes	Yes	99.90	81.22
		Yes	No	99.82	79.66
		No	Yes	100.0	77.36
		No	No	100.0	76.07
(fitting random labels)		No	No	99.82	9.86
MLP 3 × 512	1,735,178	No	Yes	100.0	53.35
		No	No	100.0	52.39
(fitting random labels)		No	No	100.0	10.48
MLP 1 × 512	1,209,866	No	Yes	99.80	50.39
		No	No	100.0	50.51
(fitting random labels)		No	No	99.34	10.61

Figure 2: Performance with and without data augmentation and weight decay are compared. Taken from Zhang et al. (2021).

Figure 2 shows that even with all of the regularizers turned off, all of the models still generalize well. Observe that even with the regularizers turned on the models can fit the random labels on ImageNet (see Figure 3).

Table 2: The top-1 and top-5 accuracy (in percentage) of the Inception v3 model on the ImageNet dataset. We compare the training and test accuracy with various regularization turned on and off, for both true labels and random labels. The original reported top-5 accuracy of the Alexnet on ILSVRC 2012 is also listed for reference. The numbers in parentheses are the best test accuracy during training, as a reference for potential performance gain of early stopping.

data aug	dropout	weight decay	top-1 train	top-5 train	top-1 test	top-5 test
ImageNet 1000 classes with the original labels						
yes	yes	yes	92.18	99.21	77.84	93.92
yes	no	no	92.33	99.17	72.95	90.43
no	no	yes	90.60	100.0	67.18 (72.57)	86.44 (91.31)
no	no	no	99.53	100.0	59.80 (63.16)	80.38 (84.49)
Alexnet (Krizhevsky et al., 2012)			-	-	-	83.6
ImageNet 1000 classes with random labels						
no	yes	yes	91.18	97.95	0.09	0.49
no	no	yes	87.81	96.15	0.12	0.50
no	no	no	95.20	99.14	0.11	0.56

Table 2 shows the performance on Imagenet with true labels and random labels, respectively.

Figure 3: Taken from Zhang et al. (2021).

References

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526.

Daskalakis, C., Gouleakis, T., Tzamos, C., and Zampetakis, M. (2018). Efficient statistics, in high dimensions, from truncated samples. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 639–649. IEEE.

Edgington, E. and Onghena, P. (2007). *Randomization Tests*. CRC Press.

Hardt, M., Recht, B., and Singer, Y. (2016). Train faster, generalize better: Stability of stochastic gradient descent. In *International conference on machine learning*, pages 1225–1234. PMLR.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.