SANJEEV ARORA

OTHER CONTRIBUTORS: RAMAN ARORA, JOAN BRUNA, NADAV COHEN, SIMON DU, RONG GE, SURIYA GUNASEKAR, ELAD HAZAN, CHI JIN, JASON LEE, TENGYU MA, BEHNAM NEYSHABUR, ZHAO SONG

THEORY OF DEEP LEARNING

Contents

List of Figures

List of Tables

Introduction

Date of this version: March 28 2022

This monograph discusses the emerging theory of deep learning. It originated from notes by the lecturers at a graduate seminar taught at Princeton University in Fall 2019 in conjunction with a Special Year on Optimization, Statistics, and Machine Learning at the Institute for Advanced Study. Sanjeev Arora has cleaned up and extended the book during two subsequent offerings of the course in Spring'21 and Spring'22.

This is closer to lecture notes than to a book. It probably has many errors and typos.

1

Basic Setup and some math notions

This Chapter introduces the basic nomenclature. Training/test error, generalization error etc. ≪Tengyu notes: Todos: Illustrate with plots: a typical training curve and test curve

Mention some popular architectures (feed forward, convolutional, pooling, resnet, densenet) in a brief para each. ≫

We review the basic notions in statistical learning theory.

- A space of possible data points $\mathcal{X}$.
- A space of possible labels $\mathcal{Y}$.
- A joint probability distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$. We assume that our training data consist of n data points

  $$(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \overset{\text{i.i.d.}}{\sim} \mathcal{D},$$

  each drawn independently from $\mathcal{D}$.
- Hypothesis space: $\mathcal{H}$ is a family of hypotheses, or a family of predictors. E.g., $\mathcal{H}$ could be the set of all neural networks with a fixed architecture: $\mathcal{H} = \{h_\theta\}$ where h_θ is neural net that is parameterized by parameters θ.
- Loss function: $\ell : (\mathcal{X} \times \mathcal{Y}) \times \mathcal{H} \to \mathbb{R}$.
  - E.g., in binary classification where $\mathcal{Y} = \{-1, +1\}$, and suppose we have a hypothesis $h_\theta(x)$, then the logistic loss function for the hypothesis h_θ on data point (x, y) is

    $$\ell((x,y), \theta) = \frac{1}{1 + \exp(-y h_\theta(x))}.$$
- Expected loss:

  $$L(h) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} [\ell((x,y), h)] .$$

  Recall $\mathcal{D}$ is the data distribution over $\mathcal{X} \times \mathcal{Y}$.

- Training loss (also known as empirical risk):

$$\widehat{L}(h) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(\left(x^{(i)}, y^{(i)}\right), h\right),$$

  where $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right)$ are n training examples drawn i.i.d. from $\mathcal{D}$.

- Empirical risk minimizer (ERM): $\widehat{h} \in \arg\min_{h \in \mathcal{H}} \widehat{L}(h)$.

- Regularization: Suppose we have a regularizer $R(h)$, then the regularized loss is

$$\widehat{L}_\lambda(h) = \widehat{L}(h) + \lambda R(h)$$

  .

≪Suriya notes: Misc notations: gradient, hessian, norms≫

1.1 List of useful math facts

Now we list some useful math facts.

1.1.1 Probability tools

In this section we introduce the probability tools we use in the proof. Lemma 1.1.4, 1.1.5 and 1.1.6 are about tail bounds for random scalar variables. Lemma 1.1.7 is about cdf of Gaussian distributions. Finally, Lemma 1.1.8 is a concentration result on random matrices.

Lemma 1.1.1 (Markov's inequality)**.** *If x is a nonnegative random variable and $t > 0$, then the probability that x is at least t is at most the expectation of x divided by t:*

$$\Pr[x \geq t] \leq \mathbb{E}[x]/t.$$

Lemma 1.1.2 (Chebyshev's inequality)**.** *Let x denote a nonnegative random variable and $t > 0$, then*

$$\Pr[|x - \mathbb{E}[x]| \geq t] \leq \mathrm{Var}[x]/t^2.$$

Next we present some concentration bounds regarding sum of independent random variables. The rule of thumb underlying concentration bounds is the *Central Limit Theorem*.

Theorem 1.1.3 (Central Limit Thm, informal)**.** *If $X_1, X_2, \ldots, X_n$ are independent random variables of mean $\mu_1, \mu_2, \ldots, \mu_n$ and variances $\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2$ then as n gets larger, $\sum_i X_i$ behaves like the normal distribution $\mathcal{N}(\sum_i \mu_i, \sum_i \sigma_i^2)$.*

Concentration bounds are quantitative versions of this and work also in settings where p_i's and σ_i's could depend on n, the total number of variables. But in many setting the CLT is a good rule of thumb.

Lemma 1.1.4 (Chernoff bound [Che52]). *Let $X = \sum_{i=1}^n X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Let $\mu = \mathbb{E}[X] = \sum_{i=1}^n p_i$. Then*

1. $\Pr[X \geq (1+\delta)\mu] \leq \exp(-\delta^2\mu/3), \forall \delta > 0$;
2. $\Pr[X \leq (1-\delta)\mu] \leq \exp(-\delta^2\mu/2), \forall 0 < \delta < 1$.

Lemma 1.1.5 (Hoeffding bound [Hoe63]). *Let $X_1, \cdots, X_n$ denote n independent bounded variables in $[a_i, b_i]$. Let $X = \sum_{i=1}^n X_i$, then we have*

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq 2\exp\left(-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

Lemma 1.1.6 (Bernstein inequality [Ber24]). *Let $X_1, \cdots, X_n$ be independent zero-mean random variables. Suppose that $|X_i| \leq M$ almost surely, for all i. Then, for all positive t,*

$$\Pr\left[\sum_{i=1}^n X_i > t\right] \leq \exp\left(-\frac{t^2/2}{\sum_{j=1}^n \mathbb{E}[X_j^2] + Mt/3}\right).$$

Lemma 1.1.7 (Anti-concentration of Gaussian distribution). *Let $X \sim N(0, \sigma^2)$, that is, the probability density function of X is given by $\phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$. Then*

$$\Pr[|X| \leq t] \in \left(\frac{2}{3}\frac{t}{\sigma}, \frac{4}{5}\frac{t}{\sigma}\right).$$

Lemma 1.1.8 (Matrix Bernstein, Theorem 6.1.1 in [Tro15]). *Consider a finite sequence $\{X_1, \cdots, X_m\} \subset \mathbb{R}^{n_1 \times n_2}$ of independent, random matrices with common dimension $n_1 \times n_2$. Assume that*

$$\mathbb{E}[X_i] = 0, \forall i \in [m] \quad \text{and} \quad \|X_i\| \leq M, \forall i \in [m].$$

Let $Z = \sum_{i=1}^m X_i$. Let $\mathrm{Var}[Z]$ be the matrix variance statistic of sum:

$$\mathrm{Var}[Z] = \max\left\{\left\|\sum_{i=1}^m \mathbb{E}[X_i X_i^\top]\right\|, \left\|\sum_{i=1}^m \mathbb{E}[X_i^\top X_i]\right\|\right\}.$$

Then

$$\mathbb{E}[\|Z\|] \leq (2\mathrm{Var}[Z] \cdot \log(n_1 + n_2))^{1/2} + M \cdot \log(n_1 + n_2)/3.$$

Furthermore, for all $t \geq 0$,

$$\Pr[\|Z\| \geq t] \leq (n_1 + n_2) \cdot \exp\left(-\frac{t^2/2}{\mathrm{Var}[Z] + Mt/3}\right).$$

explain these in a para

A useful shorthand will be the following: If $y_1, y_2, \ldots, y_m$ are independent random variables each having mean 0 and taking values in $[-1, 1]$, then their average $\frac{1}{m}\sum_i y_i$ behaves like a Gaussian variable with mean zero and variance at most $1/m$. In other words, the probability that this average is at least ϵ in absolute value is at most $\exp(-\epsilon^2 m)$.

1.1.2 *Singular Value Decomposition*

TBD.

2
Basics of Optimization

This chapter sets up the basic analysis framework for gradient-based optimization algorithms and discuss how it applies to deep learning. The algorithms work well in practice; the question for theory is to analyse them and give recommendations for practice. This has proved harder, and in recent years it has become clearer that classical ways of thinking about optimization may not match well with phenomena encountered in deep learning.

The basic conceptual framework in optimization builds upon simple Taylor approximation (Equation 2.1) of the loss function and thus relies upon derivatives (of various orders) of the loss function.

≪Suriya notes: To ground optimization to our case, we can also mention that f is often of the either the ERM or stochastic optimization form $L(w) = \sum l(w; x, y)$ - it might also be useful to mention that outside of this chapter, we typically use f as an alternative for h to denote a function computed≫

2.1 Gradient descent (GD)

Suppose we wish to minimize a continuous function $f(w)$ over $\mathbb{R}^d$.

$$\min_{w \in \mathbb{R}^d} f(w) \,.$$

The gradient descent (GD) algorithm is

$$\begin{aligned} w_0 &= \text{initialization} \\ w_{t+1} &= w_t - \eta \nabla f(w_t) \end{aligned}$$

where η is called *step size* or *learning rate.* The choice of η is important and a main subject in the rest of the Chapter.

One motivation or justification of the GD is that the update direction $-\nabla f(w_t)$ is the steepest descent direction locally. Consider the

Taylor expansion at a point w_t

$$f(w) = f(w_t) + \underbrace{\langle \nabla f(w_t), w - w_t \rangle}_{\text{linear in } w} + \underbrace{\frac{1}{2}(w - w_t)^T \nabla^2 f(w_t)(w - w_t)}_{\text{quadratic in } w} + \cdots \tag{2.1}$$

here $\nabla^2(f)$ is the matrix of 2nd order derivatives called *Hessian*. Its (i, j) entry is $\partial^2 f / \partial w_i \partial w_j$. Note that it is a symmetric matrix.

Suppose we drop the higher-order term and only optimize the first order approximation within a neighborhood of w_t

$$\begin{aligned} \arg\min_{w \in \mathbb{R}^d} \quad & f(w_t) + \langle \nabla f(w_t), w - w_t \rangle \\ \text{s.t.} \quad & \|w - w_t\|_2 \leq \epsilon \end{aligned}$$

Problem 2.1.1. *Show that the optimizer of the program above is equal to $w + \delta$ where $\delta = -\alpha \nabla f(w_t)$ for some positive scalar α.*

In other words, to locally minimize the first order approximation of $f(\cdot)$ around w_t, we should move towards the direction $-\nabla f(w_t)$. [1]

The classic back-propagation algorithm (Chapter 4) is used to efficiently compute the gradient of the loss. Note that today's deep nets often use nonlinear activations, e.g., ReLU, that make the function computed by the net non-differentiable. However, this differentiability is of the mild sort and does not appear to be an issue in practice. [2]

2.1.1 Upperbound on the Taylor Expansion via Smoothness

The most basic analysis of training speed of GD involves the smoothness of the loss function.

Definition 2.1.2 (L-smooth). *A function f is L-smooth in a domain if for every w in the domain all eigenvalues of $\nabla^2 f(w)$ lie in the interval $[-L, L]$.*

Problem 2.1.3. *Prove that if f is L-smooth then*

$$f(w) \leq f(w_t) + \langle \nabla f(w_t), w - w_t \rangle + \frac{L}{2}\|w - w_t\|_2^2 \tag{2.2}$$

2.1.2 Descent lemma for gradient descent

The following says that with gradient descent and small enough learning rate, the function value always decreases unless the gradient at the iterate is zero. (Points where gradient is zero are called *stationary points.*)

Lemma 2.1.4 (Descent Lemma). *Suppose f is L-smooth. Then, if $\eta < 1/L$, we have*

$$f(w_{t+1}) \leq f(w_t) - \frac{\eta}{2} \cdot \|\nabla f(w_t)\|_2^2$$

[1] Gradient descent is not guaranteed to find optimum solutions for general loss functions. For instance, complexity theory shows that given a degree 4 polynomial $p(w_1, w_2, \ldots, w_n)$ in n variables of total degree at most 6, it is NP-hard to determine whether or not it is 0 for some assignment to the variables. This can be proven easily using NP-completeness of 3SAT problem.

[2] More recently, replacing ReLU activations with differentiable ones such as SWISH or GeLU has been found to result in no reduction in performance.

The proof uses the Taylor expansion. The main idea is that even using the upper bound provided by equation (2.2) suffices. [3]

Proof. We have that

$$\begin{aligned} f(w_{t+1}) &= f(w_t - \eta \nabla f(w_t)) \\ &\le f(w_t) + \langle \nabla f(w_t), -\eta \nabla f(w_t) \rangle + \frac{L}{2} \| \eta^2 \nabla f(w_t) \|_2^2 \\ &= f(w_t) - (\eta - \eta^2 L/2) \| \nabla f(w_t) \|_2^2 \\ &\le f(w_t) - \frac{\eta}{2} \cdot \| \nabla f(w_t) \|_2^2, \end{aligned}$$

where the second step follows from Eq. (2.2), and the last step follows from $\eta \le 1/L$. □

We've shown GD stops making progress when the gradient ∇ becomes zero. Is this good enough?

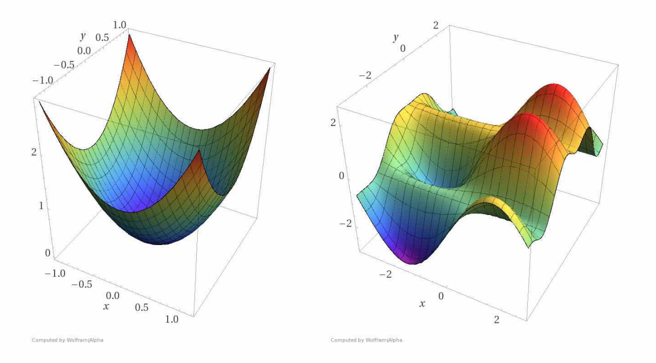


Figure 2.1: Convex and Nonconvex Functions in two variables. For nonconvex functions GD will reach a stationary point, where gradient is zero. (Figure from kdnuggets.org)

The loss function for deep learning is non-convex when the network has more than one layer. Thus GD is not guaranteed to produce a global optimum. Nevertheless, the solutions it finds in practice attain fairly low —even near zero– value of the objective. (Recall that the loss function is usually non-negative.) Elsewhere in the book this property of GD is explained in some concrete settings.

Definition 2.1.5. *We say w is a* stationary point *of f if $\nabla(f(w)) = 0$. If in addition $\nabla^2(f)$ is positive-semidefinite at w then w is called a local minimum.*

2.2 *Stochastic gradient descent (SGD)*

SGD is a very practical variant of gradient descent for large datasets. Recall that

$$\widehat{L}(h) = \frac{1}{n} \sum_{i=1}^{n} \ell \left((x^{(i)}, y^{(i)}), h \right).$$

[3] The proof also shows that descent occurs so long as $\eta < 2/L$.

Computing the gradient $\nabla \widehat{L}(h)$ scales linearly in n, the size of the training dataset. Stochastic gradient descent (SGD) estimates the gradient by sampling a small number of training datapoints and computing the average. By usual sampling theorems, the gradient estimate approaches true gradient as the sample size grows.

The updates: We simplify the notations a bit for ease of exposition. We consider optimizing the function

$$\frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

So here f_i corresponds to $\ell((x^i, y^{(i)}), h)$ in the statistical learning setting. At each iteration t, the SGD algorithm first samples $i_1, \dots, i_B$ uniformly from $[n]$, and then computes the estimated gradient using the samples:

$$g_S(w) = \frac{1}{B} \sum_{k=1}^{B} \nabla f_{i_k}(w_t)$$

Here S is a shorthand for $\{i_1, \dots, i_B\}$. The SGD algorithm updates the iterate by

$$w_{t+1} = w_t - \eta \cdot g_S(w_t).$$

Note that if the learning rate η is very small, then the parameters will not change much over a sequence of updates, and so SGD would tend to be similar to (full) GD. However, when η is not too small (and batch size is small), the gradient estimates from batches are noisy estimates of the true gradient. One would imagine this makes SGD worse than GD. In practice SGD tends to be superior to GD. First, since the stochastic estimation of gradient is so efficient, SGD allows far more iterations in the same computational budget. Second, SGD appears to have beneficial effect on generalization, meaning the solutions it finds tend to have better test error than those found by GD. Later chapters will cover theories that try to account for superiority of SGD over GD with respect to generalization.

2.3 *Accelerated Gradient Descent*

The basic version of accelerated gradient descent algorithm is called heavy-ball algorithm. It has the following update rule:

$$w_{t+1} = w_t - \eta \nabla f(w_t) + \beta(w_t - w_{t-1})$$

Here $\beta(w_{t+1} - w_t)$ is the so-called momentum term. The motivation and the origin of the name of the algorithm comes from that it can be viewed as a discretization of the second order ODE:

$$\ddot{w} + a\dot{w} + b\nabla f(w) = 0$$

Another equivalent way to write the algorithm is

$$\begin{aligned} u_t &= -\nabla f(w_t) + \beta u_{t-1} \\ w_{t+1} &= w_t + \eta u_t \end{aligned}$$

Exercise: verify the two forms of the algorithm are indeed equivalent.

Another variant of the heavy-ball algorithm is due to Nesterov

$$\begin{aligned} u_t &= -\nabla f(w_t + \beta \cdot (u_t - u_{t-1})) + \beta \cdot u_{t-1}, \\ w_{t+1} &= w_t + \eta \cdot u_t. \end{aligned}$$

One can see that u_t stores a weighed sum of the all the past gradients. In effect, the update of w_t depends on all past gradients. This is another interpretation of the accelerate gradient descent algorithm.

Nesterov gradient descent works similarly to the heavy ball algorithm empirically for training deep neural networks. For convex loss functions it has the advantage of stronger worst case guarantees. Both versions of accelerated GD can be used with stochastic gradient, but little is know about the theoretical guarantees about stochastic accelerated gradient descent.

2.4 *Running time: Learning Rates and Update Directions*

When the iterations of GD are near a local minimum, the behavior of gradient descent is clearer because the function can be locally approximated by a quadratic function. In this section, we assume for simplicity that we are optimizing a convex quadratic function, and get some insight on how the curvature of the function influences the convergence of the algorithm.

We use gradient descent to optimize

$$\min_w \frac{1}{2} w^\top A w$$

where $A \in \mathbb{R}^{d\times d}$ is a positive semidefinite matrix, and $w \in \mathbb{R}^d$. Remark: w.l.o.g, we can assume that A is a diagonal matrix. **Diagonalization is a fundamental idea in linear algebra.** Suppose A has singular vector decomposition $A = U\Sigma U^\top$ where Σ is a diagonal matrix. We can verify that $w^\top A w = \widehat{w}^\top \Sigma \widehat{w}$ with $\widehat{w} = U^\top w$. In other words, in a difference coordinate system defined by U, we are dealing with a quadratic form with a diagonal matrix Σ as the coefficient. Note the diagonalization technique here is only used for analysis.

Therefore, we assume that $A = \text{diag}(\lambda_1, \dots, \lambda_d)$ with $\lambda_1 \geq \cdots \geq \lambda_d$. The function can be simplified to

$$f(w) = \frac{1}{2} \sum_{i=1}^{d} \lambda_i w_i^2$$

The gradient descent update can be written as

$$x \leftarrow w - \eta \nabla f(w) = w - \eta \Sigma w$$

Here we omit the subscript t for the time step and use the subscript for coordinate. Equivalently, we can write the per-coordinate update rule

$$w_i \leftarrow w_i - \eta \lambda_i w_i = (1 - \lambda_i \eta_i) w_i$$

Now we see that if $\eta > 2/\lambda_i$ for some i, then the absolute value of w_i will blow up exponentially and lead to an instable behavior. Thus, we need $\eta \lesssim \frac{1}{\max \lambda_i}$. Note that $\max \lambda_i$ corresponds to the smoothness parameter of f because λ_1 is the largest eigenvalue of $\nabla^2 f = A$. This is consistent with the condition in Lemma 2.1.4 that η needs to be small.

Suppose for simplicity we set $\eta = 1/(2\lambda_1)$, then we see that the convergence for the w_1 coordinate is very fast — the coordinate w_1 is halved every iteration. However, the convergence of the coordinate w_d is slower, because it's only reduced by a factor of $(1 - \lambda_d/(2\lambda_1))$ every iteration. Therefore, it takes $O(\lambda_1/\lambda_d \cdot \log(1/\epsilon))$ iterations to converge to an error ϵ. The analysis here can be extended to general convex function, which also reflects the principle that:

The condition number is defined as $\kappa = \sigma_{\max}(A)/\sigma_{\min}(A) = \lambda_1/\lambda_d$. It governs the convergence rate of GD.

≪Tengyu notes: add figure≫

2.4.1 Pre-conditioners

From the toy quadratic example above, we can see that it would be ideal if we could use a different learning rate for different coordinate. In other words, if we introduce a learning rate $\eta_i = 1/\lambda_i$ for each coordinate, then we can achieve faster convergence. In the more general setting where A is not diagonal, we don't know the coordinate system in advance, and the algorithm corresponds to

$$w \leftarrow w - A^{-1} \nabla f(w)$$

In the even more general setting where f is not quadratic, this corresponds to the Newton's algorithm

$$w \leftarrow w - \nabla^2 f(w)^{-1} \nabla f(w)$$

Computing the hessian $\nabla^2 f(w)$ can be computational difficult because it scales quadratically in d (which can be more than 1 million in practice). Therefore, approximation of the hessian and its inverse is used:

$$w \leftarrow w - \eta Q(w) \nabla f(w)$$

where $Q(w)$ is supposed to be a good approximation of $\nabla^2 f(w)$, and sometimes is referred to as a pre-conditioner. In practice, often people first approximate $\nabla^2 f(w)$ by a diagonal matrix and then take its inverse. E.g., in Adagrad one uses a weighted sum of recent values of $\text{diag}(\nabla f(w) \nabla f(w)^\top)$ to approximate the Hessian, and then use the inverse of the diagonal matrix as the pre-conditioner (see Section 2.5.4).

2.5 *Convergence rates under smoothness conditions*

As mentioned in Chapter 2, gradient-based methods cannot in general find the optimum value of simple functions such as low-degree polynomials. But we did note that if the function is differentiable and smooth, then with a suitably small learning rate, loss does decrease monotonically so long as the gradient is nonzero. In other words, the process ends up with a *stationary point,* where $\nabla = 0$. This chapter establishes upper bounds on how long it takes to get close to a stationary point. See Chapter 7 for analysis of convergence rate to a stronger type of solution: local optimum.

As usual the objective/loss function is denoted $f(w)$ where $w \in \Re^d$. The procedure has T iterations, and the parameter vectors in these iterations are denoted $w_1, ..., w_T$ respectively. We assume *boundedness*: i.e., there is a known M such that $|f(w_t)| \leq \frac{M}{2}$ for all $t = 1, \ldots, T$. We also assume f is β-smooth, i.e.,

$$f(w) \leq f(w') + \nabla f(w')(w - w') + \frac{\beta}{2}\|w - w'\|^2. \tag{2.3}$$

Throughout the chapter, ∇_t is shorthand for $\nabla f(w_t)$.

This β is the same as L elsewhere in the chapter.

2.5.1 *Lower bounds, and the need for smoothness*

In constrained non-convex optimization, minimizing the gradient presents difficult computational challenges. In general, even when objective functions are bounded, local information may provide no information about the location of a stationary point.

Consider, for example, the function sketched in Figure 2.2. In this construction, defined on the hypercube in $\mathbb{R}^n$, the unique point with a vanishing gradient is a hidden valley, and gradients outside this valley are all identical. Clearly, it is hopeless in an information-theoretic sense to find this point efficiently: the number of value or gradient evaluations of this function must be $\exp(\Omega(n))$ to discover the valley.

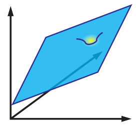


Figure 2.2: A difficult "needle in a haystack" case for non-convex optimization. A function with a hidden valley, with small gradients shown in yellow.

To circumvent such inherently difficult and degenerate cases, we require that the objective function be smooth. As we shall see, this allows efficient algorithms for finding a point with small gradient.

2.5.2 *Convergence rates for GD*

This section analyses gradient descent given exact gradient. Next section analyses stochastic GD.

Algorithm 1 Gradient descent

```
Input: $f$, $T$, initial point $w_1 \in \mathrm{K}$, sequence of step sizes $\{\eta_t\}$
for $t = 1$ to $T$ do
    Let $w_{t+1} = w_t - \eta_t \nabla f(w_t)$
end for
return $w_\tau, \tau \in [T]$ s.t. $\nabla_\tau$ is smallest in Euclidean norm.
```

Theorem 2.5.1. *For unconstrained minimization of β-smooth functions and $\eta_t = \frac{1}{\beta}$, Algorithm 1 satisfies*

$$\|\nabla_\tau\|^2 \leq \frac{1}{T}\sum_t \|\nabla_t\|^2 \leq \frac{4M\beta}{T}.$$

Proof. Denote $h_t = f(w_t) - f(w^*)$. The **Descent Lemma** is given in the following simple equation,

$$\begin{aligned}
h_{t+1} - h_t &= f(w_{t+1}) - f(w_t) \\
&\leq \nabla_t^\top (w_{t+1} - w_t) + \frac{\beta}{2}\|w_{t+1} - w_t\|^2 && \beta\text{-smoothness} \\
&= -\eta_t\|\nabla_t\|^2 + \frac{\beta}{2}\eta_t^2\|\nabla_t\|^2 && \text{algorithm defn.} \\
&= -\frac{1}{2\beta}\|\nabla_t\|^2 && \text{choice of } \eta_t = \frac{1}{\beta}
\end{aligned}$$

Thus, summing up over T iterations, we have

$$\frac{1}{2\beta}\sum_{t=1}^{T}\|\nabla_t\|^2 \leq \sum_t (h_t - h_{t+1}) = h_1 - h_{T+1} \leq 2M$$

□

2.5.3 *Stochastic gradient descent*

In optimization for machine learning, the objective function f takes the form

$$f(w) = \frac{1}{m}\sum_i \ell(w, z_i),$$

where $z_i, i \in [m]$ are the training set examples, and ℓ is some loss function that applies to the parameters w and datapoint z_i. The key idea of Stochastic Gradient Descent is that a random variable can be used in lieu of the gradient, that has the same expectation. This random variable is simply the average gradient of small *batch* of examples from the training set. The analysis below even allows batch size 1 (see Problem 2.5.3).

We denote by $\widehat{\nabla}_t$ a random variable such that $\mathbb{E}[\widehat{\nabla}_t] = \nabla f(w_t) = \nabla_t$ (where expectation is over randomness used in gradient estimation) and a bound on the second moment of this random variable by

$$\mathbb{E}[\|\widehat{\nabla}_t\|^2] = \sigma^2. \tag{2.4}$$

Algorithm 2 Stochastic gradient descent

```
Input: $f$, $T$, initial point $w_1 \in$ K, sequence of step sizes $\{\eta_t\}$
for $t = 1$ to $T$ do
    Let $w_{t+1} = w_t - \eta_t \widehat{\nabla}_t$
end for
return $w_\tau, \tau \in [T]$ s.t. $\nabla_\tau$ is smallest in Euclidean norm.
```

Theorem 2.5.2. *For unconstrained minimization of β-smooth functions and $\eta_t = \eta = \sqrt{\frac{M}{\beta\sigma^2 T}}$, Algorithm 2 satisfies*

$$\mathbb{E}[\|\nabla_\tau\|^2] \leq \mathbb{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq 2\sqrt{\frac{M\beta\sigma^2}{T}}.$$

Proof. Denote by ∇_t the shorthand for $\nabla f(w_t)$, and $h_t = f(w_t) - f(w^*)$. The stochastic descent lemma is given in the following equation,

$$\begin{aligned} \mathbb{E}[h_{t+1} - h_t] &= \mathbb{E}[f(w_{t+1}) - f(w_t)] \\ &\le \mathbb{E}[\nabla_t^\top (w_{t+1} - w_t) + \frac{\beta}{2}\|w_{t+1} - w_t\|^2] && \beta\text{-smoothness} \\ &= -\,\mathbb{E}[\eta \nabla_t^\top \widetilde{\nabla}_t] + \frac{\beta}{2}\eta^2\,\mathbb{E}\,\|\widetilde{\nabla}_t\|^2 && \text{algorithm defn.} \\ &= -\eta\|\nabla_t\|^2 + \frac{\beta}{2}\eta^2\sigma^2 && \text{variance bound.} \end{aligned}$$

Thus, summing up over T iterations, we have for $\eta = \sqrt{\frac{M}{\beta\sigma^2 T}}$,

$$\begin{aligned} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\|\nabla_t\|^2\right] &\le \tfrac{1}{T\eta}\textstyle\sum_t \mathbb{E}\left[h_t - h_{t+1}\right] + \eta\tfrac{\beta}{2}\sigma^2 \le \tfrac{M}{T\eta} + \eta\tfrac{\beta}{2}\sigma^2 \\ &= \sqrt{\tfrac{M\beta\sigma^2}{T}} + \tfrac{1}{2}\sqrt{\tfrac{M\beta\sigma^2}{T}} \le 2\sqrt{\tfrac{M\beta\sigma^2}{T}}. \end{aligned}$$

□

We thus conclude that $O(\frac{1}{\epsilon^4})$ iterations are needed to find a point with $\|\nabla f(w)\| \le \epsilon$. However, each iteration only needs a stochastic estimate of the gradient, so in practice SGD ends up being much faster.

Problem 2.5.3. *Suppose the gradient is estimated using a random sample of B datapoints. (a) Let $\widetilde{\nabla}_t^{(B)}$ be the stochastic gradient at time t when the batchsize is B. Suppose the variance of $\widetilde{\nabla}_t^{(1)}$ (defined as $\mathbb{E}\left[\left\|\widetilde{\nabla}_t^{(1)} - \nabla_t\right\|^2\right]$ is bounded by γ_1^2. Show that there exists an upper bound γ_B^2 on the variance of $\widetilde{\nabla}_t^{(B)}$ that scales with $1/B$. (b) Compute the asymptotic size of T to find a point with $\|\nabla f(w)\| \le \epsilon$ depending on B and ϵ. For simplicity, you only need to consider the case when $\eta \le \frac{1}{\beta}$.*

2.5.4 *Adaptive Algorithms and AdaGrad*

Adaptive methods maintain some information from past updates and use them to modify the basic gradient step. A simple example, *momentum*, was briefly discussed in Chapter 2. Adaptive methods require more space to store their parameters, usually 2 or 3 parameters for each of the d coordinates in the gradient. But they can have faster convergence, as well as other myterious properties in deep learning setting that are not mathematically understood.

TBD: DESCRIBE RMSPROP, ADAM

Several of these algorithms are not guaranteed to converge even for convex loss. We analyse AdaGrad [4], which was a precursor to modern adaptive algorithms and does have a proof of convergence.

[4] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011

Algorithm 3 AdaGrad

```
for $t = 1$ to $T$ do
    Input: Matrices/scalars $P_t$, as below
    Set
        $w_{t+1} = w_t - P_t \hat{\nabla}_t$
end for
return $w_\tau, \tau \in [T]$ s.t. $\nabla_\tau$ is smallest in Euclidean norm.
```

We start with a simple analysis of an adaptive stepsize first, as per the following theorem. In this section, in addition to the aforementioned notation, we also use the shorthand notation $\nabla_{1:t} = \sum_{i=1}^{t} \nabla_i$, and let $G \geq \|\nabla_t\|$ be an upper bound on the gradient norm.

Theorem 2.5.4. *For unconstrained minimization of β-smooth functions and $P_t = \|\hat{\nabla}^2_{1:t}\|^{-1} \cdot I$, Algorithm 3 satisfies*

$$\mathbb{E}[\|\nabla_\tau\|^2] \leq \mathbb{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq \frac{(\beta + M \log GT) \cdot \|\hat{\nabla}^2_{1:t-1}\|}{T}.$$

Proof. From the descent lemma:

$$\begin{aligned} -M &\leq f(w_{T+1}) - f(w_1) \\ &= \textstyle\sum_t (f(w_{t+1}) - f(w_t)) \\ &\leq \textstyle\sum_t (\nabla_t^\top (w_{t+1} - w_t) + \frac{\beta}{2}\|w_t - w_{t+1}\|^2) \quad \text{smoothness} \\ &\leq \textstyle\sum_t (-\nabla_t^\top P_t \hat{\nabla}_t + \frac{\beta}{2} \hat{\nabla}_t^\top P_t^2 \hat{\nabla}_t) \end{aligned}$$

Let $P_t = \|\hat{\nabla}^2_{1:t}\|^{-1}$, and let $\sigma^2 \geq \|\hat{\nabla}_t\|^2$ be an upper bound on the second moment of the stochastic gradient. Then notice that by the Harmonic series,

$$\sum_t \hat{\nabla}_t^\top P_t^2 \hat{\nabla}_t = \sum_t \frac{\|\hat{\nabla}_t\|^2}{\|\hat{\nabla}^2_{1:t}\|^2} = \sum_t \frac{\|\hat{\nabla}_t\|^2}{\sum_{i=1}^{t} \|\hat{\nabla}^2_i\|^2} \leq \log GT$$

Using this inequality in the previous derivation, we get that

$$\sum_t \nabla_t^\top P_t \hat{\nabla}_t \leq M + \frac{\beta}{2} \log GT.$$

Taking the minimal valued LHS, we get

$$\nabla_\tau^\top \hat{\nabla}_\tau \cdot P_{\tau-1} \leq \nabla_\tau^\top \hat{\nabla}_\tau \cdot P_\tau \leq \frac{(\beta + M \log GT)}{T}.$$

Taking expectation over the unbiased gradient estimator, and shifting sides, we get

$$\|\nabla_\tau\|^2 \leq \frac{(\beta + M \log GT) \cdot \|\hat{\nabla}^2_{1:t-1}\|}{T}.$$

□

2.5.5 Adagrad Convergence: Diagonal Matrix case

Theorem 2.5.5. *For unconstrained minimization of β-smooth functions and $P_t = diag(\sum_{i=1}^{t-1} \widehat{\nabla}_i \widehat{\nabla}_i^\top + \sigma^2 I)^{-1/2}$, Algorithm 3 satisfies*

$$\mathbb{E}[\|\nabla_\tau\|^2] \le \mathbb{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \le (M+\beta\log GT)\cdot \frac{\sum_j \sqrt{\widehat{\nabla}^2_{1:t}(j)}}{T}.$$

Proof. From the descent lemma:

$$\begin{aligned} M \quad & \ge f(w_1) - f(w_{T+1}) \\ & = \textstyle\sum_t (f(w_t) - f(w_{t+1})) \\ & \ge \textstyle\sum_t (\nabla_t^\top (w_t - w_{t+1}) - \frac{\beta}{2}\|w_t - w_{t+1}\|^2) \quad \text{smoothness} \\ & = \textstyle\sum_t (\nabla_t^\top P_t \widehat{\nabla}_t - \frac{\beta}{2} \widehat{\nabla}_t^\top P_t^2 \widehat{\nabla}_t). \end{aligned}$$

Taking conditional expectation, and the definition of P_t which is conditionally independent of $\widehat{\nabla}_t$, we get

$$\begin{aligned} M \quad & \ge \textstyle\sum_{i=1}^d \left\{ \sum_t (\nabla_t^2(i) P_t(i) - \frac{\beta}{2} \widehat{\nabla}_t^2(i) P_t^2(i)) \right\} \\ & \ge \textstyle\sum_{i=1}^d \left\{ \sum_t (\nabla_t^2(i) P_t(i) - \frac{\beta}{2} \log \sigma^2 T) \right\} \\ & \ge \textstyle\max_{i=1}^d \sum_t \nabla_t^2(i) P_t(i) - \frac{\beta}{2} \log \sigma^2 T, \end{aligned}$$

where the second inequality is due to the Harmonic series,

$$\sum_t \widehat{\nabla}_t^2(i) P_t^2(i) = \sum_t \frac{\widehat{\nabla}_t^2(i)}{\widehat{\nabla}^2_{1:t-1}(i) + \sigma^2} \le \sum_t \frac{\widehat{\nabla}_t^2(i)}{\widehat{\nabla}^2_{1:t}(i)} \le \log \sigma^2 T.$$

We conclude that any j,

$$\sum_t \nabla_t^2(j) P_t(j) \le \max_i \sum_t \nabla_t^2(i) P_t(i) \le M + \frac{\beta}{2} \log \sigma^2 T.$$

Let c_j be a random variable which is equal to $\nabla_t^2(j)$ with probability $\frac{1}{T}$. Then the above implies that

$$\mathbb{E}[c_j] \le \frac{M + \frac{\beta}{2}\log\sigma^2 T}{T P_t(j)} = (M+\beta\log GT)\cdot \frac{\sqrt{\widehat{\nabla}^2_{1:t}(j)}}{T}.$$

Thus, summing over the coordinates j, we get

$$\mathbb{E}\,\|\nabla_\tau^2\|^2 = \mathbb{E}[\sum_j c_j] \le (M+\beta\log GT)\cdot \frac{\sum_j \sqrt{\widehat{\nabla}^2_{1:t}(j)}}{T}.$$

□

2.6 Correspondence of theory with practice

Now we describe how the above theory compares with reality. Turns out that the assumption of a fixed and known smoothness (used also in Chapters 6, 7, and various other places) is problematic in today's deep learning settings.

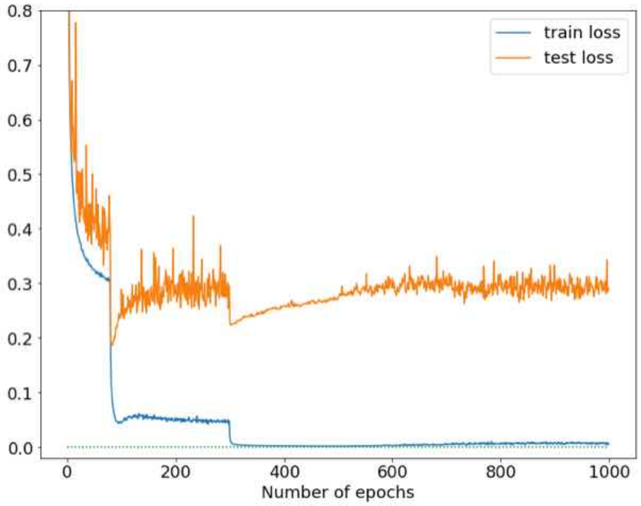


Figure 2.3: How train and test loss behaved during SGD on PreResNet32 on CIFAR10 (50k datapoints). The test loss was estimated at various steps via a fixed held-out dataset. Initial learning rate was 1, and it was reduced by a factor of 10 at epoch 80 and epoch 300. (An epoch is a full pass over the training dataset, where the dataset has been randomly partitioned into batches for use in SGDs–in this case the batchses were of size 128.) Note the complicated relationship between train and test loss, in particular, a slight rise in test loss can happen even when training loss is flat or going down.

Setting learning rate Various algorithms in this chapter set learning rate using smoothness parameter. Unfortunately, it is not easy to estimate the smoothness parameter for the loss function over the *entire* space of parameter vectors. For a particular parameter vector w however it is possible to estimate the maximum eigenvalue of the Hessian $H = \nabla^2(f)$ at w. This uses the *power method*, which starts with a gaussian unit vector u and repeatedly computes $u \leftarrow Hu/\|Hu\|_2$. (Each iteration is efficiently implemented thanks to the Hessian-vector product computation described in Chapter 4.) This method is called the power method because it effectively amounts to computing $H^t x/\|H^t x\|_2$, which can be easily checked to converge to a vector that is a combination of the eigenvectors corresponding to the largest eigenvalue (in absolute value) of H. In particular, if v is the final vector, $v^T Hv$ would be a good approximation to the smoothness.

Of course, this only yields the smoothness parameter for a particular parameter vector w. As w changes, the smoothness can change too. Recomputing smoothness frequently would be computationally expensive. In practice the learning rate is set heuristically to some value. If GD does not lower the loss for a few iterations then learning rate is reduced by a small factor, like 2 or 5. In later chapters we will revisit the topic of learning rates.

Edge of stability phenomenon. The exposition of learning rates given above is canonical in classical optimization theory—it takes smoothness L as given and describes how learning rate must be set less than $2/L$ to ensure consistent decrease in the loss. A recent paper [5] gives evidence that in deep nets the cart appears before the horse, so to

[5] Jeremy Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. *ICLR*, 2021

speak. In other words, if we set the learning rate to some small η, the smoothness *adjusts* quickly to around $2/\eta$. This "edge of stability"phase appears to be important for good final performance.

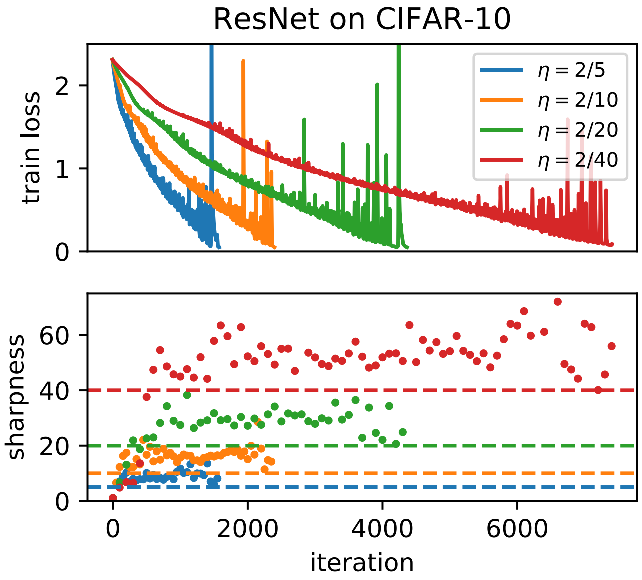


Figure 2.4: **Edge of stability phenomenon.** Figure 5 in Cohen et al. 2021 shows a ResNet trained on $5k$ examples. When doing GD (as opposed to SGD) with a small learning rate η, the smoothness is observed to rise to $2/\eta$ and slightly beyond (figure on right). After this point one sees loss go up and down during iterations, with a long-term downward trend. No theoretical explanation is known as of now. The authors use "sharpness"instead of smoothness, which actually makes some sense because higher L corresponds to a more uneven landscape.

3
Note on overparametrized linear regression and kernel regression

This brief section analyzes gradient descent for a very classic model: least-squares linear regression. The problem is convex, and optimization does work well. We are interested primarily in the underdetermined version, where one has infinitely many zero-loss solution and the interesting question is: what does gradient descent find? We find an elegant exact analysis using pseudo-inverse. The analysis also extends to Kernel least squares regression.

Though classical, these analyses are the starting point for efforts (described in Chapters 9 and 8)) to understand over-parametrized deep nets, which also have an abundance of low cost solutions, and we wish to understand which ones are found by gradient descent and related algorithms.

3.1 Overparametrized least squares linear regression

As in Chapter 1 we assume our training data consist of n data points, each drawn independently from a distribution $\mathcal{D}$,

$$(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \overset{\text{i.i.d.}}{\sim} \mathcal{D}.$$

Here $x^{(i)} \in \Re^d$ and $y^{(i)} \in \Re$. It will be convenient to write the ℓ_2 loss

$$\widehat{L}(w) = \frac{1}{2} \sum_i (x^{(i)} \cdot w - y^{(i)})^2,$$

using matrix notation as $\widehat{L}(w) = \frac{1}{2}\|Xw - y\|^2$ where X is the matrix whose rows are the $x^{(i)}$'s, w is a column vector and y is the column vector whose ith entry is $y^{(i)}$. We're interested in the case where $x^{(i)}$'s are independent and $d > n$. Then the loss has infinitely many minimizers[1], all of which attain zero training loss. What does gradient descent find?

[1] You can verify this by noticing that a feasible solution exists after setting an arbitrary subset of $d - n$ coordinates in w to zero.

For simplicity, initialize gradient descent with starting point $w_0 = 0$. The gradient at any w is $\nabla\widehat{L}(w) = X^T(Xw - y)$. Thus gradient descent with learning rate η gives the following trajectory

$$\begin{aligned} w_{t+1} &= w_t - \eta X^\top(Xw_t - y) && (3.1)\\ &= (I_{d\times d} - \eta X^\top X)w_t + \eta X^\top y && (3.2)\\ &= (\sum_{j=0}^{t}(I_{d\times d} - \eta X^\top X))^j \eta X^\top y && (3.3)\end{aligned}$$

Assuming η was small enough by trial and error, specifically, $\eta < 1/\lambda_{max}(X^T X)$, the infinite series as $t \to \infty$ in the above equation converges to [2]

$$\begin{aligned} \lim_{t\to\infty} w_t &= (X^\top X)^\dagger X^\top y && (3.4)\\ &= X^\top (XX^\top)^{-1} y && (3.5)\end{aligned}$$

which of course is the famous *pseudo-inverse* solution for overdetermined systems of linear equations. This solution is also the minimizing ℓ_2-norm solution that fits the data: $\arg\min_{w\in\Re^d} \|w\|_2$ s.t. $Xw = y$.

3.1.1 SVD and Matrix pseudo-inverse

The inverse of a matrix is defined only for the square matrices with full rank. The above example illustrates that we need notions of inverse for non-square matrices as well as for rank-deficient square matrices. The *Moore-Penrose* pseudo-inverse was defined in the 20th century with these motivations. For simplicity we describe this theory for real $m \times n$ matrices, which are known to have a *singular value decomposition* (SVD) of the form:

$$M = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \qquad (3.6)$$

where k is the rank of M, the σ_i's are the singular values, and $u_i \in \Re^m$, $v_i \in \Re^n$ are column vectors.

The *pseudo-inverse* is the $n \times m$ matrix $M^\dagger$ defined as follows, where the notation assumes all σ_i's are nonzero:

$$M^\dagger = \sum_{i=1}^{k} \frac{1}{\sigma_i} v_i u_i^T \qquad (3.7)$$

A special case is when M is symmetric $m \times m$, in which case the SVD has $v_i = u_i$ and called the *spectral decomposition*.

Problem 3.1.1. *Show that $MM^\dagger M = M$ and $M^\dagger M M^\dagger = M^\dagger$.*

Problem 3.1.2. *Prove the properties mentioned in (3.5).* [3]

[2] We're using that $\sum_{i\geq 0} A^i = (I - A)^\dagger$ when the largest eigenvalue of positive semidefinite matrix A is less than 1 and $Z^\dagger$ denotes the pseudo inverse of Z. Further, the second equality in eq. (3.5) can be verified by using the SVD of X.

[3] Hint: if M is symmetric, the eigenvalues of M^j are jth powers of the eigenvalues of M. Also, the eigenvectors constitute an orthonormal basis.

3.2 Kernel least-squares regression

Kernel models involve a new representation for the data space $\mathcal{X}$, which represents datapoint $x \in \mathcal{X}$ as $\phi(x)$ where ϕ is a mapping from $\Re^d$ to a suitable Hilbert space [4] known as the "Reproducing Kernel Hilbert Space"or RKHS. This means that the inner product $\phi(x) \cdot \phi(y)$ is well-defined for every $x, y \in \mathcal{X}$. It is customary to denote the inner product as $K(x, y)$, which is called the kernel function. The function class of interest are linear models over the transformed features $h_w(x) = \phi(x) \cdot w$. A plethora of useful kernels are known in mathematics and data science.

Example 3.2.1. *The Kernel $K(x, y) = (1 + x \cdot y)^2$ corresponds to representing $x = (x_1, x_2, \ldots, x_d)$ by*

$$\phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_d, \sqrt{2}x_2x_3, \ldots, \sqrt{2}x_2x_d, \ldots, \sqrt{2}x_{d-1}x_d, x_1^2, x_2^2, \ldots, x_d^2).$$

You can verify that $\phi(x) \cdot \phi(y) = K(x, y)$.

In data science the key property needed is that the inner product $K(x_1, x_2)$ be efficiently computable. In fact in practice researchers design the kernel by starting with this property of efficient computability and never consider the underlying representation $\phi(\cdot)$ because that plays no role in the training.

Problem 3.2.2. *Suppose datapoints are unit vectors in $\Re^d$. Find an infinite dimensional representation $\varphi()$ that realizes the following kernels. (a) (polynomial kernel) $K(x_1, x_2) = (1 + (x \cdot y)^d)$. (b) (Gaussian Kernel) $K(x_1, x_2) = \exp(-\|x - y\|^2)$. (c) (Laplace Kernel) $K(x_1, x_2) =$ $\exp(\|x_1 - x_2\|_2) = \exp(\sqrt{1 - 2x_1 \cdot x_2})$. (Hint for (b) and (c): look at the Taylor expansion of $K()$.)*

Now let's see how to solve the following kernel regression efficiently.

$$\ell(w) = \frac{1}{2} \sum_i (\phi(x)^{(i)} \cdot w - y^{(i)})^2.$$

While $\phi(x)$ is infinite dimensional, we quickly realize that the earlier analysis of over-parametrized regression applies. Computing data matrix $XX^\top$ requires only inner products, which are well-defined in RKHS and so the data matrix turns into an $n \times n$ *gram matrix* G where $G_{ij} = \phi(x^{(i)}) \cdot \phi(x^{(j)}) = K(x^{(i)}, x^{(j)})$ In fact computing G allows performing gradient descent without explicitly computing $\phi(x^{(i)})$.Expression (3.5) shows gradient descent ends with a classifier $h(x) = \lim_{t \to \infty} w_t . \phi(x)$ that maps an input point $x \in \mathcal{X}$ to

$$h(x) = z^T G^{-1} y \tag{3.8}$$

[4] Hilbert space is the generalization of vector spaces to infinite dimensions, with inner products being well-defined.

where z is the column vector whose ith coordinate is $K(x, x_i)$. Alternatively, denoting $\alpha = G^{-1}y \in \Re^n$, the solution in eq. (3.8) is alternatively viewed as a weighted combinations of kernel evaluations at training points,

$$h(x) = \sum_i \alpha_i K(x, x_i). \tag{3.9}$$

Expression in eq. (3.9) is equivalent to minimizing the ℓ_2 norm of w while fitting the kernel regression objective, which viewed in the function space corresponds to the minimum RKHS norm solution wrt the kernel K.

4
Note on Backpropagation and its Variants

Throughout the book we rely on computing the gradient of the loss with respect to model parameters. For deep nets, this computation is done with Backpropagation, a simple algorithm that uses the chain rule of calculus. For convenience we describe this more generally as a way to compute the sensitivity of the output of a neural network to all of its parameters, namely, $\partial f / \partial w_i$, where f is the output and w_i is the ith parameter. Here *parameters* can be edge weights or biases associated with nodes or edges of the network. Versions of this basic algorithm have been apparently independently rediscovered several times from 1960s to 1980s in several fields. This chapter introduces this algorithms as well as some advanced variants involving not just the gradient but also the Hessian.

In most of the book, the quantity of interest is the gradient of the training loss. But the above phrasing —computing gradient of the output with respect to the inputs—is fully general since one can simply add a new output node to the network that computes the training loss from the old output. Then the quantity of interest is indeed the gradient of this new output with respect to network parameters.

The importance of backpropagation derives from its efficiency. Assuming node operations take unit time, the running time is *linear*, specifically, $O(\text{Network Size}) = O(V + E)$, where V is the number of nodes in the network and E is the number of edges. As in many other settings in computer science —for example, sorting numbers— the naive algorithm would take quadratic time, and that would be hugely inefficient or even infeasible for today's large networks.

4.1 Problem Setup

Backpropagation applies only to acyclic networks with directed edges. (It can be heuristically applied to networks with cycles, as sketched later.) Without loss of generality, acyclic networks can be visualized as being structured in numbered layers, with nodes in the $t+1$th layer getting all their inputs from the outputs of nodes in layers t and earlier. We use $f \in \mathbb{R}$ to denote the output of the network. In all our figures, the input of the network is at the bottom and the output on the top.

Our exposition uses the notion $\partial f/\partial u$, where f is the output and u is a node in the net. This means the following: suppose we cut off all the incoming edges of the node u, and fix/clamp the current values of all network parameters. Now imagine changing u from its current value. This change may affect values of nodes at higher levels that are connected to u, and the final output f is one such node. Then $\partial f/\partial u$ denotes the rate at which f will change as we vary u. (Aside: Readers familiar with the usual exposition of back-propagation should note that there f is the training error and this $\partial f/\partial u$ turns out to be exactly the "error" propagated back to on the node u.)

Claim 4.1.1. *To compute the desired gradient with respect to the parameters, it suffices to compute $\partial f/\partial u$ for every node u.*

Proof. Follows from direct application of chain rule and we prove it by picture, namely Figure 4.1. Suppose node u is a weighted sum of the nodes $z_1, \dots, z_m$ (which will be passed through a non-linear activation σ afterwards). That is, we have $u = w_1 z_1 + \cdots + w_n z_n$. By Chain rule, we have

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial w_1} = \frac{\partial f}{\partial u} \cdot z_1.$$

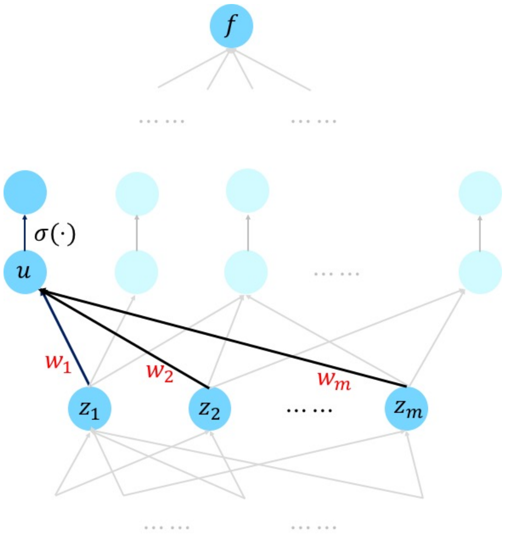


Figure 4.1: Why it suffices to compute derivatives with respect to nodes.

Hence, we see that having computed $\partial f/\partial u$ we can compute

$\partial f / \partial w_1$, and moreover this can be done locally by the endpoints of the edge where w_1 resides. □

4.1.1 *Multivariate Chain Rule*

Towards computing the derivatives with respect to the nodes, we first recall the multivariate Chain rule, which handily describes the relationships between these partial derivatives (depending on the graph structure).

Suppose a variable f is a function of variables $u_1, \ldots, u_n$, which in turn depend on the variable z. Then, multivariate Chain rule says that

$$\frac{\partial f}{\partial z} = \sum_{j=1}^{n} \frac{\partial f}{\partial u_j} \cdot \frac{\partial u_j}{\partial z} \ .$$

To illustrate, in Figure 4.2 we apply it to the same example as we used before but with a different focus and numbering of the nodes.

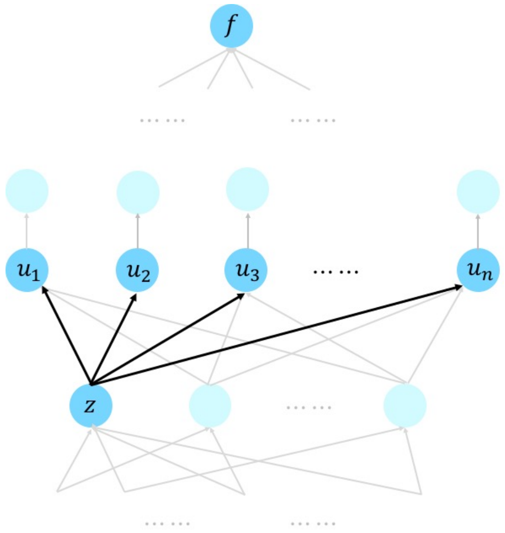


Figure 4.2: Multivariate chain rule: derivative with respect to node z can be computed using weighted sum of derivatives with respect to all nodes that z feeds into.

We see that given we've computed the derivatives with respect to all the nodes that is above the node z, we can compute the derivative with respect to the node z via a weighted sum, where the weights involve the local derivative $\partial u_j / \partial z$ that is often easy to compute. This brings us to the question of how we measure running time. For book-keeping, we assume that

Basic assumption: If u is a node at level $t + 1$ and z is any node at level $\leq t$ whose output is an input to u, then computing $\frac{\partial u}{\partial z}$ takes unit time on our computer.

4.1.2 *Naive feedforward algorithm (not efficient!)*

It is useful to first point out the naive quadratic time algorithm implied by the chain rule. Most authors skip this trivial version, which we think is analogous to teaching sorting using only quicksort, and skipping over the less efficient bubblesort.

The naive algorithm is to compute $\partial u_i/\partial u_j$ for every pair of nodes where u_i is at a higher level than u_j. Of course, among these V^2 values (where V is the number of nodes) are also the desired $\partial f/\partial u_i$ for all i since f is itself the value of the output node.

This computation can be done in feedforward fashion. If such value has been obtained for every u_j on the level up to and including level t, then one can express (by inspecting the multivariate chain rule) the value $\partial u_\ell/\partial u_j$ for some u_ℓ at level $t+1$ as a weighted combination of values $\partial u_i/\partial u_j$ for each u_i that is a direct input to u_ℓ. This description shows that the amount of computation for a fixed j is proportional to the number of edges E. This amount of work happens for all $j \in V$, letting us conclude that the total work in the algorithm is $O(VE)$.

4.2 *Backpropagation (Linear Time)*

The more efficient backpropagation, as the name suggests, computes the partial derivatives in the reverse direction. Messages are passed in one wave backwards from higher number layers to lower number layers. (Some presentations of the algorithm describe it as dynamic programming.)

Algorithm 4 Backpropagation

The node u receives a message along each outgoing edge from the node at the other end of that edge. It sums these messages to get a number S (if u is the output of the entire net, then define $S = 1$) and then it sends the following message to any node z adjacent to it at a lower level:

$$S \cdot \frac{\partial u}{\partial z}$$

Clearly, the amount of work done by each node is proportional to its degree, and thus overall work is the sum of the node degrees. Summing all node degrees ends up double-counting eac edge, and thus the overall work is $O(\text{Network Size})$.

To prove correctness, we prove the following:

Lemma 4.2.1. *At each node z, the value S is exactly $\partial f/\partial z$.*

Proof. Follows from simple induction on depth.

Base Case: At the output layer this is true, since $\partial f / \partial f = 1$.
Inductive step: Suppose the claim was true for layers $t+1$ and higher and u is at layer t, with outgoing edges go to some nodes $u_1, u_2, \ldots, u_m$ at levels $t+1$ or higher. By inductive hypothesis, node z indeed receives $\frac{\partial f}{\partial u_j} \times \frac{\partial u_j}{\partial z}$ from each of u_j. Thus by Chain rule,

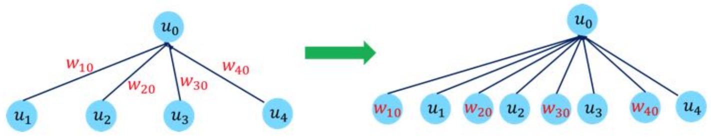
$$S = \sum_{i=1}^{m} \frac{\partial f}{\partial u_i} \frac{\partial u_i}{\partial z} = \frac{\partial f}{\partial z}.$$

This completes the induction and proves the Main Claim. □

4.3 *Auto-differentiation*

Since the exposition above used almost no details about the network and the operations that the node perform, it extends to every computation that can be organized as an acyclic graph whose each node computes a differentiable function of its incoming neighbors. This observation underlies many auto-differentiation packages found in deep learning environments: they allow computing the gradient of the output of such a computation with respect to the network parameters.

We first observe that Claim 4.1.1 continues to hold in this very general setting. This is without loss of generality because we can view the parameters associated to the edges as also sitting on the nodes (actually, leaf nodes). This can be done via a simple transformation to the network; for a single node it is shown in the picture below; and one would need to continue to do this transformation in the rest of the networks feeding into $u_1, u_2, ..$ etc from below.



Then, we can use the messaging protocol to compute the derivatives with respect to the nodes, as long as the local partial derivative can be computed efficiently. We note that the algorithm can be implemented in a fairly modular manner: For every node u, it suffices to specify (a) how it depends on the incoming nodes, say, $z_1, \ldots, z_n$ and (b) how to compute the partial derivative times S, that is, $S \cdot \frac{\partial u}{\partial z_j}$.

Extension to vector messages : In fact (b) can be done efficiently in more general settings where we allow the output of each node in the network to be a vector (or even matrix/tensor) instead of only a real number. Here we need to replace $\frac{\partial u}{\partial z_j} \cdot S$ by $\frac{\partial u}{\partial z_j}[S]$, which denotes the result of applying the operator $\frac{\partial u}{\partial z_j}$ on S. We note that to be consistent with the convention in the usual exposition of backpropagation, when $y \in \mathbb{R}^p$ is a funciton of $x \in \mathbb{R}^q$, we use $\frac{\partial y}{\partial x}$ to denote $q \times p$ dimensional matrix with $\partial y_j / \partial x_i$ as the (i, j)-th entry. Readers might notice that this is the transpose of the usual Jacobian matrix defined in mathematics. Thus $\frac{\partial y}{\partial x}$ is an operator that maps $\mathbb{R}^p$ to $\mathbb{R}^q$ and we can verify S has the same dimension as u and $\frac{\partial u}{\partial z_j}[S]$ has the same dimension as z_j.

For example, as illustrated below, suppose the node $U \in \mathbb{R}^{d_1 \times d_3}$ is a product of two matrices $W \in \mathbb{R}^{d_2 \times d_3}$ and $Z \in \mathbb{R}^{d_1 \times d_2}$. Then we have that $\partial U / \partial Z$ is a linear operator that maps $\mathbb{R}^{d_2 \times d_3}$ to $\mathbb{R}^{d_1 \times d_3}$, which naively requires a matrix representation of dimension $d_2 d_3 \times d_1 d_3$. However, the computation (b) can be done efficiently because

$$\frac{\partial U}{\partial Z}[S] = W^\top S.$$

Such vector operations can also be implemented efficiently using today's GPUs.

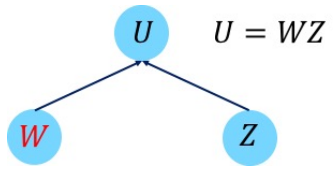


Figure 4.3: Vector version of above

4.4 Notable Extensions

Allowing weight tying: In many neural architectures, the designer wants to force many network units such as edges or nodes to share the same parameter. For example, in including the ubiquitous convolutional net, the same filter has to be applied all over the image, which implies reusing the same parameter for a large set of edges between two layers of the net.

For simplicity, suppose two parameters a and b are supposed to share the same value. This is equivalent to adding a new node u and connecting u to both a and b with the operation $a = u$ and $b = u$. Thus, by chain rule,

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial u} + \frac{\partial f}{\partial b} \cdot \frac{\partial b}{\partial u} = \frac{\partial f}{\partial a} + \frac{\partial f}{\partial b}.$$

Hence, equivalently, the gradient with respect to a shared parameter is the sum of the gradients with respect to individual occurrences.

Backpropagation on networks with loops. The above exposition assumed the network is acyclic. Many cutting-edge applications such as machine translation and language understanding use networks with directed loops (e.g., recurrent neural networks). These architectures —all examples of the "differentiable computing" paradigm below—can get complicated and may involve operations on a separate memory as well as mechanisms to shift attention to different parts of data and memory.

Networks with loops are trained using gradient descent as well, using *back-propagation through time* which consists of expanding the network through a finite number of time steps into an acyclic graph, with replicated copies of the same network. These replicas share the weights (weight tying!) so the gradient can be computed. In practice an issue may arise with *exploding or vanishing gradients,* which impact convergence. Such issues can be carefully addressed in practice by clipping the gradient or re-parameterization techniques such as *long short-term memory.* Recent work suggests that careful initialization of parameters can ameliorate some of the vanishing gradient problems.

The fact that the gradient can be computed efficiently for such general networks with loops has motivated neural net models with memory or even data structures (see for example *neural Turing machines* and *differentiable neural computer*). Using gradient descent, one can optimize over a family of parameterized networks with loops to find the best one that solves a certain computational task (on the training examples). The limits of these ideas are still being explored.

4.4.1 Hessian-vector product in linear time: Werbos-Pearlmutter trick

It is possible to generalize backpropagation to work with 2nd order derivatives, specifically with the Hessian H which is the symmetric matrix whose (i, j) entry is $\partial^2 f / \partial w_i \partial w_j$. Sometimes H is also denoted $\nabla^2 f$. Just writing down this matrix takes quadratic time and memory, which is infeasible for today's deep nets. Surprisingly, using backpropagation it is possible to compute in linear time the matrix-vector product Hx for any vector x. The trick is described by Pearlmutter who attributes it to an earlier work of Werbos [1].

Claim 4.4.1. *Suppose an acyclic network with V nodes and E edges has output f and leaves $z_1, \ldots, z_m$. Then there exists a network of size $O(V +$*

[1] P. J. Werbos. Backpropagation: Past and future. In *IEEE InternationalConference on Neural Networks*, page 343–353, 1988; and Barak Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 1994

E) that has $z_1, \ldots, z_m$ as input nodes and $\frac{\partial f}{\partial z_1}, \ldots, \frac{\partial f}{\partial z_m}$ as output nodes.

The proof of the Claim follows in straightforward fashion from implementing the message passing protocol as an acyclic circuit.

Next we show how to compute $\nabla^2 f(z) \cdot v$ where v is a given fixed vector. Let $g(z) = \langle \nabla f(z), v \rangle$ be a function from $\mathbb{R}^d \to \mathbb{R}$. Then by the Claim above, $g(z)$ can be computed by a network of size $O(V + E)$. Now apply the Claim again on $g(z)$, we obtain that $\nabla g(z)$ can also be computed by a network of size $O(V + E)$.

Note that by construction,

$$\nabla g(z) = \nabla^2 f(z) \cdot v.$$

Hence we have computed the Hessian vector product in network size time.

5
Basics of generalization theory

Recall from Chapter 1 the language of Empirical Risk Minimization from Chapter 1. A datapoint x (for classification this is actually a pairing of a vector and a label) come from a distribution $\mathcal{D}$ and S denotes the training sample. The loss of a hypothesis h on datapoint x is $\ell(x, h)$. (Since hypothesis in deep learning is given by a parameter vector w we may also represent this as $\ell(x, w)$.) In generalization theory we are interested in understanding the relationship between the test loss and the training loss (respectively):

$$L_{\mathcal{D}}(h) = \mathop{\mathbb{E}}_{x \in \mathcal{D}} [\ell(x, h)] \quad \text{and} \quad \widehat{L}_S(h) = \mathop{\mathbb{E}}_{x \in S} [\ell(x, h)]. \tag{5.1}$$

(Here $\widehat{\cdot}$ refers to "empirical." The training is considered a success if $L_S(h)$ is small and the *generalization error* $\Delta_S(h) = L_{\mathcal{D}}(h) - \widehat{L}_S(h)$ is small too.

Generalization theory gives estimates of the number of training samples sufficient to guarantee low generalization error. The classic ideas described in this chapter give very loose (i.e., trivial) estimates for deep learning. We survey attempts to provide tighter estimates.

Generalization theory takes inspiration from an old philosophical principle called *Occam's razor*: given a choice between a simpler theory of science and a more convoluted theory, both of which explain some empirical observations, we should trust the simpler one. For instance, Copernicus's heliocentric theory of the solar system gained favor in science because it explained known facts more simply than the ancient Aristotelian theory. While this makes intuitive sense, Occam's razor is a bit vague and hand-wavy. What makes a theory "simpler" or "better"?

5.1 Occam's razor formalized for ML

The following is the mapping from the above intuitive notions to notions in ML. (For simplicity we focus only on supervised learning here, and consider other settings in later chapters.)

Observations/evidence	↔	Training dataset S
theory	↔	hypothesis h
All possible theories	↔	hypothesis class $\mathcal{H}$
Finding theory to fit observations	↔	Minimize training loss to find $h \in \mathcal{H}$
Theory is good (good predictions in new settings)	↔	h has low test loss
Simpler theory	↔	h has shorter description

The notion of "shorter description"will be formalized in a variety of ways using a *complexity measure* for the class $\mathcal{H}$, denoted $\mathcal{C}(\mathcal{H})$, and use it to upper bound the generalization error.

Let S be a sample of m datapoints. Empirical Risk Minimization (ERM) paradigm (see Chapter 1) involves finding $\widehat{h} = \arg\min \widehat{L_S}(h)$. Of course, in deep learning we may not find the absolute optimum h but in practice the training loss becomes very small and near-zero. Intuitively, if generalization error is large then the hypothesis's performance on training sample S does not accurately reflect the performance on the full distribution of examples, and we say it *overfitted* to the sample S.

The typical upper bound on generalization error [1] shows that with probability at least $1 - \delta$ over the choice of training data, the following

$$\Delta_S(h) \leq \sqrt{\frac{\mathcal{C}(\mathcal{H}) + O(\log(1/\delta))}{m}} \tag{5.2}$$

Thus to drive the generalization error down it suffices to make m significantly larger than the "Complexity Measure". Hence classes with lower complexity require fewer training samples, in line with Occam's intuition.

5.1.1 *Motivation for generalization theory*

Generalization bounds seek to estimate the generalization error using the trained model h and the training dataset S. Students can wonder if the bound is any use if the experimenter has already decided on the architecture, training algorithm etc. Indeed, then the experiment can proceed with training and use a held out dataset to estimate the generalization error.

The hope in developing generalization theory is that it provides insights into how to design architectures and algorithms in the first place so that they result in "low complexity" in the trained net, causing it to generalize well. Clearly, more principled understanding along such lines would be nice.

[1] This is the format of typical generalization bound! In general this chapter focuses on clear exposition of the basic ideas, while being a bit sloppy with constants.

5.1.2 Warmup: Classical polynomial interpolation

Suppose we are given n points $(x^{(1)}, y^{(1)})$, ..., $(x^{(n)}, y^{(n)})$ chosen according to the following probabilistic process: $x^{(i)}$ is chosen from uniform distribution on $[0,1]$ and $y^{(i)} = p(x^{(i)}) + \eta$ where $p()$ is an unknown degree d polynomial and η is a sample from the noise distribution $\mathcal{N}(0, \sigma^2)$. Since $\mathbb{E}[\eta] = 0$ and $\mathbb{E}[\eta^2] = \sigma^2$ the obvious way to find p is to minimize the least square fit to find the polynomial's coefficients $\theta_0, \theta_1, \ldots, \theta_d$:

$$\ell(\vec{\theta}) = \sum_{i=1}^{n} (y^{(i)} - \sum_{j=0} \theta_j (x^{(i)}))^2.$$

This is implicitly doing linear regression using a new data representation, whereby the point $x \in \Re$ is represented using the vector $(1, x, x^2, \ldots, x^d)$.

But what if we don't know d and try to fit a degree N polynomial where $N \gg d$? Under what conditions would minimizing the above loss give us roughly the same polynomial as $p()$? A practical idea —noting the fact that the above loss is $n\sigma^2$ even for the ground truth polynomial $p()$—is to add for some large-ish $\lambda > 0$ the regularizer $\lambda \|\theta\|_2^2$ to the above loss. This signals to gradient descent that it is unimportant to reduce the least squares loss all the way to zero, and instead it should find solutions θ's of low norm. More generally one could use other measures of "complexity" than square of the Euclidean norm.

This example is intuitive and can be analysed more rigorously but requires the theory of orthonormal polynomials with respect to natural distributions on $[0,1]$.

5.2 Some simple upper bounds on generalization error

Recall the *union bound* in elementary probability: every set of events $A_1, A_2, \ldots$ satisfy $\Pr[\cup_i A_i] \leq \sum_i \Pr[A_i]$. Many analyses of generalization leverage this simple fact.

The first example illustrates this in an almost trivial setting. Later we shall see the same idea also at the heart of other generalization bounds[2], albeit often hidden inside the proof. The bound shows that if a hypothesis class contains at most N distinct hypotheses, then $\log N$ (i.e., close to the number of bits needed to represent the index of the hypotheses in this class) is the effective complexity measure in (5.2).

Theorem 5.2.1 (Finite Hypothesis Class). *If the loss function takes values in* $[0,1]$ *and hypothesis class* $\mathcal{H}$ *contains N distinct hypotheses then*

[2] The union bound is also referred to as **uniform convergence framework** in many books. Often the hypothesis class is infinite but the proof discretizes it, as in Theorem 5.2.7.

with probability at least $1 - \delta$, *every* $h \in \mathcal{H}$ *satisfies*

$$\Delta_S(h) \leq 2\sqrt{(\log N + \log(1/\delta))/m}.$$

Proof. For any *fixed* hypothesis g imagine drawing a training sample of size m. Then $\widehat{L_S}(g)$ is an average of i.i.d. variables and its expectation is $L_{\mathcal{D}}(g)$. Concentration bounds imply that $L_{\mathcal{D}}(g) - \widehat{L_S}(g)$ has a concentration property at least as strong as univariate Gaussian $\mathcal{N}(0, 1/m)$. The previous statement is true for all hypotheses g in the class, so the union bound implies that the probability is at most $N \exp(-\epsilon^2 m/4)$ that this quantity exceeds ϵ for *some* hypothesis in the class. Since h is the solution to ERM, we conclude that when $\delta \leq N \exp(-\epsilon^2 m/4)$ then $\Delta_S(h) \leq \epsilon$. Simplifying and eliminating ϵ, we obtain the theorem. □

At first sight the union bound appears useless for deep nets because if the net has k real-valued parameters, the set of hypotheses — even after we have fixed the architecture and number of parameters— consists of all vectors in $\mathbb{R}^k$, an uncountable set!

Example 5.2.2 (Possible way around?)**.** *As we saw in Chapter 2, the end result of gradient descent on the loss is special. For instance it must be a stationary point (i.e., where* $\nabla = 0$*) of the training loss. One can similarly imagine other conditions on Hessian* $\nabla^2()$ *and so forth. One could hope that the set of points in the loss landscape with such properties could be small and finite. This line of investigation hasn't yet worked out because current nets are so overparametrized (i.e., number of parameters far exceeding the number of training data points) that the set of such solution points in the landscape is also in general a continuous set (i.e., uncountable). The next hope is to take into account the training algorithm, because not all solution points may be accessible via standard training algorithms. These are some ideas for restricting attention to a finite set of solutions, though they haven't yet worked out.*

There is a more obvious way to turn the set of possible deep nets into a finite set: *discretization!* Suppose we assume that the ℓ_2 norm of the parameter vectors is at most 1, meaning the set of all deep nets has been identified with Ball$(0, 1)$. (Here Ball(w, r) refers to set of all points in $\mathbb{R}^k$ within distance r of w.)

Suppose the loss is Lipschitz in the parameters: for every datapoint x and parameter vectors w_1, w_2 $\|\ell(x, w_1) - \ell(x, w_2)\| \leq C\|w_1 - w_2\|_2$ for some explicit constant C. The arguments below only need local Lipschitz-ness, namely for the condition to hold for $\|w_1 - w_2\|_2 \leq \rho$ for some explicit constant ρ. Furthermore it only requires Lipschitz-ness of the average loss on the training set, not loss on single data points.

Suppose $\rho > 0$ is such that if $w_1, w_2 \in \mathbb{R}^k$ satisfy $\|w_1 - w_2\|_2 \leq \rho$ then the nets with these two parameter vectors have essentially the same loss on every input, meaning the losses differ by at most γ for some $\gamma > 0$. (NB: This amounts to a *local* Lipschitz constant, and it is at most γ/ρ.) It makes intuitive sense such a ρ must exist for every $\gamma > 0$ since as we let $\rho \to 0$ the two nets become equal[3].

Problem 5.2.3. *Compute Lipschitz constant of the ℓ_2 regression loss: the loss on datapoint (x, y) is $(w \cdot x - y)^2$.*

Problem 5.2.4. *Compute Lipschitz constant of ℓ_2 loss for a two layer deep net with ReLU gates (zero bias) on the middle layer. Assume the two parameter vectors are infinitesimally close.*

Definition 5.2.5 (ρ-cover)**.** *A set of points $w_1, w_2, \ldots \in \mathbb{R}^k$ is a* ρ-cover *if for every $w \in \text{Ball}(0, 1)$ there is some w_i such that $w \in \text{Ball}(w_i, \rho)$.*

Lemma 5.2.6 (Existence of ρ-cover)**.** *There exists a ρ-cover of size at most $((2 + \rho)/2\rho)^k$.*

Proof. The proof is simple but clever. Let us pick w_1 arbitrarily in $\text{Ball}(0, 1)$. For $i = 2, 3, \ldots$ do the following: arbitrarily pick any point in $\text{Ball}(0, 1)$ outside $\cup_{j \leq i}\text{Ball}(w_j, \rho)$ and designate it as w_{i+1}.

A priori it is unclear if this process will ever terminate. We now show it does after at most $(2/\rho)^k$ steps. To see this, it suffices to note that $\text{Ball}(w_i, \rho/2) \cap \text{Ball}(w_j, \rho/2) = \emptyset$ for all $i < j$. (Because if not, then $w_j \in \text{Ball}(w_i, \rho)$, which means that w_j could not have been picked during the above process.) Thus we conclude that the process must have stopped after at most

$$\text{volume}(\text{Ball}(0, 1 + \rho/2))/\text{volume}(\text{Ball}(0, \rho/2))$$

iterations[4], which is at most $((2 + \rho)/2\rho)^k$ since ball volume in $\mathbb{R}^k$ scales as the kth power of the radius.

Finally, the sequence of w_i's at the end must be a ρ-cover because the process stops only when no point can be found outside $\cup_j\text{Ball}(w_j, \rho)$. □

Theorem 5.2.7 (Generalization bound for normed spaces)**.** *If (i) hypotheses are unit vectors in $\mathbb{R}^k$ and (ii) every two hypotheses h_1, h_2 with $\|h_1 - h_2\|_2 \leq \rho$ differ in terms of loss on every datapoint by at most γ then* [5]

$$\Delta_S(h) \leq \gamma + 2\sqrt{k \log(2/\rho)/m}.$$

Proof. Apply the union bound on the ρ-cover. Every other net can have loss at most γ higher than nets in the ρ-cover. □

[3] The issue we are ignoring is that ρ, γ may depend upon the size of the training set. This unfortunately appears to be the case in real-life deep learning, which this analysis is ignoring.

[4] The reason for $1 + \rho/2$ in the numerator is that if a w_i lies at the surface of $\text{Ball}(0, 1)$ then the ball of radius $\rho/2$ around it lies in the ball of radius $1 + \rho/2$ around the origin

[5] Even ignoring the dependence on the Lipschitz constant, this bound requires the number of datapoints to grow linearly with the number of trainable parameters in the net. This does not begin to explain the surprising effectiveness of real-life deep learning, where the number of parameters greatly exceeds the number of training datapoints.

5.3 Data dependent complexity measures

Thus far we considered complexity measures for hypothesis classes as a way to quantify their "complicatedness." : the size of the hypothesis class (assuming it is finite) and the size of a γ-cover in it. Of course, the resulting bounds on sample complexity were still loose.

But these simple bounds hold for every data distribution $\mathcal{D}$. In practice, it seems clear that deep nets —or any learning method— works by being able to exploit properties of the input distribution (e.g., convolutional structure exploits the fact that all subpatches of images can be processed very similarly). Thus one should try to prove some measure of complicatedness that depends on the data distribution.

5.3.1 Rademacher Complexity

Rademacher complexity is a complexity measure that depends on data distribution. As usual our description assumes loss function takes values in $[0,1]$.

The definition concerns the following thought experiment. Recall that the distribution $\mathcal{D}$ is on labeled datapoints (x,y). For simplicity we denote the labeled datapoint as z.

Now *Rademacher Complexity* [6] of hypothesis class $\mathcal{H}$ on a distribution $\mathcal{D}$ is defined as follows where $l(z,h)$ is loss of hypothesis h on labeled datapoint z.

$$\mathcal{R}_{m,D}(\mathcal{H}) = \mathop{\mathbb{E}}_{S_1,S_2}\left[\frac{1}{2m}\sup_{h\in\mathcal{H}}\left|\sum_{z\in S_1} l(z,h) - \sum_{z\in S_2} l(z,h)\right|\right], \tag{5.3}$$

where the expectation is over S_1, S_2 are two iid sample sets (i.e., multisets) of size m each from the data distribution $\mathcal{D}$. Note that this definition involves the thought experiment of picking S_1, S_2 and picking a classifier where whose training error on these is as different as possible. The following theorem relates this to generalization error of the trained hypothesis.

Theorem 5.3.1. *If h is the hypothesis trained via ERM using a training set S_2 of size m, then the probability (over S_2) is $> 1-\delta$, that*

$$\Delta_{S_2}(h) \leq 2\mathcal{R}_{m,D}(\mathcal{H}) + O((\log(1/\delta))/\sqrt{m}).$$

Proof. The generalization error $\Delta_{S_2}(h) = L_{\mathcal{D}}(h) - \widehat{L_{S_2}}(h)$, and ERM guarantees an h that maximizes this. Imagine we pick another m iid samples from distribution $\mathcal{D}$ to get another (multi)set S_1. Then with probability at least $1-\delta$ the loss on these samples closely approximates $L_{\mathcal{D}}(h)$:

$$\Delta_{S_2}(h) \leq \widehat{L_{S_1}}(h) - \widehat{L_{S_2}}(h) + O((\log(1/\delta))/\sqrt{m}).$$

[6] Standard accounts of this often confuse students, or at least falsely impress them with a complicated proof of Thm 5.3.1 that hides the simple idea below. Our definition is simplified a bit: in the standard definition, one picks a sign ± 1 (or *Rademacher* random variables) for each of the $2m$ datapionts and looks at loss weighted by this sign. The value yielded by our definition is within $\pm O(1/\sqrt{m})$ of the one in the standard definition.

Now we notice that S_1, S_2 thus drawn are exactly like the sets drawn in the thought experiment of (5.3) [7] (5.3) and the maximizer h for this expression defined $\mathcal{R}_{m,D}$. So the right hand side is at most

$$2\mathcal{R}_{m,D}(\mathcal{H}) + O((\log(1/\delta))/\sqrt{m}).$$

□

Problem 5.3.2. *Show that the Rademacher complexity of the set of linear classifiers (unit norm vectors $U = \{w | w \in \mathbb{R}^d, \|w\|_2 = 1\}$), on a given sample $S = \{x_1, x_2, \cdots, x_m\}$ (each $x_i \in \mathbb{R}^d$) is $\leq \max_{i \in [m]} \|x_i\|_2 / \sqrt{m}$.*

Problem 5.3.3. *Consider the kernel classifier of the form $h(x) = z^\top G^{-1} y$ we studied in Section 3.2 where G is the $n \times n$ kernel matrix, y is the labels and z is the column vector whose i-th coordinate is $K(x, x_i)$. Show that the Rademacher complexity upper is $\sqrt{2y^\top G y \cdot \mathrm{Tr}(G)}/n$. (We will use this result in Chapter 9 to prove certain over-parameterized student nets can learn simple two-layer teacher nets.)*

5.3.2 Alternative Interpretation: Ability to correlate with random labels

Teachers explain Rademacher complexity more intuitively as *ability of classifiers in $\mathcal{H}$ to correlate with random labelings of the data.* This is best understood for binary classification (i.e., labels are 0/1), and the loss function is also binary (loss 0 for correct label and 1 incorrect label). Now consider the following experiment: Pick S_1, S_2 as in the definition of Rademacher Complexity, and imagine flipping the labels of S_1. Now average loss on S_2 is $1 - \widehat{L_{S_2}}(h)$. Thus selecting h to maximise the right hand side of (5.3) is like finding an h that has low loss on $S_1 \cup S_2$ where the labels have been flipped on S_1. In other words, h is able to achieve low loss on datasets where labels were flipped for some randomly chosen set of half of the training points.

When the loss is not binary a similar statement still holds qualitatively.

5.4 Understanding limitations of the union-bound approach

The phenomenon captured in the union bound approach and related approaches is also refered to as *uniform convergence*. If we have identified a finite set $\mathcal{H}$ of hypotheses and sample S of datapoints is large enough then with probability is at least $1 - \delta$ over choice of S that

$$\|L_{\mathcal{D}}(h) - \widehat{L}_S(h)\| \leq \epsilon \quad \forall h \in \mathcal{H}. \tag{5.4}$$

Here the important point to note is that a fixed sample set S can be used for good estimate of generalization error for *every* classifier h

[7] Here hypothesis h is allowed to depend on S_2 but not S_1. In the thought experiment the supremum is over h that can depend on both. This only helps the inequality, since the latter h can achieve a larger value. Note that the factor 2 is because of scaling of $2m$ in (5.3).

in the class[8]. Of course, using γ-cover this kind of conclusion can be shown also for classes $\mathcal{H}$ that are a continuous set, e.g. hypotheses with bounded ℓ_2 norm. Now we describe a nice and simple example from Nagarajan and Kolter [9] that pinpoints why this framework may be tricky to apply in modern settings, especially deep learning. The point is that the hypothesis class of interest is implicitly defined via the optimization algorithm (say, gradient descent), and this class may not allow a clean analysis via a union bound.

5.4.1 *An illustrative example that mixes optimization and generalization*

Suppose the points are in $\Re^{D+K}$ and the labels are ± 1. There is a fixed vector $u \in \Re^k$ such that labeled datapoints (x, y) come from the following distribution $\mathcal{D}$: first label y is uniformly picked in $\{\pm 1\}$ and then the first K coordinates of x —which we denote x_1 for convenience—are set to the vector $y \cdot u$. The remaining D coordinates, denoted x_2 consist of a random vector $\Re^D$, whose each coordinate is drawn independently from $\mathcal{N}(0, 1/D)$. Measure concentration implies x_2 is distributed essentially like a random unit vector in $\Re^D$.

The classification can clearly be solved using the linear classifier $x \to \text{sgn}(w^* \cdot x)$ where the first K coordinates contain $w_1^* = u/\|u\|_2^2$, and the last D coordinates contain $w_2^* = 0$.

Let's consider a simple training objective: find linear classifer $h(x)$ that maximises $y \cdot h(x)$. Roughly speaking, this ignores the magnitude of $h(x)$ and tries to align the sign of $h(x)$ and y. Using learning rate $\eta = 1$ and a sample S of m datapoints (x^i, y^i) for $i = 1, \dots, m$ gradient descent produces a classifier with $w_S = (w_1, w_2)$ where

$$w_{S,1} = m \cdot u, \qquad w_{S,2} = \sum_i y^i x_2^i. \tag{5.5}$$

Notice that $w_{S,2}$ is a sum of m random unit vectors, which means its norm is fairly tightly concentrated around $\sqrt{m}$. In other words, unlike our ideal classifier w^* the learnt classifier has a lot of junk in the last D coordinates that is not relevant to the classification.

Now we describe how to set the various parameters. As usual m denotes training set size. We set

$$m\sqrt{(\log 1/\delta)} \approx D \tag{5.6}$$

$$\|u\|_2^2 = \frac{1}{m} \tag{5.7}$$

The ℓ_2 norm of the learnt classifier w_S is around $\sqrt{m^2\|u\|_2^2 + m}$, and thus (5.7) implies this norm is $\sqrt{2m}$.

[8] Note that most of these classifiers may have terrible loss on S; the union bound only guarantees that the generalization error is small.

[9] V Nagarajan and Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. *NeurIPS*, 2019

Let's check that the junk coordinates do not interfere with classification for randomly chosen data points m—in other words, has good test error. Given a new data point $x = (y \cdot u, x_2)$ where x_2 is a random unit vector, the learnt classifier produces the answer $w_S \cdot x = my\|u\|_2^2 + x_2 \cdot (\sum_i y^i x_2^i)$. Since inner product between a fixed vector and a random gaussian vector $\mathcal{N}(0, 1/D)$ is a univariate gaussian with standard deviation $1/\sqrt{D}$ times the norm of the fixed vector, we see that the sign of this is correct i.e. y, with probability $1 - \delta$ so long as

$$m\|u\|_2^2 > \sqrt{\frac{m \log 1/\delta}{D}},$$

which holds from (5.6) and (5.7). Thus the learnt classifier works fine on random test data points.

But now imagine we try to explain the success of learning via a union-bound argument. Let's denote by $\mathcal{H}_0$ the set of such classifiers that could result from GD on training sets of m datapoints. The argument would have to prove that with high probability, $\Delta_S(h)$ is small for all classifiers $h \in \mathcal{H}_0$. The next result shows this is not true.

Claim 5.4.1. *For a random sample set S, whp there is a classifier w_{flip} whose generalization error is large (specifically, whose loss on full distribution $\mathcal{D}$ is small but whose loss on S is large.)*

Proof. We let w_{flip} be the classifier trained on the set S_{flip}, which is obtained by taking S and flipping the sign of the x_2 part. In other words, datapoint $z = (y^i u, x_2^i), y^i)$ of S turns into $z_{\text{flip}} = (y^i u, -x_2^i), y^i)$ in S_{flip}. Note that S_{flip} has exactly the same probability as S. By our earlier analysis, w_S and w_{flip} agree on the first K coordinates, but have the signs of the last D coordinates flipped. Thus the absolute value of $(w_S - w_{\text{flip}}) \cdot z$ is at least $2x_2^i \cdot x_2^i = 2$. Thus we have shown that the signs of $w_S \cdot z$ and $w_{\text{flip}} \cdot z$ are different. □

Let us consider what we have shown. The classifier w_S and $w_{\textbf{flip}}$ both have excellent test error. However, on the training set S used to produce w_S, the classifier $w_{\textbf{flip}}$ has bad generalization error. This shows a stumbling block on proving good generalization of w_S on training dataset S using the naive union bound.

Note that the limitations shown above do not hold if we are allowed to modify/prune the classifier obtained at the end of training. One can imagine identifying non-influential coordinates in the learnt classifier via some simple test and realizing that the last D coordinates can be zero-ed out without greatly affecting accuracy. Then all learnt classifiers become scalar multiples of the ideal classifier w^*. In other words, the limitations shown here do not apply to the approach we describe in the next Section.

5.5 A Compression-based framework

Now we described a simple compression-based technique[10] from Arora et al [11] that formalizes a very simple idea. Suppose the training dataset S contains m samples, and h is a classifier from a complicated class (e.g., deep nets with much more than m parameters) that incurs very low empirical loss. We are trying to understand from looking at h and S how well h will generalize. Now suppose we can compute a classifier g with discrete trainable parameters much less than m and which incurs similar loss on the training data as h. We call this an *approximator* for h. Then if g has sufficiently low description length, it's generalization follows by simple union bound argument. [12]

This framework has the advantage of staying with intuitive parameter counting and to avoid explicitly dealing with the hypothesis class that includes h (see note after Theorem 5.5.3). Notice, the mapping from f to g merely needs to *exist*, not to be efficiently computable. But in all our examples the mapping will be explicit and fairly efficient. Now we formalize the notions. The proofs are elementary via concentration bounds and appear in the appendix.

Definition 5.5.1 ((γ,S)-compressible)**.** *Let f be a classifier and $G_{\mathcal{A}} = \{g_A | A \in \mathcal{A}\}$ be a class of classifiers. We say f is (γ, S)-compressible via $G_{\mathcal{A}}$ if there exists $A \in \mathcal{A}$ such that for any $x \in S$, we have for all y*

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

We also consider a different setting where the compression algorithm is allowed a"helper string" s, which is arbitrary but fixed before looking at the training samples. Often s will contain random numbers. [13]

Definition 5.5.2 ((γ,S)-compressible using helper string s)**.** *Suppose $G_{\mathcal{A},s} = \{g_{A,s} | A \in \mathcal{A}\}$ is a class of classifiers indexed by trainable parameters A and fixed strings s. A classifier f is (γ, S)-compressible with respect to $G_{\mathcal{A},s}$ using helper string s if there exists $A \in \mathcal{A}$ such that for any $x \in S$, we have for all y*

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

The following theorem is a simple application of the union bound method above.

Theorem 5.5.3. *Suppose $G_{\mathcal{A},s} = \{g_{A,s} | A \in \mathcal{A}\}$ where A is a set of q parameters each of which can have at most r discrete values and s is a helper string. Let S be a training set with m samples. If the trained classifier f is (γ, S)-compressible via $G_{\mathcal{A},s}$ with helper string s, then there exists $A \in \mathcal{A}$*

[10] Do not confuse this with another older and unrelated technique in generalization theory based upon *data compression*, which is not applicable to deep learning.

[11] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *Proc. ICML 2018*, pages 254–263, 2018

[12] This scenario is quite reminiscent of empirical work in network pruning, whereby trained deep nets are compressed using one of a long list of methods that prune away lots of parameters and retrain the rest. If network left after pruning is compact enough, one can conceivably prove generalization bounds for the pruned net. See

[13] A simple example is to let s be the random initialization used for training the deep net. Then one could compress the *difference* between the final weights and s; this can give better generalization bounds.

with high probability over the training set,

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

where $L(f) = \mathbb{E}_{(x,y)\in\mathcal{D}}[f(x)[y] \leq \max_{j\neq y} f(x)[j]]$ *is the expected error and* $\widehat{L}_\gamma(f)$ *is the proportion of data* (x,y) *satisfying* $f(x)[y] \leq \max_{j\neq y} f(x)[j]$ *in the training set S.*

Remarks: (1) The framework proves the generalization not of f but of its compression g_A. (An exception is if the two are shown to have similar loss at every point in the domain, not just the training set. This is the case in Theorem 5.5.6.)
(2) The previous item highlights the difference from what we called the union bound earlier (Theorem 5.2.1). There, one needs to fix a hypothesis class *independent* of the training set. By contrast we have no hypothesis class, only a *single* neural net that has some specific properties on a *single* finite training set. But if we can compress this specific neural net to a simpler neural nets with fewer parameters then we can use covering number argument on this simpler class to get the generalization of the compressed net.
(3) Issue (1) exists also in how researchers often apply the standard PAC-Bayes framework for deep nets (Section 5.6).

5.5.1 *Example 1: Linear classifiers with margin*

To illustrate the above compression method we use linear classifiers with high margins. Consider a simple family of linear classifiers, consisting of unit vectors $c \in \mathbb{R}^d$ whose ± 1 output on input x is given by $\mathrm{sgn}(c \cdot x)$ (i.e., sign of the inner product with the datapoint). Assume that all data points are also unit vectors. Say c has *margin* γ if for all training pairs (x, y) we have $y(c^\top x) \geq \gamma$.

We show how to compress such a classifier with margin γ to one that has only $O(1/\gamma^2)$ non-zero entries. First, assume all c_i have absolute value less than $\gamma^2/8$.

For each coordinate i, toss a coin with $\Pr[heads] = p_i = 8c_i^2/\gamma^2$ and if it comes up heads set the coordinate to equal to $c_i/p_i = \gamma^2/8c_i$. This yields a vector $\widehat{c}$. The expected number of non-zero entries in $\widehat{c}$ is $\sum_{i=1}^d p_i = 8/\gamma^2$. By Chernoff bound we know with high probability the number of non-zero entries is at most $O(1/\gamma^2)$.

Furthermore, variance of coordinate i of $\widehat{c}$ is $2p_i(1-p_i)\frac{c_i^2}{p_i^2} \leq \frac{2c_i^2}{p_i} \leq \gamma^2/4$. Therefore, for any unit vector u that is independent with the choice of $\widehat{c}$, we have $\mathbb{E}[\widehat{c}^\top u] = c^\top u$. Now we estimate variance of the random variable $\widehat{c}^\top u$. It is $\leq \gamma^2/4 \cdot \|u\|^2 \leq \gamma^2/4$. By Chebyshev's inequality we know $\Pr[|\widehat{c}^\top u - c^\top u| \geq \gamma] \leq 1/4$, so $\widehat{c}$ and c will make the same prediction for all u satisfying $|c^\top u| \geq \gamma$. We can then apply Theorem 5.5.3 on a discretized version of $\widehat{c}$ (via trivial rounding) to show that the sparsified classifier has good generalization with $O(\log d/\gamma^2)$ samples.

Problem 5.5.4. *Redo the proof above when some coordinates have absolute value more than $\gamma^2/8$.*

This compressed classifier works correctly for a fixed input x with constant probability but not high probability. To fix this, one can recourse to the "compression with fixed string" model. The fixed string is a random linear transformation. When applied to unit vector c, it tends to equalize all coordinates and the guarantee $|\widehat{c}^\top u - c^\top u| < \gamma$ can hold with high probability. This random linear transformation can be fixed before seeing the training data.

Problem 5.5.5. *Prove the above property of random linear transformations. That is, let M be a random matrix of size $O(1/\gamma^2) \times d$, drawn from a suitable distribution you choose before seeing the unit vector c and the training data. Then, show that the following holds for fixed unit vectors c and u with high probability*

$$\|Mc\|_\infty = O(1), \qquad |\langle Mc, Mu\rangle - \langle c, u\rangle| < \gamma.$$

This means we can compress a unit vector c to $\widehat{c} = M^\top Mc$. Finally, Apply Theorem 5.5.3 on a discretized version of $\widehat{c}$ to show a good generalization bound with $\widetilde{O}(1/\gamma^2)$ samples, where $\widetilde{O}$ can hide polylog factors of d and $1/\gamma$.

5.5.2 *Example 2: Generalization bounds for deep nets using low rank approximations*

Some of the early generalization bounds for fully connected nets used the fact that layer matrices are often found to be low rank. (Or perhaps the final matrix minus the initialization.) We give a simple proof of such a result.

Realize that an $h \times h$ matrix of rank r has effectively $2hr$ parameters despite having h^2 entries. We recall that for a square matrix A the spectral norm (i.e., largest singular value) is denoted $\|A\|_2$ and sum of squares of singular values is denoted denoted $\|A\|_F^2$ where $\|\cdot\|_F$ is also called *Frobenius norm*. The ratio $\|A\|_F^2/\|A\|_2^2$ is called *stable rank*, and it is clearly upper bounded by the rank. Often the layers of the trained net have low stable rank even though rank per se is high.

Theorem 5.5.6. ([14]) *For a depth-d ReLU net with hidden layers of equal width h and single coordinate output, let $A^1, A^2, \ldots A^d$ be weight matrices*

[14] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *ICLR*, 2018

and γ be the output margin on a training set S of size m. Then the generalization error can be bounded by

$$\widetilde{O}\left(\sqrt{\frac{hd^2 \max_{x\in S} \|x\| \prod_{i=1}^{d} \|A^i\|_2^2 \sum_{i=1}^{d} \frac{\|A^i\|_F^2}{\|A^i\|_2^2}}{\gamma^2 m}}\right).$$

The second part of this expression ($\sum_{i=1}^{d} \frac{\|A^i\|_F^2}{\|A^i\|_2^2}$) is sum of stable ranks of the layers, a natural measure of their true parameter count. The first part ($\prod_{i=1}^{d} \|A^i\|_2^2$) is related to the Lipschitz constant of the network, namely, the maximum norm of the vector it can produce if the input is a unit vector. The Lipschitz constant of a matrix operator B is just its spectral norm $\|B\|_2$. Since the network applies a sequence of matrix operations interspersed with ReLU, and ReLU is 1-Lipschitz we conclude that the Lipschitz constant of the full network is at most $\prod_{i=1}^{d} \|A^i\|_2$.

To prove Theorem 5.5.6 we use the following lemma to compress the matrix at each layer to a matrix of smaller rank. Since a matrix of rank r can be expressed as the product of two matrices of inner dimension r, it has $2hr$ parameters (instead of the trivial h^2). (Furthermore, the parameters can be discretized via trivial rounding to get a compression with discrete parameters as needed by Definition 5.5.1.)

Lemma 5.5.7. *For any matrix $A \in \mathbb{R}^{m\times n}$, let $\widehat{A}$ be the truncated version of A where singular values that are smaller than $\delta\|A\|_2$ are removed. Then $\|\widehat{A} - A\|_2 \leq \delta\|A\|_2$ and $\widehat{A}$ has rank at most $\|A\|_F^2/(\delta^2\|A\|_2^2)$.*

Proof. Let r be the rank of $\widehat{A}$. By construction, the maximum singular value of $\widehat{A} - A$ is at most $\delta\|A\|_2$. Since the remaining singular values are at least $\delta\|A\|_2$, we have $\|A\|_F \geq \|\widehat{A}\|_F \geq \sqrt{r}\delta\|A\|_2$. □

For each i replace layer i by its compression using the above lemma, with $\delta = \gamma(3d\|x\| \prod_{i=1}^{d} \|A^i\|_2)^{-1}$. How much error does this introduce at each layer and how much does it affect the output after passing through the intermediate layers (and getting magnified by their Lipschitz constants)? Since $A - \widehat{A}^i$ has spectral norm (i.e., Lipschitz constant) at most $\delta\|A^i\|_2$, the error at the output due to changing layer i in isolation is at most $\prod_{j=i+1}^{d} \|A^j\|_2 \cdot \delta\|A^i\|_2 \cdot \prod_{j=1}^{i-1} \|A^j\|_2 \cdot \|x\| \leq \gamma/3d$. Rest of the proof is left to the reader and generalization bound follows immediately from Theorem 5.5.3.

Problem 5.5.8. *Complete the above proof using a simple induction (see [15] if needed) to show the total error incurred in all layers is strictly bounded by γ. That is, for an input x, the change in the deep net output is smaller than γ after replacing every weight matrix A^i with its truncated version $\widehat{A}^i$.*

[15] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *ICLR*, 2018

5.6 PAC-Bayes bounds

These bounds due to McAllester (1999) [McA99] are in principle the tightest, meaning previous bounds in this chapter are its subcases. They are descended from an old philosophical tradition of considering the logical foundations for belief systems, which often uses Bayes' Theorem. For example, in the 18th century, Laplace sought to give meaning to questions like "*What is the probability that the sun will rise tomorrow?*" The answer to this question depends upon the person's prior beliefs (e.g., degree of scientific knowledge) as well as their empirical observation that the sun has risen every day in their lifetime. This philosophical connection sometimes helps students improve their understanding of generalization.

In ML context, PAC-Bayes bounds assume that experimenter (i.e., ML expert) has some prior distribution P over the hypothesis $\mathcal{H}$. If asked to classify without seeing any concrete training data, the experimenter would pick a hypothesis h according to P (denoted $h \sim P$) and classify using it h. After seeing the training data and running computations, the experimenter's distribution changes to the posterior Q, meaning now if asked to classify they would pick $h \sim Q$ and use that. Thus the expected test loss is

$$\mathop{\mathbb{E}}_{h \sim Q}[L_{\mathcal{D}}(h)].$$

The theory requires Q to be a *valid posterior* with respect to P, meaning every hypothesis h that gets zero probability under P also must have zero probability under Q. The following form of PAC-Bayes bound is from [16].

Theorem 5.6.1 (PAC-Bayes bound)**.** *Let $\mathcal{D}$ be the data distribution and P be a prior distribution over hypothesis class $\mathcal{H}$ and $\delta > 0$. If S is a set of i.i.d. samples of size m from $\mathcal{D}$ and Q is any valid posterior (possibly depending arbitrarily on S) then $\Delta_S(Q) = \mathbb{E}_{h\sim Q}[L_{\mathcal{D}}(h) - \widehat{L}_S(h)]$ satisfies the following bound with probability $1-\delta$,*

$$\Delta_S(Q) \leq \sqrt{\frac{D(Q||P) + \ln(2m/\delta)}{2(m-1)}},$$

where $D(Q||P) = \mathbb{E}_{h\sim Q}[\ln(Q(h)/P(h))]$ is the so-called KL-divergence[17]*.*

In other words, generalization error can be upper bounded using the (square root of) KL-divergence of the distributions, plus some terms that arise from concentration bounds.

Example 5.6.2. *P could be the standard normal distribution, which assigns nonzero probability to every vector. For any sample set S, we could let Q be*

[16] John Langford. *Quantitatively tight sample complexity bounds*. PhD Thesis CMU, 2002

[17] This is a measure of distance between distributions, meaningful when P dominates Q, in the sense that every h with nonzero probability in Q also has nonzero probability in P. Note that in this definition, $0 \ln 0$ is interpreted as 0.

the distribution on parameter vectors obtained by vanilla deep learning using S: that is, initialize parameters using random Gaaussian, and train with SGD with a predetermined learning rate schedule. Since SGD is a stochastic process (due to randomness of batches) it leads to a natural distribution Q on trained classifiers at the end of training. Notice, Q is a valid posterior of P because P assigns nonzero probability to every classifier h. As this example emphasizes, one can consider various P and Q for the same classification setup (e.g., by changing some aspect of training) and the generalization bound will hold for every fixed choice.

Example 5.6.3. *Suppose h is any classifier and P, Q are the distribution that assigns probability 1 to h and zero to all other hypotheses. Then $D(Q||P) = 0$, and by Hoeffding bound we have $\Delta_S(Q) = \Delta_S(h) \leq \sqrt{\frac{\log(1/\delta)}{2m}}$. The inequality in PAC-Bayes bound is satisfied.*

Problem 5.6.4. *Derive the union bound Theorem 5.2.1 using PAC-Bayes.*

Now we're ready to prove Theorem 5.6.1. In interest of exposition, we prove a weaker statement that is qualitatively similar but not quite correct:

$$\Delta_S(Q) \leq \sqrt{\frac{2(D(Q||P) + \ln(2/\delta))}{m}} \tag{5.8}$$

The incorrectness arises due to a simplifying assumption about the quantity $z = \sqrt{m}\Delta_S(h)$ where h is a fixed classifier and S is a random subset of m samples. Since $\Delta_S(h)$ is an average of m iid variables taking values in $[-1, 1]$ and with mean 0, we assume z behaves *exactly* like a normal distribution $\mathcal{N}(0, 1)$. Of course, in truth z is dominated in distribution by $\mathcal{N}(0, 1)$ in the limit $m \to \infty$. This assumption can be removed by using a more quantitative argument with Hoeffding bound. The assumption allows us to assume that expected value of $e^{z^2/(2+\epsilon)}$ approaches $\sqrt{2}$ when x is drawn from $\mathcal{N}(0, 1)$ and ϵ is an arbitrarily small constant. For simplicity we will assume $e^{z^2/2} = \sqrt{2}$. It is possible to fix the proof using concentration bounds.

Proof. (Theorem 5.6.1, weaker version (5.8)) Rearranging the expression in the theorem statement, we see that it gives an upper bound of $\ln(2/\delta)$ on $(m/2)\,\mathbb{E}_{h\sim Q}[\Delta_S(h)]^2 - D(Q||P)$. By Jensen's inequality [18] applied to the square function $f(x) = x^2$, this expression is at most $(m/2)\,\mathbb{E}_{h\sim Q}[\Delta_S(h)^2] - D(Q||P)$. We show this is upper bounded by $\ln(2/\delta)$. The steps are:

$$\begin{aligned}
&= \mathop{\mathbb{E}}_{h\sim Q}\left[(m/2)\Delta_S(h)^2 - \ln(Q(h)/P(h))\right] \\
&= \mathop{\mathbb{E}}_{h\sim Q}\left[\ln\left(\exp((m/2)\Delta_S(h)^2)\cdot P(h)/Q(h)\right)\right] \\
&\leq \ln\left(\mathop{\mathbb{E}}_{h\sim Q}\left[\exp((m/2)\Delta_S(h)^2)\cdot P(h)/Q(h)\right]\right),
\end{aligned}$$

[18] Jensen's Inequality: For a concave function f and random variable X, $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$. For convex function the inequality is reversed.

where the last inequality uses Jensen's inequality along with the concavity of ln. Also, since taking expectation over $h \sim Q$ is effectively like a weighted sum with term for h weighted by $Q(h)$, we have [19]

$$\ln \mathop{\mathbb{E}}_{h\sim Q}\left[\exp((m/2)\Delta(h)^2)\cdot P(h)/Q(h)\right] = \ln \mathop{\mathbb{E}}_{h\sim P}\left[\exp((m/2)\Delta(h)^2)\right].$$

Recapping, we thus thus shown the following for a fixed dataset S:

$$(m/2)\mathop{\mathbb{E}}_{h\sim Q}[\Delta_S(h)]^2 - D(Q||P) \leq \ln\left(\mathop{\mathbb{E}}_{h\sim P}\left[e^{(m/2)\Delta_S(h)^2}\right]\right) \tag{5.9}$$

Note that the RHS has no dependence on posterior Q. Using the fact that S is a random sample of size m and that prior belief P was fixed before seeing S (i.e., is independent of S):

$$\mathop{\mathbb{E}}_{S}\left[\mathop{\mathbb{E}}_{h\sim P}\left[e^{(m/2)\Delta_S(h)^2}\right]\right] = \mathop{\mathbb{E}}_{h\sim P}\left[\mathop{\mathbb{E}}_{S}\left[e^{(m/2)\Delta_S(h)^2}\right]\right] = \sqrt{2} \leq 2.$$

Simple averaging implies that with probability $1-\delta$ over S,

$$\mathop{\mathbb{E}}_{h\sim P}\left[e^{(m/2)\Delta_S(h)^2}\right] \leq 2/\delta \tag{5.10}$$

and now by taking logarithm of both sides the proof is completed.

□

5.7 *Exercises*

1. Assume the loss function ℓ is 1-Lipschitz. Consider the kernel classifier of the form $h(x) = z^\top G^{-1}y$ we studied in Section 3.2 where G is the $n \times n$ kernel matrix, y is the labels and z is the column vector whose i-th coordinate is $K(x, x_i)$. Prove that its Rademacher complexity is upper bounded $\sqrt{2y^\top Gy \cdot \mathrm{Tr}(G)/n}$. (Hint: view kernel classifier as a linear classifier in RKHS)

[19] Often when you see KL-divergence in machine learning, you will see this trick being used to switch the distribution over which expectation is taken!

6
Tractable Landscapes for Nonconvex Optimization

Deep learning relies on optimizing complicated, nonconvex loss functions. Finding the global minimum of a nonconvex objective is NP-hard in the worst case. However in deep learning simple algorithms such as stochastic gradient descent often the objective value to zero or near-zero at the end. This chapter focuses on the *optimization landscape* defined by a nonconvex objective and identifies properties of these landscapes that allow simple optimization algorithms to find global minima (or near-minima). These properties thus far apply to simpler nnonconvex problems than deep learning, and it is open how to analyse deep learning with such landscape analysis.

Warm-up: Convex Optimization To understand optimization landscape, one can first look at optimizing a convex function. If a function $f(w)$ is convex, then it satisfies many nice properties, including

$$\forall \alpha \in [0,1], w, w', f(\alpha w + (1-\alpha)w') \leq \alpha f(w) + (1-\alpha) f(w'). \tag{6.1}$$

$$\forall w, w', f(w') \geq f(w) + \langle \nabla f(w), w' - w \rangle. \tag{6.2}$$

These equations characterize important geometric properties of the objective function $f(w)$. In particular, Equation (6.1) shows that all the global minima of $f(w)$ must be connected, because if w, w' are both globally optimal, anything on the segment $\alpha w + (1-\alpha)w'$ must also be optimal. Such properties are important because it gives a characterization of all the global minima. Equation (6.2) shows that every point with $\nabla f(w) = 0$ must be a global minimum, because for every w' we have $f(w') \geq f(w) + \langle \nabla f(w), w' - w) \rangle \geq f(w)$. Such properties are important because it connects a local property (gradient being 0) to global optimality.

In general, optimization landscape looks for properties of the objective function that characterizes its local/global optimal points (such as Equation (6.1)) or connects local properties with global optimality (such as Equation (6.2)).

6.1 Preliminaries and challenges in nonconvex landscapes

We have been discussing global/local minimum informally, here we first give a precise definition:

Definition 6.1.1 (Global/Local minimum)**.** *For an objective function $f(w) : \mathbb{R}^d \to \mathbb{R}$, a point w^* is a* global minimum *if for every w we have $f(w^*) \leq f(w)$. A point w is a* local minimum/maximum *if there exists a radius $\epsilon > 0$ such that for every $\|w' - w\|_2 \leq \epsilon$, we have $f(w) \leq f(w')$ ($f(w) \geq f(w')$ for local maximum). A point w with $\nabla f(w) = 0$ is called a* critical point, *for smooth functions all local minimum/maximum are critical points.*

Throughout the chapter, we will always work with functions whose global minimum exists, and use $f(w^*)$ to denote the optimal value of the function[1]. For simplicity we focus on optimization problems that do not have any constraints ($w \in \mathbb{R}^d$). It is possible to extend everything in this chapter to optimization with nondegenerate equality constraints, which would require definitions of gradient and Hessians with respect to a manifold and is out of the scope for this book.

Spurious local minimum The first obstacle in nonconvex optimization is a *spurious local minimum*.

Definition 6.1.2 (Spurious local minimum)**.** *For an objective function $f(w) : \mathbb{R}^d \to \mathbb{R}$, a point w is a spurious local minimum if it is a local minimum, but $f(w) > f(w^*)$.*

Many of the simple optimization algorithms are based on the idea of local search, thus are not able to escape from a spurious local minimum. As we will later see, many noncovex objectives do not have spurious local minima.

Saddle points The second obstacle in nonconvex optimization is a *saddle point*. The simplest example of a saddle point is $f(w) = w_1^2 - w_2^2$ at the point $w = (0, 0)$. In this case, if w moves along direction $(\pm 1, 0)$, the function value increases; if w moves along direction $(0, \pm 1)$, the function value decreases.

Definition 6.1.3 (Saddle point)**.** *For an objective function $f(w) : \mathbb{R}^d \to \mathbb{R}$, a point w is a saddle point if $\nabla f(w) = 0$, and for every radius $\epsilon > 0$, there exists w^+, w^- within distance ϵ of w such that $f(w^-) < f(w) < f(w^+)$.*

This definition covers all cases but makes it very hard to verify whether a point is a saddle point. In most cases, it is possible to tell

[1] Even though there might be multiple global minima w^*, the value $f(w^*)$ is unique by definition.

whether a point is a saddle point, local minimum or local maximum based on its Hessian.

Claim 6.1.4. *For an objective function $f(w) : \mathbb{R}^d \to \mathbb{R}$ and a critical point w ($\nabla f(w) = 0$), we know*

- *If $\nabla^2 f(w) \succ 0$, w is a local minimum.*
- *If $\nabla^2 f(w) \prec 0$, w is a local maximum.*
- *If $\nabla^2 f(w)$ has both a positive and a negative eigenvalue, w is a saddle point.*

These criteria are known as second order sufficient conditions in optimization. Intuitively, one can prove this claim by looking at the second-order Taylor expansion. The three cases in the claim do not cover all the possible Hessian matrices. The remaining cases are considered to be degenerate, and can either be a local minimum, local maximum or a saddle point[2].

Flat regions Even if a function does not have any spurious local minima or saddle point, it can still be nonconvex, see Figure 6.1. In high dimensions such functions can still be very hard to optimize. The main difficulty here is that even if the norm $\|\nabla f(w)\|_2$ is small, unlike convex functions one cannot conclude that $f(w)$ is close to $f(w^*)$. However, often in such cases one can hope the function $f(w)$ to satisfy some relaxed notion of convexity, and design efficient algorithms accordingly. We discuss one of such cases in Section 6.2.

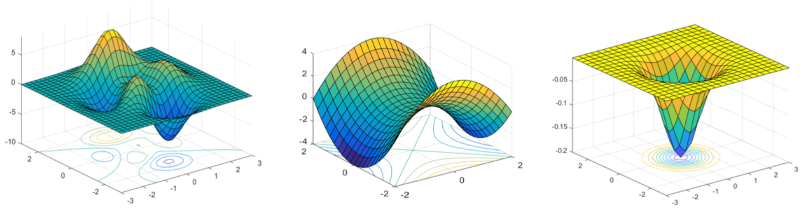


Figure 6.1: Obstacles for nonconvex optimization. From left to right: local minimum, saddle point and flat region.

6.2 *Cases with a unique global minimum*

We first consider the case that is most similar to convex objectives. In this section, the objective functions we look at have no spurious local minima or saddle points. In fact, in our example the objective is only going to have a unique global minimum. The only obstacle in optimizing these functions is that points with small gradients may not be near-optimal.

[2] One can consider the $w = 0$ point of functions $w^4, -w^4, w^3$, and it is a local minimum, maximum and saddle point respectively.

The main idea here is to identify properties of the objective and also a *potential function*, such that the potential function keeps decreasing as we run simple optimization algorithms such as gradient descent. Many properties were used in previous literature, including

Definition 6.2.1. *Let $f(w)$ be an objective function with a unique global minimum w^*, then*

Polyak-Lojasiewicz *f satisfies Polyak-Lojasiewicz if there exists a value $\mu > 0$ such that for every w, $\|\nabla f(x)\|_2^2 \geq \mu(f(w) - f(w^*))$.*

weakly-quasi-convex *f is weakly-quasi-convex if there exists a value $\tau > 0$ such that for every w, $\langle \nabla f(w), w - w^* \rangle \geq \mu(f(w) - f(w^*))$.*

Restricted Secant Inequality (RSI) *f satisfies RSI if there exists a value τ such that for every w, $\langle \nabla f(w), w - w^* \rangle \geq \mu \|w - w^*\|_2^2$.*

Any one of these three properties can imply fast convergence together with some smoothness of f.

Claim 6.2.2. *If an objective function f satisfies one of Polyak-Lojasiewicz, weakly-quasi-convex or RSI, and f is smooth*[3]*, then gradient descent converges to global minimum with a geometric rate*[4]*.*

Intuitively, Polyak-Lojasiewicz condition requires that the gradient to be nonzero for any point that is not a global minimum, therefore one can always follow the gradient and further decrease the function value. This condition can also work in some settings when the global minimum is not unique. Weakly-quasi-convex and RSI are similar in the sense that they both require the (negative) gradient to be correlated with the correct direction - direction from the current point w to the global minimum w^*.

In this section we are going to use generalized linear model as an example to show how some of these properties can be used.

6.2.1 *Generalized linear model*

In generalized linear model (also known as isotonic regression) [KS09, KKSK11], the input consists of samples $\{x^{(i)}, y^{(i)}\}$ that are drawn from distribution $\mathcal{D}$, where $(x, y) \sim \mathcal{D}$ satisfies

$$y = \sigma(w_*^\top x) + \epsilon. \tag{6.3}$$

Here $\sigma : \mathbb{R} \to \mathbb{R}$ is a known monotone function, ϵ is a noise that satisfies $\mathbb{E}\left[\epsilon | x\right] = 0$, and w_* is the unknown parameter that we are trying to learn.

In this case, it is natural to consider the following expected loss

$$L(w) = \frac{1}{2} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[(y - \sigma(w^\top x)^2 \right]. \tag{6.4}$$

[3] Polyak-Lojasiewicz and RSI requires standard smoothness definition as in Equation (2.2), weakly-quasi-convex requires a special smoothness property detailed in [HMR18].

[4] The potential functions for Polyak-Lojasiewicz and weakly-quasi-convex are function value f; potential function for RSI is the squared distance $\|w - w_*\|_2^2$

Of course, in practice one can only access the training loss which is an average over the observed $\{x^{(i)}, y^{(i)}\}$ pairs. For simplicity we work with the expected loss here. The difference between the two losses can be bounded using techniques in Chapter **??**.

Generalized linear model can be viewed as learning a single neuron where σ is its nonlinearity.

We will give high level ideas on how to prove properties such as weakly-quasi-convex or RSI for generalized linear model. First we rewrite the objective as:

$$\begin{aligned} L(w) &= \frac{1}{2} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[(y - \sigma(w^\top x)^2 \right] \\ &= \frac{1}{2} \mathop{\mathbb{E}}_{(x,\epsilon)} \left[(\epsilon + \sigma(w_*^\top x) - \sigma(w^\top x))^2 \right]. \\ &= \frac{1}{2} \mathop{\mathbb{E}}_{\epsilon} \left[\epsilon^2 \right] + \frac{1}{2} \mathop{\mathbb{E}}_{x} \left[(\sigma(w_*^\top x) - \sigma(w^\top x)^2 \right]. \end{aligned}$$

Here the second equality uses Definition of the model (Equation (6.3)), and the third equality uses the fact that $\mathbb{E}\left[\epsilon|x\right] = 0$ (so there are no cross terms). This decomposition is helpful as the first term $\frac{1}{2}\mathbb{E}_\epsilon\left[\epsilon^2\right]$ is now just a constant.

Now we can take the derivative of the objective:

$$\nabla L(w) = \mathop{\mathbb{E}}_{x} \left[(\sigma(w^\top x) - \sigma(w_*^\top x))\sigma'(w^\top x)x \right].$$

Notice that both weakly-quasi-convex and RSI requires that the objective to be correlated with $w - w_*$, so we compute

$$\langle \nabla L(w), w - w^* \rangle = \mathop{\mathbb{E}}_{x} \left[(\sigma(w^\top x) - \sigma(w_*^\top x))\sigma'(w^\top x)(w^\top x - w_*^\top x) \right].$$

The goal here is to show that the RHS is bigger than 0. A simple way to see that is to use the intermediate value theorem: $\sigma(w^\top x) - \sigma(w_*^\top x) = \sigma'(\xi)(w^\top x - w_*^\top x)$, where ξ is a value between $w^\top x$ and $w_*^\top x$. Then we have

$$\langle \nabla L(w), w - w^* \rangle = \mathop{\mathbb{E}}_{x} \left[\sigma'(\xi)\sigma'(w^\top x)(w^\top x - w_*^\top x)^2 \right].$$

In the expectation in the RHS, both derivatives $(\sigma'(\xi), \sigma'(w^\top x))$ are positive as σ is monotone, and $(w^\top x - w_*^\top x)^2$ is clearly nonnegative. By making more assumptions on σ and the distribution of x, it is possible to lowerbound the RHS in the form required by either weakly-quasi-convex or RSI. We leave this as an exercise.

6.2.2 *Alternative objective for generalized linear model*

There is another way to find w_* for generalized linear model that is more specific to this setting. In this method, one estimate a different "gradient" for generalized linear model:

$$\nabla g(w) = \mathop{\mathbb{E}}_{x,y}\left[(\sigma(w^\top x) - y)x\right] = \mathop{\mathbb{E}}_{x}\left[(\sigma(w^\top x) - \sigma(w_*^\top x))x\right]. \tag{6.5}$$

The first equation gives a way to estimate this "gradient". The main difference here is that in the RHS we no longer have a factor $\sigma'(w^\top x)$ as in $\nabla L(w)$. Of course, it is unclear why this formula is the gradient of some function g, but we can construct the function g in the following way:

Let $\tau(x)$ be the integral of $\sigma(x)$: $\tau(x) = \int_0^x \sigma(x')\mathrm{d}x'$. Define $g(w) := \mathbb{E}_x\left[\tau(w^\top x) - \sigma(w_*^\top x)w^\top x\right]$. One can check $\nabla g(w)$ is indeed the function in Equation (6.5). What's very surprising is that $g(w)$ is actually a convex function with $\nabla g(w_*) = 0$! This means that w_* is a global minimum of g and we only need to follow $\nabla g(w)$ to find it. Nonconvex optimization is unnecessary here.

Of course, this technique is quite special and uses a lot of structure in generalized linear model. However similar ideas were also used in [5] to learn a single neuron. In general, when one objective is hard to analyze, it might be easier to look for an alternative objective that has the same global minimum but easier to optimize.

[5]

6.3 Symmetry, saddle points and locally optimizable functions

In the previous section, we saw some conditions that allow nonconvex objectives to be optimized efficiently. However, such conditions often do not apply to neural networks, or more generally any function that has some symmetry properties.

More concretely, consider a two-layer neural network $h_\theta(x) : \mathbb{R}^d \to \mathbb{R}$. The parameters θ is $(w_1, w_2, ..., w_k)$ where $w_i \in \mathbb{R}^d$ represents the weight vector of the i-th neuron. The function can be evaluated as $h_\theta(x) = \sum_{i=1}^k \sigma(\langle w_i, x\rangle)$, where σ is a nonlinear activation function. Given a dataset $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \overset{\text{i.i.d.}}{\sim} \mathcal{D}$, one can define the training loss and expected loss as in Chapter 1. Now the objective for this neural network $f(\theta) = L(h_\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\ell((x,y), h_\theta)\right]$ has *permutation symmetry*. That is, for any permutation $\pi(\theta)$ that permutes the weights of the neurons, we know $f(\theta) = f(\pi(\theta))$.

The symmetry has many implications. First, if the global minimum θ^* is a point where not all neurons have the same weight vector (which is very likely to be true), then there must be equivalent global minimum $f(\pi(\theta^*))$ for every permutation π. An objective with this symmetry must also be nonconvex, because if it were convex, the point $\bar{\theta} = \frac{1}{k!}\sum_\pi \pi(\theta^*)$ (where π sums over all the permutations) is a convex combination of global minima, so it must also be a global minimum. However, for $\bar{\theta}$ the weight vectors of the neurons are all equal to $\frac{1}{k}\sum_{i=1}^{k} w_i$ (where w_i is the weight of i-th neuron in θ^*), so $h_{\bar{\theta}}(x) = k\sigma(\langle \frac{1}{k}\sum_{i=1}^{k} w_i, x\rangle)$ is equivalent to a neural network with a single neuron. In most cases a single-neuron network should not achieve the global minimum, so by proof of contradiction we know f should not be convex.

It's also possible to show that functions with symmetry must have saddle points[6]. Therefore to optimize such a function, the algorithm needs to be able to either avoid or escape from saddle points. More concretely, one would like to find a *second order stationary point*.

Definition 6.3.1 (Second order stationary point (SOSP))**.** *For an objective function $f(w) : \mathbb{R}^d \to \mathbb{R}$, a point w is a second order stationary point if $\nabla f(w) = 0$ and $\nabla^2 f(w) \succeq 0$.*

The conditions for second order stationary point are known as the second order necessary conditions for a local minimum. Of course, generally an optimization algorithm will not be able to find an exact second order stationary point (just like in Section **??** we only show gradient descent finds a point with small gradient, but not 0 gradient). The optimization algorithms can be used to find an approximate second order stationary point:

Definition 6.3.2 (Approximate second order stationary point)**.** *For an objective function $f(w) : \mathbb{R}^d \to \mathbb{R}$, a point w is a (ϵ, γ)-second order stationary point (later abbreviated as (ϵ, γ)-SOSP) if $\|\nabla f(w)\|_2 \le \epsilon$ and $\lambda_{\min}(\nabla^2 f(w)) \ge -\gamma$.*

Later in Chapter **??** we will show that simple variants of gradient descent can in fact find (ϵ, γ)-SOSPs efficiently.

Now we are ready to define a class of functions that can be optimized efficiently and allow symmetry and saddle points.

Definition 6.3.3 (Locally optimizable functions)**.** *An objective function $f(w)$ is locally optimizable, if for every $\tau > 0$, there exists $\epsilon, \gamma = poly(\tau)$ such that every (ϵ, γ)-SOSP w of f satisfies $f(w) \le f(w_*) + \tau$.*

Roughly speaking, an objective function is locally optimizable if every local minimum of the function is also a global minimum, and the Hessian of every saddle point has a negative eigenvalue. Similar class of functions were called "strict saddle" or "ridable" in some previous results. Many nonconvex objectives, including matrix sensing [BNS16a, PKCS17, GJZ17a], matrix completion [GLM16a, GJZ17a], dictionary learning [SQW16a], phase retrieval [SQW18], tensor decomposition [GHJY15a], synchronization problems [BBV16] and certain objective for two-layer neural network [GLM18] are known to be locally optimizable.

[6] Except some degenerate cases such as constant functions.

6.4 Case study: top eigenvector of a matrix

In this section we look at a simple example of a locally optimizable function. Given a symmetric PSD matrix $M \in \mathbb{R}^{d\times d}$, our goal is to find its top eigenvector (eigenvector that corresponds to the largest eigenvalue). More precisely, using SVD we can write M as

$$M = \sum_{i=1}^{d} \lambda_i v_i v_i^\top.$$

Here v_i's are orthonormal vectors that are eigenvectors of M, and λ_i's are the eigenvalues. For simplicity we assume $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_d \geq 0$[7].

There are many objective functions whose global optima gives the top eigenvector. For example, using basic definition of spectral norm, we know for PSD matrix M the global optima of

$$\max_{\|x\|_2=1} x^\top M x$$

is the top eigenvector of M. However, this formulation requires a constraint. We instead work with an unconstrained version whose correctness follows from Eckhart-Young Theorem:

$$\min_{x\in\mathbb{R}^d} f(x) := \frac{1}{4}\|M - xx^\top\|_F^2. \tag{6.6}$$

Note that this function does have a symmetry in the sense that $f(x) = f(-x)$. Under our assumptions, the only global minima of this function are $x = \pm\sqrt{\lambda_1}v_1$. We are going to show that these are also the only second order stationary points. We will give two proof strategies that are commonly used to prove the locally optimizable property.

6.4.1 Characterizing all critical points

The first idea is simple – we will just try to solve the Equation $\nabla f(x) = 0$ to get the position of all critical points; then for the critical points that are not the desired global minimum, try to prove that they are local maximum or saddle points.

Computing gradient and Hessian Before we solve the equation $\nabla f(x) = 0$ for the objective function $f(x)$ defined in Equation (6.6), we first give a simple way of computing the gradient and Hessian. We will first expand $f(x+\delta)$ (where δ should be thought of as a small pertur-

[7] Note that the only real assumption here is $\lambda_1 > \lambda_2$, so the top eigenvector is unique. Other inequalities are without loss of generality.

bation):

$$\begin{aligned}
f(x+\delta) &= \frac{1}{4}\|M - (x+\delta)(x+\delta)^\top\|_F^2 \\
&= \frac{1}{4}\|M - xx^\top - (x\delta^\top + \delta x^\top) - \delta\delta^\top\|_F^2 \\
&= \frac{1}{4}\|M - xx^\top\|_F^2 - \frac{1}{2}\langle M - xx^\top, x\delta + \delta x^\top\rangle \\
&\quad + \left[\frac{1}{4}\|x\delta^\top + \delta x^\top\|_F^2 - \frac{1}{2}\langle M - xx^\top, \delta\delta^\top\rangle\right] + o(\|\delta\|_2^2).
\end{aligned}$$

Note that in the last step, we have collected the terms based on the degree of δ, and ignored all the terms that are smaller than $o(\|\delta\|_2^2)$. We can now compare this expression with the Taylor's expansion of $f(x+\delta)$:

$$f(x+\delta) = f(x) + \langle \nabla f(x), \delta\rangle + \frac{1}{2}\delta^\top[\nabla^2 f(x)]\delta + o(\|\delta\|_2^2).$$

By matching terms, we immediately have

$$\begin{aligned}
\langle \nabla f(x), \delta\rangle &= -\frac{1}{2}\langle M - xx^\top, x\delta^\top + \delta x^\top\rangle, \\
\delta^\top[\nabla^2 f(x)]\delta &= \frac{1}{2}\|x\delta^\top + \delta x^\top\|_F^2 - \langle M - xx^\top, \delta\delta^\top\rangle.
\end{aligned}$$

These can be simplified to give the actual gradient and Hessian[8]

$$\nabla f(x) = (xx^\top - M)x, \quad \nabla^2 f(x) = \|x\|_2^2 I + 2xx^\top - M. \tag{6.7}$$

Characterizing critical points Now we can execute the original plan. First set $\nabla f(x) = 0$, we have

$$Mx = xx^\top x = \|x\|_2^2 x.$$

Luckily, this is a well studied equation because we know the only solutions to $Mx = \lambda x$ are if λ is an eigenvalue and x is (a scaled version) of the corresponding eigenvector. Therefore we know $x = \pm\sqrt{\lambda_i}v_i$ or $x = 0$. These are the only critical points of the objective function $f(x)$.

Among these critical points, $x = \pm\sqrt{\lambda_1}v_1$ are our intended solutions. Next we need to show for every other critical point, its Hessian has a negative eigendirection. We will do this for $x = \pm\sqrt{\lambda_i}v_i (i > 1)$. By definition, it suffices to show there exists a δ such that $\delta^\top[\nabla^2 f(x)]\delta < 0$. The main step of the proof involves guessing what is this direction δ. In this case we will choose $\delta = v_1$ (we will give more intuitions about how to choose such a direction in the next subsection).

[8] In fact in the next subsection we will see it is often good enough to know how to compute $\langle \nabla f(x), \delta\rangle$ and $\delta^\top[\nabla^2 f(x)]\delta$.

When $x = \pm\sqrt{\lambda_i} v_i$, and $\delta = v_1$, we have

$$\delta^\top [\nabla^2 f(x)]\delta = v_1^\top [\|\sqrt{\lambda_i} v_i\|_2^2 I + 2\lambda_i v_i v_i^\top - M] v_1 = \lambda_i - \lambda_1 < 0.$$

Here the last step uses the fact that v_i's are orthonormal vectors and $v_1^\top M v_1 = \lambda_1$. The proof for $x = 0$ is very similar. Combining all the steps above, we proved the following claim:

Claim 6.4.1 (Properties of critical points)**.** *The only critical points of $f(x)$ are of the form $x = \pm\sqrt{\lambda_i} v_i$ or $x = 0$. For all critical points except $x = \pm\sqrt{\lambda_1} v_1$, $\nabla^2 f(x)$ has a negative eigenvalue.*

This claim directly implies that the only second order stationary points are $x = \pm\sqrt{\lambda_1} v_1$, so all second order stationary points are also global minima.

6.4.2 *Finding directions of improvements*

The approach in Section 6.4.1 is straight-forward. However, in more complicated problems it is often infeasible to enumerate all the solutions for $\nabla f(x) = 0$. What we proved in Section 6.4.1 is also not strong enough for showing $f(x)$ is locally optimizable, because we only proved every exact SOSP is a global minimum, and a locally optimizable function requires every approximate SOSP to be close to a global minimum. We will now give an alternative approach that is often more flexible and robust.

For every point x that is not a global minimum, we define its direction of improvements as below:

Definition 6.4.2 (Direction of improvement)**.** *For an objective function f and a point x, we say δ is a direction of improvement (of f at x) if $|\langle \nabla f(x), \delta \rangle| > 0$ or $\delta^\top [\nabla^2 f(x)]\delta < 0$. We say δ is an $(epsilon, \gamma)$ direction of improvement (of f at x) if $|\langle \nabla f(x), \delta \rangle| > \epsilon \|\delta\|_2$ or $\delta^\top [\nabla^2 f(x)]\delta < -\gamma \|\delta\|_2^2$.*

Intuitively, if δ is a direction of improvement for f at x, then moving along one of δ or $-\delta$ for a small enough step can decrease the objective function. In fact, if a point x has a direction of improvement, it cannot be a second order stationary point; if a point x has an $(epsilon, \gamma)$ direction of improvement, then it cannot be an (ϵ, γ)-SOSP.

Now we can look at the contrapositive of what we were trying to prove in the definition of locally optimizable functions: if every point x with $f(x) > f(x^*) + \tau$ has an (ϵ, γ) direction of improvement, then every (ϵ, γ)-second order stationary point must satisfy $f(x) \leq f(x^*) + \delta$. Therefore, our goal in this part is to find a direction of improvement for every point that is not globally optimal.

For simplicity, we will look at an even simpler version of the top eigenvector problem. In particular, we consider the case where $M = zz^\top$ is a rank-1 matrix, and z is a unit vector. In this case, the objective function we defined in Equation (6.6) becomes

$$\min_x f(x) = \frac{1}{4}\|zz^\top - xx^\top\|_F^2. \tag{6.8}$$

The intended global optimal solutions are $x = \pm z$. This problem is often called the matrix factorization problem as we are given a matrix $M = zz^\top$[9] and the goal is to find a decomposition $M = xx^\top$.

Which direction should we move to decrease the objective function? In this problem we only have the optimal direction z and the current direction x, so the natural guesses would be z, x or $z - x$. Indeed, these directions are enough:

Lemma 6.4.3. *For objective function (6.8), there exists a universal constant $c > 0$ such that for any $\tau < 1$, if neither x or z is an $(c\tau,, 1/4)$-direction of improvement for the point x, then $f(x) \le \tau$.*

The proof of this lemma involves some detailed calculation. To get some intuition, we can first think about what happens if neither x or z is a direction of improvement.

Lemma 6.4.4. *For objective function (6.8), if neither x or z is a direction of improvement of f at x, then $f(x) = 0$.*

Proof. We will use the same calculation for gradient and Hessian as in Equation (6.7), except that M is now $zz^\top$. First, since x is not a direction of improvement, we must have

$$\langle \nabla f(x), x\rangle = 0 \implies \|x\|_2^4 = \langle x, z\rangle^2. \tag{6.9}$$

If z is not a direction of improvement, we know $z^\top[\nabla^2 f(x)]z \ge 0$, which means

$$\|x\|^2 + 2\langle x, z\rangle^2 - 1 \ge 0 \implies \|x\|^2 \ge 1/3.$$

Here we used the fact that $\langle x, z\rangle^2 \le \|x\|_2^2\|z\|_2^2 = \|x\|_2^2$. Together with Equation (6.9) we know $\langle x, z\rangle^2 = \|x\|_2^4 \ge 1/9$.

Finally, since z is not a direction of improvement, we know $\langle \nabla f(x), z\rangle = 0$, which implies $\langle x, z\rangle(\|x\|_2^2 - 1) = 0$. We have already proved $\langle x, z\rangle^2 \ge 1/9 > 0$, thus $\|x\|_2^2 = 1$. Again combining with Equation (6.9) we know $\langle x, z\rangle^2 = \|x\|_2^4 = 1$. The only two vectors with $\langle x, z\rangle^2 = 1$ and $\|x\|_2^2 = 1$ are $x = \pm z$. □

The proof of Lemma 6.4.3 is very similar to Lemma 6.4.4, except we need to allow slacks in every equation and inequality we use. The additional benefit of having the more robust Lemma 6.4.3 is that the

[9] Note that we only observe M, not z.

proof is also robust if we don't have access to the exact objective - in settings where only a subset of coordinates of $zz^\top$[10], one can still prove that the objective function is locally optimizable, and hence find z by nonconvex optimization.

Lemma 6.4.4 and Lemma 6.4.3 both use directions x and z. It is also possible to use the direction $x - z$ when $\langle x, z\rangle \geq 0$ (and $x + z$ when $\langle x, z\rangle < 0$). Both ideas can be generalized to handle the case when $M = ZZ^\top$ where $Z \in \mathbb{R}^{d\times r}$, so M is a rank-r matrix.

[10] This setting is known as *matrix completion* and has been widely applied to recommendation systems.

7
Escaping Saddle Points

Gradient descent (GD) and stochastic gradient descent (SGD) are the workhorses of large-scale machine learning. While classical theory focused on analyzing the performance of these methods in *convex* optimization problems, the most notable successes in machine learning have involved *nonconvex* optimization, and a gap has arisen between theory and practice.

Indeed, traditional analyses of GD and SGD show that both algorithms converge to stationary points efficiently. But these analyses do not take into account the possibility of converging to saddle points. Motivated by the geometric characterizations in the last chapter, the central difficulty in solving many nonconvex machine learning problems becomes escaping saddle points.

In this chapter, we will discuss a simple perturbed form of gradient descent, which is capable of escaping saddle points very efficiently. Particularly, in terms of convergence rate and dimension dependence, it is almost as if the saddle points are not there!

7.1 Preliminaries

≪Chi notes: Many definitions have appeared in the earlier chapters. Coordinatition may be required.≫

In this chapter, we are interested in solving general unconstrained optimization problems of the form:

$$\min_{\mathrm{x}\in\mathbb{R}^d} f(\mathrm{x}),$$

where f is a smooth function that can be nonconvex. In particular we assume that f has Lipschitz gradients and Lipschitz Hessians, which ensures that the gradient and Hessian can not change too rapidly.

Definition 7.1.1. *A differentiable function f is **ℓ-gradient Lipschitz** if:*

$$\|\nabla f(x_1) - \nabla f(x_2)\| \leq \ell \, \|x_1 - x_2\| \quad \forall \, x_1, x_2.$$

Definition 7.1.2. *A twice-differentiable function f is ρ-**Hessian Lipschitz** if:*

$$\left\|\nabla^2 f(x_1) - \nabla^2 f(x_2)\right\| \leq \rho \left\|x_1 - x_2\right\| \quad \forall\, x_1, x_2.$$

For minimization problems, both saddle points and local maxima are clearly undesirable. Our focus will be "saddle points," although our results also apply directly to local maxima as well. Unfortunately, distinguishing saddle points from local minima for smooth functions is still NP-hard in general [Nesoo]. To avoid these hardness results, we focus on a subclass of saddle points.

Definition 7.1.3 (strict saddle point)**.** *For a twice-differentiable function f, x is a **strict saddle point** if x is a stationary point and $\lambda_{\min}(\nabla^2 f(x)) < 0$.*

A generic saddle point must satisfy that $\lambda_{\min}(\nabla^2 f(\mathrm{x})) \leq 0$. Being "strict" simply rules out the case where $\lambda_{\min}(\nabla^2 f(\mathrm{x})) = 0$. We reformulate our goal as that of finding stationary points that are not strict saddle points.

Definition 7.1.4 (SOSP)**.** *For twice-differentiable function $f(\cdot)$, x is a **second-order stationary point** if*

$$\nabla f(x) = 0, \quad \textit{and} \quad \nabla^2 f(x) \succeq 0.$$

Definition 7.1.5 (ϵ-SOSP)**.** *For a ρ-Hessian Lipschitz function $f(\cdot)$, x is an ϵ-**second-order stationary point** if:*

$$\|\nabla f(x)\| \leq \epsilon \quad \textit{and} \quad \nabla^2 f(x) \succeq -\sqrt{\rho\epsilon} \cdot I.$$

Definition 7.1.5 characterizes an ϵ-approximate version of SOSP so that we can discuss rates. The condition on the Hessian in Definition 7.1.5 uses the Hessian Lipschitz parameter ρ to retain a single accuracy parameter and to match the units of the gradient and Hessian [1] , following the convention of [NP06].

7.2 *Perturbed Gradient Descent*

According to the update equations, Gradient Descent (GD) makes a non-zero step only when the gradient is non-zero, and thus in the nonconvex setting it will be stuck at saddle points if initialized there. We thus consider a simple variant of GD which adds perturbations to the iterates at each step.

$$\mathrm{x}_{t+1} \leftarrow \mathrm{x}_t - \eta(\nabla f(\mathrm{x}_t) + \xi_t), \qquad \xi_t \sim \mathcal{N}(0, (r^2/d)\mathrm{I})$$

At each iteration, Perturbed Gradient Descent (PGD) is almost the same as gradient descent, except it adds a small isotropic random

[1] By matching the "units", we can make the optimization results invariant to simple rescaling $g(x) = af(bx)$ for scalar $a, b > 0$. Here, ρ has the scaling of third-order derivative, ϵ has the scaling of the first-order derivative, so $\sqrt{\rho\epsilon}$ has the scaling of the second-order derivative.

Gaussian perturbation to the gradient. The perturbation ξ_t is sampled from a zero-mean Gaussian with covariance $(r^2/d)\mathrm{I}$ so that $\mathbb{E}\|\xi_t\|^2 = r^2$. Parameter r control the effective radius of the perturbation, which is often chosen to be very small. [JNG+19] proves that this simple form of PGD is capable of escaping strict saddle points and finding SOSP efficiently.

In this chapter, to show the insights behind PGD, we turn to an alternative variant of the algorithm, which has a slightly more complicated form, but a easier analysis. The variant we consider here performs the following two steps at each iteration:

1. If $\|\nabla f(\mathrm{x}_t)\| \leq \epsilon$ and no perturbation has been added in the last $\mathcal{T}$ steps, then add small perturbation $\mathrm{x}_t \leftarrow \mathrm{x}_t - \eta\xi_t$ where $\xi_t \sim \text{Uniform}(\mathbb{B}_0(r)))$.

2. $\mathrm{x}_{t+1} \leftarrow \mathrm{x}_t - \eta\nabla f(\mathrm{x}_t)$.

where $\mathbb{B}_0(r)$ is the Euclidean ball centered at 0 with radius r. This variant of PGD only adds perturbations when gradient is small and no perturbation has been added in the last $\mathcal{T}$ steps, thus adds less stochasticity to the algorithm compared to the original form of PGD. In the following theorem, we provide theoretical guarantees for this variant of PGD as follows:

Theorem 7.2.1. *Assume f is ℓ-gradient Lipschitz, and ρ-Hessian Lipschitz. For any $\epsilon, \delta > 0$, if we choose $\eta = 1/\ell$, $r = \widetilde{\Theta}(\epsilon)$, $\mathcal{T} = \widetilde{\Theta}(\ell/\sqrt{\rho\epsilon})$, and run PGD for more than $\widetilde{O}(\ell(f(x_0) - f^\star)/\epsilon^2)$ iterations, then with probability at least $1-\delta$, at least one of the iterates will be ϵ-SOSP.*

Here $\widetilde{O}(\cdot), \widetilde{\Theta}(\cdot)$ hides absolute constant and poly-logarithmic dependence in $d, \ell, \rho, \epsilon, \delta$ and $f(x_0) - f^\star$. Our choice of $\mathcal{T}$ is, up to logarithmic factors, the ratio of the gradient Lipschitz parameter ℓ and the Hessian accuracy tolerance in ϵ-SOSP — $\sqrt{\rho\epsilon}$.

We remark that, in classic optimization literature, it is known that GD finds an ϵ-first-order stationary point (a point x satisfying $\|\nabla f(\mathrm{x})\| \leq \epsilon$) in $O(\ell(f(\mathrm{x}_0) - f^\star)/\epsilon^2)$ iterations [Nes98]. Theorem 7.2.1 shows that PGD finds second-order stationary points in almost the same time as GD finds first-order stationary points, up to only logarithmic factors. In particular, despite there might be only one escaping direction within the d-dimensional space at saddle points, the dimension dependency of PGD is only polylogarithmic. This is extremely important in high-dimensional settings, such as training deep neural networks. This provides a compelling explanation why strict saddle points are computationally benign for first-order gradient methods.

The overall proof strategy is as follows. According to Definition 7.1.5, if an iterate x_t is not an ϵ-second-order stationary point, then x_t

must either have large gradient or be an approximate saddle point. We prove the following two claims:

1. Large gradient ($\|\nabla f(\mathrm{x}_t)\| > \epsilon$), then function value decreases significantly in one step: $f(\mathrm{x}_{t+1}) - f(\mathrm{x}_t) \leq -\Omega(\epsilon^2/\ell)$.

2. Approximate saddle point ($\|\nabla f(\mathrm{x}_t)\| \leq \epsilon$ and $\lambda_{\min}(\nabla^2 f(\mathrm{x}_t)) < -\sqrt{\rho\epsilon}$), then, with high probability, function value decreases significantly in $\mathcal{T}$ steps: $f(\mathrm{x}_{t+\mathcal{T}}) - f(\mathrm{x}_t) \leq -\widetilde{\Omega}(\mathcal{T} \cdot \epsilon^2/\ell)$.

That is, in either case, the function value will decrease by $\widetilde{\Omega}(\epsilon^2/\ell)$ on average per step. Since the function value can be no less than the optimal value $f^\star$, we know after $\widetilde{O}(\ell(f(\mathrm{x}_0) - f^\star)/\epsilon^2)$ steps, at least one of the iterates must to ϵ-second-order stationary point. The first claim immediately follows from the descent lemma (Lemma 2.1.4). In the next section, we will show how to prove the second claim.

7.3 Saddle Points Escaping Lemma

In this section, we formally prove that if the starting point has a strictly negative eigenvalue of the Hessian, then adding a perturbation and following by gradient descent will yield a significant decrease in function value in $\mathcal{T}$ iterations.

Lemma 7.3.1 (Saddle Points Escaping Lemma)**.** *Under the setting of Theorem 7.2.1, if $\widetilde{x}$ satisfies $\|\nabla f(\widetilde{x})\| \leq \epsilon$, and $\lambda_{\min}(\nabla^2 f(\widetilde{x})) \leq -\sqrt{\rho\epsilon}$, then let $x_0 = \widetilde{x} + \eta\xi$ ($\xi \sim$ Uniform$(B_0(r))$), and run gradient descent starting from x_0. With probability at least $1-\delta$, we have*

$$f(x_{\mathcal{T}}) - f(\widetilde{x}) \leq -\widetilde{\Omega}(\mathcal{T} \cdot \epsilon^2/\ell)$$

where $x_{\mathcal{T}}$ is the $\mathcal{T}^{th}$ gradient descent iterate starting from x_0.

Recall that $\mathcal{T} = \widetilde{\Theta}(\ell/\sqrt{\rho\epsilon})$. This implies that both saddle point escaping time, and the amount of function decrease depend on dimension d only polylogarithmically. To prove this lemma, we will show the followings:

- (*Improve or Localize*) If gradient descent keeps making little progress for a certain number of iterations, then all the iterates within those iterations must be stuck in a small Euclidean ball.

- (*Stuck probability is small around saddle points*) If the Hessian has a significant negative eigenvalue, then after a random perturbation, with high probability, gradient descent will not be stuck in a small Euclidean ball for a long time.

Combining two statements above, we conclude that GD in the second statement must make significant progress after a certain number of iterations, which proves Lemma 7.3.1.

7.3.1 *Improve or Localize*

We first prove the following lemma which says that if the function value does not decrease too much over t iterations, then all iterates $\{\mathrm{x}_\tau\}_{\tau=0}^{t}$ will remain in a small neighborhood of x_0.

Lemma 7.3.2 (Improve or Localize)**.** *Assume function f is ℓ-gradient Lipschitz, and run GD with $\eta \le 1/\ell$, then for any $t \ge \tau > 0$, we have:*

$$\|x_\tau - x_0\| \le \sqrt{2\eta t (f(x_0) - f(x_t))}.$$

Proof. Given the gradient update, $\mathrm{x}_{t+1} = \mathrm{x}_t - \eta \nabla f(\mathrm{x}_t)$, we have that for any $\tau \le t$:

$$\begin{aligned}\|\mathrm{x}_\tau - \mathrm{x}_0\| &\le \sum_{\tau=1}^{t} \|\mathrm{x}_\tau - \mathrm{x}_{\tau-1}\| \overset{(1)}{\le} [t \sum_{\tau=1}^{t} \|\mathrm{x}_\tau - \mathrm{x}_{\tau-1}\|^2]^{\frac{1}{2}} \\ &= [\eta^2 t \sum_{\tau=1}^{t} \|\nabla f(\mathrm{x}_{\tau-1})\|^2]^{\frac{1}{2}} \overset{(2)}{\le} \sqrt{2\eta t (f(\mathrm{x}_0) - f(\mathrm{x}_t))},\end{aligned}$$

where step (1) uses Cauchy-Schwarz inequality, and step (2) is due to the descent lemma (Lemma 2.1.4). □

Lemma 7.3.2 immediately implies that if $f(\mathrm{x}_{\mathcal{T}}) - f(\mathrm{x}_0) \ge -\tilde{O}(\mathcal{T} \cdot \epsilon^2/\ell)$, i.e. GD does not make enough progress in $\mathcal{T}$ steps after perturbation, then we immediately have that $\|\mathrm{x}_t - \mathrm{x}_0\| \le \tilde{O}(\epsilon\mathcal{T}/\ell)$ for all $t \in [\mathcal{T}]$.

7.3.2 *Bounding the Width of the Stuck Region*

Second, we show that the probability for GD sequence to get stuck is small if initialized with a point around saddle point with random perturbation. Recall in Lemma 7.3.1 that $\mathrm{x}_0 \sim \mathrm{Uniform}(\mathbb{B}_{\tilde{\mathrm{x}}}(\eta r))$. We refer to $\mathbb{B}_{\tilde{\mathrm{x}}}(\eta r)$ as the *perturbation ball*, and define the *stuck region* within the perturbation ball to be the set of points starting from which GD makes little progress in $\mathcal{T}$ steps:

$$\begin{aligned}\mathcal{X}_{\mathrm{stuck}} := \{\mathrm{x} \in B_{\tilde{\mathrm{x}}}(\eta r) \mid \{\mathrm{x}_t\}_{t=0}^{\mathcal{T}} &\text{ is a GD sequence with} \\ &\mathrm{x}_0 = \mathrm{x}, \text{and } \forall t \in [\mathcal{T}], \|\mathrm{x}_t - \mathrm{x}_0\| \le \tilde{O}(\epsilon\mathcal{T}/\ell)\}.\end{aligned}$$

See Figure 7.1 for illustrations. Since x_0 sampled uniformly from this perturbation ball, the probability GD got stuck after perturbation is equal to the ratio of the volume of the stuck region and the volume of the perturbation ball. Therefore, we want to show that the stuck region has small volume.

In general, the shape of the stuck region can be very complicated, so it is very difficult to directly compute its volume. A crucial observation here is that, despite we do not know the shape of the stuck

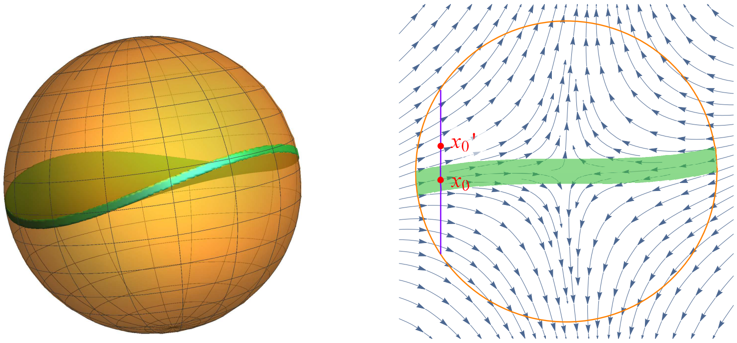


Figure 7.1: **Left:** Pertubation ball in 3D and "thin pancake" shape stuck region. **Right:** Pertubation ball in 2D and "narrow band" stuck region under gradient flow

region, we can prove the width of $\mathcal{X}_{\text{stuck}}$ along the minimum eigenvalue direction of $\nabla^2 f(\widetilde{x})$ is small. In fact, if the width is at most $\eta\omega$, then we have $\text{Vol}(\mathcal{X}_{\text{stuck}}) \leq \text{Vol}(\mathbb{B}_0^{d-1}(\eta r))\eta\omega$, and thus,

$$\Pr(\mathrm{x}_0 \in \mathcal{X}_{\text{stuck}}) = \frac{\text{Vol}(\mathcal{X}_{\text{stuck}})}{\text{Vol}(\mathbb{B}_{\widetilde{\mathrm{x}}}^d(\eta r))} \leq \frac{\eta\omega \times \text{Vol}(\mathbb{B}_0^{d-1}(\eta r))}{\text{Vol}(\mathbb{B}_0^d(\eta r))}$$
$$= \frac{\omega}{r\sqrt{\pi}} \frac{\Gamma(\frac{d}{2}+1)}{\Gamma(\frac{d}{2}+\frac{1}{2})} \leq \frac{\omega}{r} \cdot \sqrt{\frac{d}{\pi}}$$

To achieve failure probability at most δ, we hope $\omega \leq O(\delta r/\sqrt{d})$. We bound the width of the stuck region $\mathcal{X}_{\text{stuck}}$ by the novel technics of **coupling sequences**—consider two GD sequences $\{\mathrm{x}_t\}_{t=0}^{\mathcal{T}}, \{\mathrm{x}'_t\}_{t=0}^{\mathcal{T}}$ which satisfy: (1) $\max\{\|\mathrm{x}_0 - \widetilde{\mathrm{x}}\|, \|\mathrm{x}'_0 - \widetilde{\mathrm{x}}\|\} \leq \eta r$; and (2) $\mathrm{x}_0 - \mathrm{x}'_0 = \eta\omega \mathrm{e}_1$, where e_1 is the minimum eigenvector of $\nabla^2 f(\widetilde{\mathrm{x}})$, and $\omega \geq \omega_0$ for some threshold ω_0.

Lemma 7.3.3. *For any $\omega_0 \in (0, \epsilon]$, under the setting of Lemma 7.3.1, if $\{x_t\}_{t=0}^{\mathcal{T}}, \{x'_t\}_{t=0}^{\mathcal{T}}$ are coupling sequences as specified above, then for $\mathcal{T} \geq \Omega(\kappa \cdot \log(\epsilon\kappa/\omega_0))$ where $\kappa := \ell/\sqrt{\rho\epsilon}$, we have,*

$$\exists t \in [\mathcal{T}], \quad \max\{\|x_t - x_0\|, \|x'_t - x'_0\|\} \geq \widetilde{\Omega}(\epsilon\mathcal{T}/\ell)$$

Lemma 7.3.3 claims that for any pair of x_0, x'_0 whose difference is on the e_1 direction, with length greater or equal to $\eta\omega_0$, at least one of x_0, x'_0 is outside $\mathcal{X}_{\text{stuck}}$. This directly implies the width of $\mathcal{X}_{\text{stuck}}$ in e_1 direction is $\eta\omega_0$. Lemma 7.3.3 further claims that the width $\eta\omega_0$ can be made arbitrarily small by paying only logarithmic factors in the choice of $\mathcal{T}$.

Proof. We prove by contradiction. Assume the contrary of Lemma 7.3.3 is true—$\max\{\|\mathrm{x}_t - \mathrm{x}_0\|, \|\mathrm{x}'_t - \mathrm{x}'_0\|\} \leq \widetilde{O}(\epsilon\mathcal{T}/\ell)$ for all $t \in [\mathcal{T}]$, i.e. both GD sequences stuck in a small Euclidean ball for $\mathcal{T}$ steps.

We can write out the update equation for the difference of the couple sequences $\widehat{\mathrm{x}}_t := \mathrm{x}_t - \mathrm{x}'_t$ as:

$$\begin{aligned}\widehat{\mathrm{x}}_{t+1} =&\widehat{\mathrm{x}}_t - \eta[\nabla f(\mathrm{x}_t) - \nabla f(\mathrm{x}'_t)] = (\mathrm{I} - \eta\mathcal{H})\widehat{\mathrm{x}}_t - \eta\Delta_t\widehat{\mathrm{x}}_t \\ =&\underbrace{(\mathrm{I} - \eta\mathcal{H})^{t+1}\widehat{\mathrm{x}}_0}_{\mathrm{p}(t+1)} - \underbrace{\eta\sum_{\tau=0}^{t}(\mathrm{I} - \eta\mathcal{H})^{t-\tau}\Delta_\tau\widehat{\mathrm{x}}_\tau}_{\mathrm{q}(t+1)},\end{aligned}$$

where $\mathcal{H} = \nabla^2 f(\widetilde{\mathrm{x}})$ and $\Delta_t = \int_0^1[\nabla^2 f(\mathrm{x}'_t + \theta(\mathrm{x}_t - \mathrm{x}'_t) - \mathcal{H}]\mathrm{d}\theta$. We note that term $\mathrm{p}(t)$ is the formula of $\widehat{\mathrm{x}}_t$ if function f is quadratic around $\widetilde{\mathrm{x}}$, and $\mathrm{q}(t)$ is the approximation error term caused by function f being non-quadratic.

We will show later that the quadratic term is the dominating term, in the sense that $\|\mathrm{q}(t)\| \le \|\mathrm{p}(t)\|/2$ for all $t \in [\mathcal{T}]$. Given this is true, since $\widehat{\mathrm{x}}_0 = \eta\omega\mathrm{e}_1$, we have

$$\|\mathrm{p}(t)\| \ge (1 + \sqrt{\epsilon\rho}/\ell)^t \cdot \eta\omega_0$$

This term grows exponentially. Therefore, by choosing $\mathcal{T} \ge \Omega(\kappa \cdot \log(\epsilon\kappa/\omega_0))$, we have $\|\widehat{\mathrm{x}}_t\| \ge \|p(t)\|/2 \ge \widetilde{\Omega}(\epsilon\mathcal{T}/\ell)$, which contradicts the assumption that both GD sequences stuck in a small Euclidean ball with radius $\widetilde{O}(\epsilon\mathcal{T}/\ell)$ for $\mathcal{T}$ steps (we note $\|\mathrm{x}_0 - \mathrm{x}'_0\| \le 2\eta r \ll \widetilde{O}(\epsilon\mathcal{T}/\ell)$). This proves Lemma 7.3.3.

For the remaining of the proof, we only need to verify by induction that $\|\mathrm{q}(t)\| \le \|\mathrm{p}(t)\|/2$ for all $t \in [\mathcal{T}]$. The claim is true for the base case $t = 0$ as $\|\mathrm{q}(0)\| = 0 \le \|\widehat{\mathrm{x}}_0\|/2 = \|\mathrm{p}(0)\|/2$. Now suppose the induction claim is true up to t. Denote $\lambda_{\min}(\mathcal{H}) = -\gamma$. Note that $\widehat{\mathrm{x}}_0$ lies in the direction of the minimum eigenvector of $\mathcal{H}$. Thus for any $\tau \le t$, we have:

$$\|\widehat{\mathrm{x}}_\tau\| \le \|\mathrm{p}(\tau)\| + \|\mathrm{q}(\tau)\| \le 2\|\mathrm{p}(\tau)\| = 2(1 + \eta\gamma)^\tau\eta\omega.$$

By the Hessian Lipschitz property, we further have

$$\|\Delta_t\| \le \rho\max\{\|\mathrm{x}_t - \widetilde{\mathrm{x}}\|, \|\mathrm{x}'_t - \widetilde{\mathrm{x}}\|\} \le \widetilde{O}(\rho\epsilon\mathcal{T}/\ell) = \widetilde{O}(\sqrt{\rho\epsilon})$$

therefore:

$$\begin{aligned}\|\mathrm{q}(t+1)\| =& \left\|\eta\sum_{\tau=0}^{t}(\mathrm{I} - \eta\mathcal{H})^{t-\tau}\Delta_\tau\widehat{\mathrm{x}}_\tau\right\| \\ \le&\eta\sum_{\tau=0}^{t}\|\Delta_t\|\left\|(\mathrm{I} - \eta\mathcal{H})^{t-\tau}\right\|\|\widehat{\mathrm{x}}_\tau\| \le \widetilde{O}(\eta\sqrt{\rho\epsilon})\sum_{\tau=0}^{t}(1 + \eta\gamma)^t\eta\omega \\ \le&\widetilde{O}(1)(1 + \eta\gamma)^t\eta\omega \le \widetilde{O}(1)\|\mathrm{p}(t+1)\|,\end{aligned}$$

where the second-to-last inequality uses $t + 1 \le \mathcal{T}$, and $\widetilde{O}(\eta\mathcal{T}\sqrt{\rho\epsilon}) = \widetilde{O}(1)$. Finally, with a careful treatment of constant and logarithmic factors, we can in fact make this $\widetilde{O}(1)$ term less or equal to $1/2$ (we omit the detail here). This finishes the inductive proof. □

8
Algorithmic Regularization

Large scale neural networks used in practice are highly over-parameterized with far more trainable model parameters compared to the number of training examples. Consequently, the optimization objectives for learning such high capacity models have many global minima that fit training data perfectly. However, minimizing the training loss using specific optimization algorithms take us to not just any global minima, but some special global minima – in this sense the choice of optimization algorithms introduce a implicit form of inductive bias in learning which can aid generalization.

In over-parameterized models, specially deep neural networks, much, if not most, of the inductive bias of the learned model comes from this implicit regularization from the optimization algorithm. For example, early empirical work on this topic (ref. [NTS15a, NSS15, HS97, KMN+16, ZBH+16a, CCS+16, DPBB17, ADG+16, Ney17, WRS+17, HHS17, Smi18]) show that deep models often generalize well even when trained purely by minimizing the training error without any explicit regularization, and even when the networks are highly overparameterized to the extent of being able to fit random labels. Consequently, there are many zero training error solutions, all global minima of the training objective, most of which generalize horribly. Nevertheless, our choice of optimization algorithm, typically a variant of gradient descent, seems to prefer solutions that do generalize well. This generalization ability cannot be explained by the capacity of the explicitly specified model class (namely, the functions representable in the chosen architecture). Instead, the optimization algorithm biasing toward a “simple" model, minimizing some implicit “regularization measure”, say $R(\mathrm{w})$, is key for generalization. Understanding the implicit inductive bias, *e.g.* via characterizing $R(\mathrm{w})$, is thus essential for understanding how and what the model learns. For example, in linear regression it can be shown that minimizing an under-determined model (with more parameters than samples) using gradient descent yields the minimum ℓ_2 norm solution (see Proposition 8.1.1), and for linear logistic regression trained on linearly separable data, gradient descent converges in the direction of the hard margin support vector machine solution (Theorem 8.3.2), even though the norm or margin is not explicitly specified in the optimization problem. In fact, such analysis showing implicit inductive bias from optimization agorithm leading to generalization is not new. In the context of boosting algorithms, **(author?)** [EHJT04] and **(author?)** [Tel13] established connections of gradient boosting algorithm (coordinate descent) to ℓ_1 norm minimiziation, and ℓ_1 margin maximization, respectively. minimization was observed. Such minimum norm or maximum margin solutions are of course very special among all solutions or separators that fit the training data, and in particular can ensure generalization [BM03, KST09].

In this chapter, we largely present results on algorithmic regularization of vanilla gradient descent when minimizing unregularized training loss in regression and classification problem over various simple and complex model classes. We briefly discuss general algorithmic families like steepest descent and mirror descent.

Meanings of "implicit regularization due to training algorithm."

Results in the current chapter tend to show that the solution obtained by applying training algorithm A on *Objective 1* essentially to convergence (e.g. to stationary point of gradient descent), also satisfies KKT local optimality conditions for some other *Objective 2*. In many results Objective 2 is simply Objective 1 with a regularizer term, typically involving some norm of the solution. Hence we can think of the training algorithm as *implicitly regularizing* the objective.

While these results give good insight into the effect of the training a few caveats are in order, especially if we seek takeaways for deep learning. First, even though the solution found happens to be a KKT point of Objective 2, it may be never (or almost never) observed if we actually do standard training on Objective 2. [1] Second, the results in this chapter are often stated for training carried out to infinite time, which may also limit their applicability to real life.

In later chapters we will see a different type of analysis, which analyses the trajectory followed by the solution as it evolves during training. This *dynamic* view of training quickly gets complicated (as opposed to the more static view taken in understanding stationary points) and has not been achieved for realistic deep nets yet.

8.1 Linear models in regression: squared loss

suriya: pls see chapter 3 and modify the writeup as needed.

We first demonstrate the algorithmic regularization in a simple

[1] Recall that in a nonconvex landscape the solution obtained at the end of training is greatly affected by the initialization, and in deep learning the initialization is very special.

linear regression setting where the prediction function is specified by a linear function of inputs: $f_w(x) = w^\top x$ and we have the following empirical risk minimzation objective.

$$L(w) = \sum_{i=1}^{n} \left(w^\top x^{(i)} - y^{(i)} \right)^2 . \tag{8.1}$$

Such simple modes are natural starting points to build analytical tools for extending to complex models, and such results provide intuitions for understaning and improving upon the empirical practices in neural networks. Although the results in this section are specified for squared loss, the results and proof technique extend for any smooth loss a unique finite root: where $\ell(\widehat{y}, y)$ between a prediction $\widehat{y}$ and label y is minimized at a unique and finite value of $\widehat{y}$ [GLSS18a].

We are particularly interested in the case where $n < d$ and the observations are realizable, i.e., $\min_w L(w) = 0$. Under these conditions, the optimization problem in eq. (8.1) is underdetermined and has multiple global minima denoted by $\mathcal{G} = \{w : \forall i,\ w^\top x^{(i)} = y^{(i)}\}$. In this and all the following problems we consider, the goal is to answer: *Which specific global minima do different optimization algorithms reach when minimizing $L(w)$?*

The following proposition is the simplest illustration of the algorithmic regularization phenomenon.

Proposition 8.1.1. *Consider gradient descent updates w_t for the loss in eq.* (8.1) *starting with initialization w_0. For any step size schedule that minimizes the loss $L(w)$, the algorithm returns a special global minimizer that implicitly also minimzes the Euclidean distance to the initialization:* $w_t \to \underset{w \in \mathcal{G}}{argmin} \|w - w_0\|_2$.

Proof. The key idea is in noting that that the gradients of the loss function have a special structure. For the linear regression loss in eq. (8.1) $\forall w,\ \nabla L(w) = \sum_i (w^\top x^{(i)} - y^{(i)}) x^{(i)} \in \text{span}(\{x^{(i)}\})$ - that is the gradients are restricted to a n dimentional subspace that is independent of w. Thus, the gradient descent updates from iniitalization $w_t - w_0 = \sum_{t' < t} \eta w_{t'}$, which linearly accumulate gradients, are again constrained to the n dimensional subspace. It is now easy to check that there is a unique global minimizer that both fits the data ($w \in \mathcal{G}$) as well as is reacheable by gradient descent ($w \in w_0 + \text{span}(\{x^{(i)}\})$). By checking the KKT conditions, it can be verified that this unique minimizer is given by $\text{argmin}_{w \in \mathcal{G}} \|w - w_0\|_2^2$. □

In general overparameterized optimization problems, the characterization of the implicit bias or algorithmic regulariztion is often not this elegant or easy. For the same model class, changing the algorithm, or changing associated hyperparameter (like step size and initialization), or even changing the specific parameterization of the model class can change the implicit bias. For example, **(author?)** [WRS$^+$17] showed that for some standard deep learning architectures, variants of SGD algorithm with different choices of momentum and adaptive gradient updates (AdaGrad and Adam) exhibit different biases and thus have different generalization performance;**(author?)** [KMN$^+$16], **(author?)** [HHS17] and **(author?)** [Smi18] study how the size of the mini-batches used in SGD influences generalization; and **(author?)** [NSS15] compare the bias of path-SGD (steepest descent with respect to a scale invariant path-norm) to standard SGD.

A comprehensive understanding of how all the algorithmic choices affect the implicit bias is beyond the scope of this chapter (and also the current state of research). However, in the context of this chapter, we specifically want to highlight the role of *geometry* induced by optimization algorithm and specific parameterization, which are discussed briefly below.

8.1.1 Geometry induced by updates of local search algorithms

The relation of gradient descent to implicit bias towards minimizing Euclidean distance to initialization is suggestive of the connection between algorithmic regularization to the geometry of updates in local search methods. In particular, gradient descent iterations can be alternatively specified by the following equation where the $t+1$th iterate is derived by minimizing the a local (first order Taylor) approximation of the loss while constraining the step length in Euclidean norm.

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \langle w, \nabla L(w_t) \rangle + \frac{1}{2\eta} \|w - w_t\|_2^2 . \tag{8.2}$$

Motivated by the above connection, we can study other families of algorithms that work under different and non-Euclidean geometries. Two convenient families are mirror descent w.r.t. potential R [BT03, NY83] and steepest descent w.r.t. general norms [BV04].

Mirror descent w.r.t. potential R Mirror descent updates are defined for any strongly convex and differentiable potential R as

$$\begin{aligned} w_{t+1} &= \arg\min_{w} \eta \langle w, \nabla L(w_t) \rangle + D_R(w, w_t), \\ \Longrightarrow \nabla R(w_{t+1}) &= \nabla R(w_t) - \eta \nabla L(w_t) \end{aligned} \tag{8.3}$$

where $D_R(w, w') = R(w) - R(w') - \langle \nabla R(w'), w - w' \rangle$ is the *Bregman divergence* [Bre67] w.r.t. R. This family captures updates where the geometry is specified by the Bregman divergence D_R. Examples of potentials R for mirror descent include the squared ℓ_2 norm

$R(w) = {}^1\!/_2\|w\|_2^2$, which leads to gradient descent; the entropy potential $R(w) = \sum_i w[i] \log w[i] - w[i]$; the spectral entropy for matrix valued w, where $R(\mathrm{w})$ is the entropy potential on the singular values of w; general quadratic potentials $R(w) = {}^1\!/_2\|w\|_D^2 = {}^1\!/_2\, w^\top D w$ for any positive definite matrix D; and the squared ℓ_p norms for $p \in (1, 2]$.

From eq. (8.3), we see that rather than w_t (called primal iterates), it is the $\nabla R(w_t)$ (called dual iterates) that are constrained to the low dimensional data manifold $\nabla R(w_0) + \mathrm{span}(\{x^{(i)}\})$. The arguments for gradient descent can now be generalized to get the following result.

Theorem 8.1.2. *For any realizable dataset $\{x^{(i)}, y^{(i)}\}_{n=1}^N$, and any strongly convex potential R, consider the mirror descent iterates w_t from eq.* (8.3) *for minimizing the empirical loss $L(w)$ in eq.* (8.1)*. For any initializations w_0 and any step-size schedule , if w_t converges to some zero-loss solution w^*, then it holds that*

$$w^* = \underset{w: \forall i, w^\top x^{(i)} = y^{(i)}}{\arg\min} \; D_R(w, w_0). \tag{8.4}$$

In particular, if we start at $w_0 = \arg\min_{\mathrm{w}} R(w)$ (so that $\nabla R(w_0) = 0$), then we get to $\arg\min_{w \in \mathcal{G}} R(w)$. [2]

Proof of Theorem 8.1.2. From Equation (8.3), we have $\nabla R(w_t) = \nabla R(w_0) + \sum_{i=1}^n \lambda_i x_i$ for some $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$. Thus w^* is a stationary point of loss function $w \mapsto D_R(w, w_0) - \sum_{i=1}^n \lambda_i x_i$. Since this function is convex, w^* is a global minimum, and thus a constrained global minimum of the loss function among all interpolating solutions. □

Steepest descent w.r.t. general norms Gradient descent is also a special case of steepest descent (SD) w.r.t a generic norm $\|.\|$ [BV04] with updates given by,

$$w_{t+1} = w_t + \eta_t \Delta w_t, \text{ where } \Delta w_t = \arg\min_v \langle \nabla L(w_t), v \rangle + \frac{1}{2}\|v\|^2. \tag{8.5}$$

Examples of steepest descent include gradient descent, which is steepest descent w.r.t ℓ_2 norm and coordinate descent, which is steepest descent w.r.t ℓ_1 norm. In general, the update Δw_t in eq. (8.5) is not uniquely defined and there could be multiple direction Δw_t that minimize eq. (8.5). In such cases, any minimizer of eq. (8.5) is a valid steepest descent update.

Generalizing gradient descent and mirror descent, we might expect the steepest descent iterates to converge to the solution closest to initialization in corresponding norm, $\arg\min_{w \in \mathcal{G}} \|w - w_0\|$. This is indeed the case for quadratic norms $\|v\|_D = \sqrt{v^\top D v}$ when eq. 8.5 is

[2] The analysis of Theorem 8.1.2 and Proposition 8.1.1 also hold when instancewise stochastic gradients are used in place of $\nabla L(w_t)$.

equivalent to mirror descent with $R(w) = 1/2\|w\|_D^2$. Unfortunately, this does not hold for general norms as shown by the following results.

Example 1. In the case of coordinate descent, which is a special case of steepest descent w.r.t. the ℓ_1 norm, **(author?)** [EHJT04] studied this phenomenon in the context of gradient boosting: obseving that sometimes but *not always* the optimization path of coordinate descent given by $\Delta w_{t+1} \in \text{conv}\left\{-\eta_t \frac{\partial L(w_t)}{\partial w[j_t]} e_{j_t} : j_t = \text{argmax}_j \left|\frac{\partial L(w_r)}{\partial w[j]}\right|\right\}$, coincides with the ℓ_1 *regularization path* given by, $\widehat{w}(\lambda) = \arg\min_w L(w) + \lambda\|w\|_1$. The specific coordinate descent path where updates average all the optimal coordinates and the step-sizes are infinitesimal is equivalent to forward stage-wise selection, a.k.a. ϵ-boosting [Fri01]. When the ℓ_1 regularization path $\widehat{w}(\lambda)$ is monotone in each of the coordinates, it is identical to this stage-wise selection path, i.e., to a coordinate descent optimization path (and also to the related LARS path) [EHJT04]. In this case, at the limit of $\lambda \to 0$ and $t \to \infty$, the optimization and regularization paths, both converge to the minimum ℓ_1 norm solution. However, when the regularization path $\widehat{w}(\lambda)$ is not monotone, which can and does happen, the optimization and regularization paths diverge, and forward stage-wise selection can converge to solutions with sub-optimal ℓ_1 norm.

Example 2. The following example shows that even for ℓ_p norms where the $\|.\|_p^2$ is smooth and strongly convex, the global minimum returned by the steepest descent depends on the step-size.
Consider minimizing $L(w)$ with dataset $\{(x^{(1)} = [1,1,1], y^{(1)} = 1), (x^{(2)} = [1,2,0], y^{(2)} = 10)\}$ using steepest descent updates w.r.t. the $\ell_{4/3}$ norm. The empirical results for this problem in Figure 8.1 clearly show that steepest descent converges to a global minimum that depends on the step-size and even in the continuous step-size limit of $\eta \to 0$, w_t *does not* converge to the expected solution of $\arg\min_{w \in \mathcal{G}} \|w - w_0\|$.

In summary, for squared loss, we characterized the implicit bias of generic mirror descent algorithm in terms of the potential function and initialization. However, even in simple linear regression, for steepest descent with general norms, we were unable to get a useful characterization. In contrast, in Section 8.3.2, we study logistic like strictly monotonic losses used in classification, where we *can* get a characterization for steepest descent.

8.1.2 Geometry induced by parameterization of model class

In many learning problems, the same model class can be parameterized in multiple ways. Given a parameter space $\mathbb{R}^d$ and a parametrized model f_w mapping input x to output $f_w(x)$, we consider a new pa-

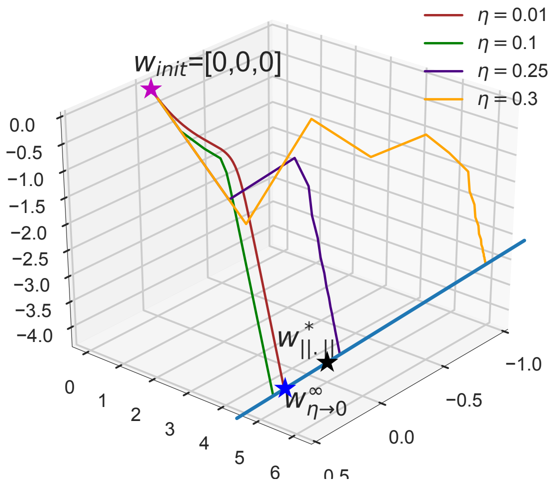


Figure 8.1: **Steepest descent w.r.t $\|.\|_{4/3}$:** the global minimum to which steepest descent converges to depends on η. Here $w_0 = [0,0,0]$, $w^*_{\|.\|} = \arg\min_{R\in G} \|w\|_{4/3}$ denotes the minimum norm global minimum, and $w^\infty_{\eta\to 0}$ denotes the solution of infinitesimal SD with $\eta \to 0$. Note that even as $\eta \to 0$, the expected characterization does not hold, i.e., $w^\infty_{\eta\to 0} \neq w^*_{\|.\|}$.

rameter space $\mathbb{R}^D$ where $D \geq d$ and a *parametrization*, which is a surjective map $G : \theta \mapsto G(\theta)$ from $\mathbb{R}^D$ to $\mathbb{R}^d$. The parametrization G induces a new parametrized model $\widetilde{f}_\theta(x) \triangleq f_{G(\theta)}(x)$, for all input x. For example, the set of linear functions in $\mathbb{R}^d$ can be parameterized in a canonical way as $w \in \mathbb{R}^d$ with $f_w(x) = w^\top x$, but also equivalently by $\theta = (u, v)$ where $u, v \in \mathbb{R}^d$ with $\widetilde{f}_\theta(x) = f_{u\odot v}(x) = (u \odot v)^\top x$ or $\widetilde{f}_\theta(x) = f_{u^{\odot 2} - v^{\odot 2}}(x) = (u^{\odot 2} - v^{\odot 2})^\top x$. All such equivalent parameterizations lead to equivalent model class, however, in overparemterized models, using vanilla gradient descent on different parameterizations lead to different trajectories $\{G(\theta_t)\}_{t\in\mathbb{N}}$ in the original parameter space $\mathbb{R}^d$, and thus different induced biases in the function space. The reason behind this phenomenon is that, though vanilla gradient descent finds the steepest descent direction w.r.t. ℓ_2-norm, but the parametrization G does not necessarily preserve the ℓ_2 distance and thus distort the geometry of local descent. For example, **(author?)** [GWB+17, GLSS18b] demonstrated this phenomenon in matrix factorization and linear convolutional networks, where these parameterizations were shown to introduce interesting and unusual biases towards minimizing nuclear norm, and ℓ_p (for $p = 2/\text{depth}$) norm in Fourier domain, respectively. In general, these results are suggestive of role of architecture choice in different neural network models, and shows how even while using the same gradient descent algorith, different geometries in the function space can be induced by the different parameterizations.

8.1.3 Equivalence between geometry inducded by local search algorithms and reparametrization

In the previous sections, we saw that both the parametrization of the model class and the local search method of optimization algorithm can induce a different geometry for the optimization landscape. In this section we show that the two geometries can be equivalent, that is, the two optimization trajectories for loss L, w_t and $G(\theta_t)$, are the same for continuous gradient/mirror descents, where x_t follows Mirror Flow Equation (8.6)

$$\frac{d\nabla R(w_t)}{dt} = -\nabla L(w_t), \tag{8.6}$$

and θ_t follows Reparametrized Gradient Flow Equation (8.7)

$$\frac{d\theta_t}{dt} = -\nabla(L \circ G)(\theta_t) \tag{8.7}$$

We stress that though the optimization geometry depends on L, the main equivalence result that will be presented in this section, is a property of the parametrization G, the potential R, and potentially the initialization θ_0, $w_0 = G(\theta_0)$. In particular, when the equivalence holds, it simultaneously holds for all differentiable loss L.

Below we present the intuition behind the equivalence. Mirror Flow Equation (8.6) can be alternatively written as:

$$\frac{dw_t}{dt} = -(\nabla^2 R(w_t))^{-1}\nabla L(w_t). \tag{8.8}$$

And Reparametrized Gradient Flow Equation (8.7) yields the following trajectory in the w-space, $\mathbb{R}^d$:

$$\frac{dG(\theta_t)}{dt} = -\partial G(\theta_t)\partial G(\theta_t)^\top \nabla L(G(\theta_t)). \tag{8.9}$$

Thus for the two trajectories $G(\theta_t)$ and w_t to be the same, it suffices to require:

$$\partial G(\theta_t)\partial G(\theta_t)^\top = (\nabla^2 R(G(\theta_t)))^{-1}, \quad \forall t \geq 0. \tag{8.10}$$

If Equation (8.10) holds, then both $G(\theta_t)$ and w_t satisfy the same differential equation, and thus are the same. A even stronger (but also more convenient) sufficient condition is the following:

$$\partial G(\theta)\partial G(\theta)^\top = (\nabla^2 R(G(\theta)))^{-1}, \quad \forall \theta. \tag{8.11}$$

Equation (8.11) is easier to check because it requires no understanding of the optimization trajectories θ_t. Below are two examples where equivalenec holds because Equation (8.11) is satisfied:

Example 8.1.3 (Quadratic Mirror Map and Linear Reparametrization)**.** *The geometry induced by mirror descent with quadratic potential $R(w) = w^\top \Sigma^{-1} w/2$ is equivalent to the geometry induced by gradient descent with reparametrization $G(\theta) = \sqrt{\Sigma}\theta$ for any positive definite matrix $\Sigma \in \mathbb{R}^{d\times d}$.*

Example 8.1.4 (Entropy Mirror Map and Quadratic Reparametrization)**.** *The geometry induced by mirror descent with entropy potential $R(w) = {}^1\!/\!_4 \sum_i w[i] \log w[i] - w[i]$ is equivalent to the geometry induced by gradient descent with reparametrization $G(\theta) = \theta^{\odot 2}$.*

However, Equation (8.11) is not necessary for the equivalence to hold. In fact, there is a large class of examples where Equation (8.11) does not hold, but the equivalence still holds. SUch examples typically have a non-invertible parametrization G, which further allows two different parameters θ_1 and θ_2 with the value in w-space, *i.e.*, $G(\theta_1) = G(\theta_2)$. See Example 8.1.5 for an concrete example.

Example 8.1.5. *Let $D = 2d$ and $\theta = (\theta^+, \theta^-) \in \mathbb{R}^D$. Consider the following parametrization $G(\theta) = (\theta^+)^{\odot 2} - (\theta^-)^{\odot 2}$.*

To see why Equation (8.11) fails, it suffices to take $d = 1$ and consider $\theta_1 = (1, 0)$ and $\theta_2 = (\sqrt{2}, 1)$. We can see $G(\theta_1) = G(\theta_2) = 1$ but $4 = \partial G(\theta_1)\partial G(\theta_1)^\top \neq \partial G(\theta_2)\partial G(\theta_2)^\top = 12$.

The following theorem Theorem 8.1.6 shows that for the parametrization defined in Example 8.1.5, for every initialization θ_0 of certain form, there exsits a mirror map depending on the initialization θ_0 satisfies Equation (8.10). [3] Thus the equivalence between mirror descent and gradient descent holds for this parametrization.

Theorem 8.1.6 ((author?) [WGL+20]). *Under setting for Example 8.1.5, for any $\alpha > 0$ and $\theta_0 = (\alpha\mathbf{1}, -\alpha\mathbf{1})$[4], where $\mathbf{1} \in \mathbb{R}^d$ is the all-one vector, the gradient flow trajectory θ_t of $L \circ G$ (Equation (8.7)) is equivalent to the mirror flow trajectory w_t of $R_\alpha(w) \triangleq {}^{\alpha^2}\!/\!_4 \cdot \sum_{i=1}^n q(\frac{w[i]}{\alpha^2})$ (Equation (8.6)), where*

$$q(z) = 2 - \sqrt{4 + z^2} + z \cdot arcsinh\left(\frac{z}{2}\right). \tag{8.12}$$

The high-level proof idea here is that we can write $\partial G(\theta_t)\partial G(\theta_t)^\top$ as a function of $G(\theta_t)$ and θ_0, which in turn can be written as the inverse of the Hessian of mirror map function w.r.t. $G(\theta_t)$. The mirror map depends on the θ_0. A key observation here is a conservation law over time, Equation (8.13).

Proof of Theorem 8.1.6. First we notice that by chain rule,

$$d(\theta_t^+ \odot \theta_t^-)/dt = 0 \tag{8.13}$$

[3] The dependence of mirror map on the initialization is necessary. Otherwise, if Equation (8.10) holds for all different initializations, we can again apply the construction in Example 8.1.5, which yields a contradiction.

[4] This theorem can be generalized to any $\theta_0 \in \mathbb{R}^{2d}$ with a potentially more complicated mirror map

and thus has $\theta_t^+ \odot \theta_t^- \equiv \theta_0^+ \odot \theta_0^- = \alpha^2 \mathbf{1}$ is constant over time t. This further implies that

$$\begin{aligned}\partial G(\theta_t)\partial G(\theta_t)^\top =& 8 \cdot \frac{(\theta_t^+)^{\odot 2} + (\theta_t^-)^{\odot 2}}{2} && (8.14)\\ =& 8\left(\left(\frac{(\theta_t^+)^{\odot 2} - (\theta_t^-)^{\odot 2}}{2}\right)^{\odot 2} + \mathbf{1}\alpha^4\right)^{1/2} && (8.15)\\ =& 8\left(\left(\frac{G(\theta_t)}{2}\right)^{\odot 2} + \mathbf{1}\alpha^4\right)^{\odot 1/2} && (8.16)\\ =& 4\left((G(\theta_t))^{\odot 2} + 4 \cdot \mathbf{1}\alpha^4\right)^{\odot 1/2}. && (8.17)\end{aligned}$$

Finally we note that for any $i, j \in [d]$ and $w[i], w[j] \in \mathbb{R}$,

$$\frac{\partial R_\alpha(w)}{\partial w[i]} = \frac{\alpha^2 q(w[i]/\alpha^2)}{dw[i]} = \frac{1}{4}\text{arcsinh}\left(\frac{w[i]}{2\alpha^2}\right), \quad (8.18)$$

and that,

$$\begin{aligned}\frac{\partial^2 R_\alpha(w)}{\partial w[i]\partial w[j]} =& \frac{\partial^2 R_\alpha(w)}{\partial w[i]\partial w[i]}\mathbf{1}_{i=j} = \frac{1}{4}\frac{d\text{arcsinh}\left(\frac{w[i]}{2\alpha^2}\right)}{dw[i]}\mathbf{1}_{i=j} && (8.19)\\ =& \frac{1}{4\sqrt{w[i]^2 + 4\alpha^4}}\mathbf{1}_{i=j} && (8.20)\end{aligned}$$

This completes the proof because now by Equation (8.9), we have

$$\frac{dG(\theta_t)}{dt} = -(\nabla^2 R_\alpha(G(\theta_t)))^{-1}\nabla L(G(\theta_t)), \quad (8.21)$$

showing $G(\theta_t)$ is equal to w_t, as they are both the unique solution of Equation (8.6). □

Combinig the above theorem with Theorem 8.1.2, we have the following theorem:

Theorem 8.1.7 (Theorem 1 in **(author?)** [WGL+20]). *Under the setting of Theorem 8.1.6, additionaly assume that*

1. $L(w) = \frac{1}{n}(w^\top x^{(i)} - y^{(i)})^2$ *where* $(x^{(i)}, y^{(i)})_{i=1}^n$ *are training datasets;*
2. *the reparametrized gradient flow Equation (8.7) with initial point* $\theta_0 = (\alpha\mathbf{1}, -\alpha\mathbf{1})$ *converges to a 0-loss solution* θ_∞.

Then $w_\infty = G(\theta_\infty)$ *satisfies that*

$$\begin{aligned} w_\infty =& \arg\min_{w\in\mathbb{R}^d} R_\alpha(w) && (8.22)\\ & s.t. L(w) = 0. && (8.23)\end{aligned}$$

[zhiyuan:add some basic comments about the mirror map?]

8.1.4 Equivalence Between Commuting Parametrization and Mirror Descent

In this subsection we are interested in the following two questions:

1. For what mirror map R, there exists a parametrization G such that the equivalence holds?

2. For what parametrization G, there exists a mirror map R such that the equivalence holds?

The proof for Theorem 8.1.6 does not give us much insight on how to decide whether such mirror map R exists for a given parametrization G. It relies on the conservation law Equation (8.13), which seems to be a special property of the parametrization G in Example 8.1.5.

In this subsection we will introduce a more general framework towards attacking the above two questions from **(author?)** [LWLA22]. It turns out the answer to the first question is always yes. And the answer to the second question is positive when certain conditions for parameterization G are met, *e.g.*, the following *commuting* condition:

Definition 8.1.8 (Commuting parametrization)**.** *We say a parametrization $G : \mathbb{R}^D \to \mathbb{R}^d$ is* commuting *iff $[\nabla G_i, \nabla G_j](\theta) = \nabla^2 G_i(\theta)\nabla G_j(\theta) - \nabla^2 G_j(\theta)\nabla G_i(\theta) = 0$ for all $\theta \in \mathbb{R}^D, i, j \in [D]$.*[5]

One can easily check that parametrizations in both Examples 8.1.3 to 8.1.5 are all commuting parametrizations. **(author?)** [LWLA22] also shows that a slightly relaxed notion of commuting parametrizations is necessary to induce equivalent geometry with some mirror map.

[zhiyuan:TBD]

8.2 Matrix factorization

≪Suriya notes: I would like to include this section here but can also move to a separate chapter. Ideally, summarize our 2017 paper, Tengyu's 2018 paper and Nadav's 2019 paper. May be we can discuss this after Nadav's lecture?≫

8.3 Linear Models in Classification

We now turn to studing classification problems with logistic or cross-entropy type losses. We focus on binary classification problems where $y^{(i)} \in \{-1, 1\}$. Many continuous surrogate of the 0-1 loss inlcuding logistic, cross-entropy, and exponential loss are examples of strictly monotone loss functions ℓ where the behavior of the implicit bias is fundamentally different, and as are the situations when the implicit bias can be characterized.

[5] Here $[\cdot, \cdot]$ denotes the Lie bracket. Formally, for any two vector fields X, Y, the Lie bracket $[X, Y]$ is defined as $[X, Y](\theta) = \partial X \cdot Y - \partial Y \cdot X$ for any θ.

We look at classification models that fit the training data $\{x^{(i)}, y^{(i)}\}_i$ with linear decision boundaries $f(x) = w^\top x$ with decision rule given by $\widehat{y}(x) = \text{sign}(f(x))$. In many instances of the proofs, we also assume without loss of generality that $y^{(i)} = 1$ for all i, since for linear models, the sign of $y^{(i)}$ can equivalently be absorbed into $x^{(i)}$. We again look at unregularized empirical risk minimization objective of the form in eq. (8.1), but now with strictly monotone losses. When the training data $\{x^{(i)}, y^{(i)}\}_n$ is not linearly separable, the empirical objective $L(w)$ can have a finite global minimum. However, if the dataset is linearly separable, i.e., $\exists w : \forall i, y^{(i)} w^\top y^{(i)} > 0$, the empirical loss $L(w)$ is again ill-posed, and moreover $L(w)$ does not have any finite minimizer, i.e, $L(w) \to 0$ only as $\|w\| \to \infty$. Thus, for any sequence $\{w_t\}_{t=0}^\infty$, if $L(w_t) \to 0$, then w_t necessarily diverges to infinity rather than converge, and hence we cannot talk about $\lim_{t\to\infty} w_t$. Instead, we look at the limit direction $\bar{w}_\infty = \lim\limits_{t\to\infty} \frac{w_t}{\|w_t\|}$ whenever the limit exists. We refer to existence of this limit as convergence in direction. Note that, the limit direction fully specifies the decision rule of the classifier that we care about.

In the remainder of the chapter, we focus on the following exponential loss $\ell(u, y) = \exp(-uy)$. However, our asymptotic results can be extended to loss functions with tight exponential tails, including logistic and sigmoid losses, along the lines of **(author?)** [SHS17] and **(author?)** [Tel13].

$$L(w) = \sum_{i=1}^{n} \exp(-y^{(i)} w^\top x^{(i)}). \tag{8.24}$$

8.3.1 Gradient Descent

(author?) [SHS17] showed that for almost all linearly separable datasets, gradient descent with *any initialization and any bounded step-size* converges in direction to maximum margin separator with unit ℓ_2 norm, i.e., the hard margin support vector machine classifier.

This characterization of the implicit bias is independent of both the step-size as well as the initialization. We already see a fundamentally difference from the implicit bias of gradient descent for losses with a unique finite root (Section **??**) where the characterization depended on the initialization. The above result is rigorously proved as part of a more general result in Theorem 8.3.2. Below is a simpler statement and with a heuristic proof sketch intended to convey the intuition for such results.

Theorem 8.3.1. *For almost all dataset which is linearly separable, consider gradient descent updates with any initialization w_0 and any step size that minimizes the exponential loss in eq. (8.24), i.e., $L(w_t) \to 0$. The gradient* *descnet iterates then converge in direction to the ℓ_2 max-margin vector, i.e.,* $\lim_{t\to\infty} \frac{w_t}{\|w_t\|_2} = \frac{\widehat{w}}{\|\widehat{w}\|}$*, where*

$$\widehat{w} = \underset{w}{\operatorname{argmin}} \|w\|^2 \text{ s.t. } \forall i,\ w^\top x^{(i)} y^{(i)} \geq 1. \tag{8.25}$$

Without loss of generality assume that $\forall i, y^{(i)} = 1$ as the sign for linear models can be absobed into $x^{(i)}$.

Proof Sketch We first understand intuitively why an exponential tail of the loss entail asymptotic convergence to the max margin vector: examine the asymptotic regime of gradient descent in when the exponential loss is minimized, as we argued earlier, this required that $\forall i:\ w^\top x^{(i)} \to \infty$. Suppose $w_t / \|w_t\|_2$ converges to some limit w_∞, so we can write $w_t = g(t) w_\infty + \rho(t)$ such that $g(t) \to \infty$, $\forall i,\ w_\infty^\top x^{(i)} > 0$, and $\lim_{t\to\infty} \rho(t)/g(t) = 0$. The gradients at w_t are given by:

$$\begin{aligned} -\nabla \mathcal{L}(w) &= \sum_{i=1}^{n} \exp\left(-w^\top x^{(i)}\right) x^{(i)} \\ &= \sum_{i=1}^{n} \exp\left(-g(t) \mathrm{w}_\infty^\top x^{(i)}\right) \exp\left(-\rho(t)^\top x^{(i)}\right) \mathbf{x}_n\,. \end{aligned} \tag{8.26}$$

As $g(t) \to \infty$ and the exponents become more negative, only those samples with the largest (*i.e.*, least negative) exponents will contribute to the gradient. These are precisely the samples with the smallest margin $\operatorname{argmin}_i w_\infty^\top x^{(i)}$, aka the "support vectors". The accumulation of negative gradient, and hence w_t, would then asymptotically be dominated by a non-negative linear combination of support vectors. These are precisely the KKT conditions for the SVM problem (eq. 8.25). Making these intuitions rigorous constitutes the bulk of the proof in **(author?)** [SHS17], which uses a proof technique very different from that in the following section (Section 8.3.2).

8.3.2 Steepest Descent

. Recall that gradient descent is a special case of steepest descent (SD) w.r.t a generic norm $\|\cdot\|$ with updates given by eq. (8.5). The optimality condition of Δw_t in eq. (8.5) requires

$$\langle \Delta w_t, -\nabla L(w_t) \rangle = \|\Delta w_t\|^2 = \|\nabla L(w_t)\|_\star^2., \tag{8.27}$$

where $\|x\|_\star = \sup_{\|y\| \leq 1} x^\top y$ is the dual norm of $\|\cdot\|$. Examples of steepest descent include gradient descent, which is steepest descent w.r.t ℓ_2 norm and greedy coordinate descent (Gauss-Southwell selection rule), which is steepest descent w.r.t ℓ_1 norm. In general, the update Δw_t in eq. (8.5) is not uniquely defined and there could be multiple direction Δw_t that minimize eq. (8.5). In such cases, any minimizer of eq. (8.5) is a valid steepest descent update and satisfies eq. (8.27).

In the preliminary result in Theorem 8.3.1, we proved the limit direction of gradient flow on the exponential loss is the ℓ_2 max-margin solution. In the following theorem, we show the natural extension of this to all steepest descent algorithms.

Theorem 8.3.2. *For any separable dataset $\{x_i, y_i\}_{i=1}^n$ and any norm $\|\cdot\|$, consider the steepest descent updates from eq. (8.27) for minimizing $L(w)$ in eq. (8.24) with the exponential loss $\ell(u, y) = \exp(-uy)$. For all initializations w_0, and all bounded step-sizes satisfying $\eta_t \leq \min\{\eta_+, \frac{1}{B^2 L(w_t)}\}$, where $B := \max_n \|x_n\|_\star$ and $\eta_+ < \infty$ is any finite number, the iterates w_t satisfy the following,*

$$\lim_{t\to\infty} \min_n \frac{y_i \langle w_t, y_i \rangle}{\|w_t\|} = \max_{w:\|w\|\leq 1} \min_n y_i \langle w, x_i \rangle =: \gamma.$$

In particular, if there is a unique maximum-$\|\cdot\|$ margin solution $w^* = \arg\max_{\|w\|\leq 1} \min_i y_i \langle w, x_i \rangle$, then the limit direction satisfies $\lim_{t\to\infty} \frac{w_t}{\|w_t\|} = w^*$.

A special case of Theorem 8.3.2 is for steepest descent w.r.t. the ℓ_1 norm, which as we already saw corresponds to greedy coordinate descent. More specifically, coordinate descent on the exponential loss with exact line search is equivalent to AdaBoost [SF12], where each coordinate represents the output of one "weak learner". Indeed, initially mysterious generalization properties of boosting have been understood in terms of implicit ℓ_1 regularization [SF12], and later on AdaBoost with small enough step-size was shown to converge in direction precisely to the maximum ℓ_1 margin solution [ZY+05, SSS10, Tel13], just as guaranteed by Theorem 8.3.2. In fact, **(author?)** [Tel13] generalized the result to a richer variety of exponential tailed loss functions including logistic loss, and a broad class of non-constant step-size rules. Interestingly, coordinate descent with exact line search (AdaBoost) can result in infinite step-sizes, leading the iterates to converge in a different direction that is not a max-ℓ_1-margin direction [RDS04], hence the bounded step-sizes rule in Theorem 8.3.2.

Theorem 8.3.2 is a generalization of the result of **(author?)** to steepest descent with respect to other norms, and our proof follows the same strategy as **(author?)**. We first prove a generalization of the duality result of **(author?)** [SSS10]: if there is a unit norm linear separator that achieves margin γ, then $\|\nabla L(w)\|_\star \geq \gamma L(w)$ for all w. By using this lower bound on the dual norm of the gradient, we are able to show that the loss decreases faster than the increase in the norm of the iterates, establishing convergence in a margin maximizing direction.

In the rest of this section, we discuss the proof of Theorem 8.3.2. The proof is divided into three steps:

1. Gradient domination condition: For all norms and any w, $\|\nabla L(w)\|_\star \geq \gamma L(w)$

2. Optimization properties of steepest descent such as decrease of loss function and convergence of the gradient in dual norm to zero.

3. Establishing sufficiently fast convergence of $L(w_t)$ relative to the growth of $\|w_t\|$ to prove the Theorem.

Proposition 8.3.3. *Gradient domination condition (Lemma 10 of [GLSS18a]) Let $\gamma = \max_{\|w\|\leq 1} \min_i y_i x_i^\top w$. For all w,*

$$\|\nabla L(w)\|_\star \geq \gamma L(w).$$

Next, we establish some optimization properties of the steepest descent algorithm including convergence of gradient norms and loss value.

Proposition 8.3.4. *(Lemma 11 and 12 of* **(author?)** *[GLSS18a]) Consider the steepest descent iterates w_t on (8.24) with stepsize $\eta \leq \frac{1}{B^2 L(w_0)}$, where $B = \max_i \|x_i\|_\star$. The following holds:*

1. *$L(w_{t+1}) \leq L(w_t)$.*

2. *$\sum_{t=0}^{\infty} \|\nabla L(w_t)\|^2 < \infty$ and hence $\|\nabla L(w_t)\|_\star \to 0$.*

3. *$L(w_t) \to 0$ and hence $w_t^\top x_i \to \infty$.*

4. *$\sum_{t=0}^{\infty} \|\nabla L(w_t)\|_\star = \infty$.*

Given these two Propositions, the proof proceeds in two steps. We first establish that the loss converges to zero sufficiently quickly to lower bound the unnormalized margin $\min_i w_t^\top x_i$. Next, we upper bound $\|w_t\|$. By dividing the lower bound in the first step by the upper bound in the second step, we can lower bound the normalized margin, which will complete the proof.

Proof of Theorem 8.3.2. **Step 1: Lower bound the unnormalized margin.** First, we establish the loss converges sufficiently quickly. Define

$\gamma_t = \|\nabla L(w_t)\|_\star$. From Taylor's theorem,

$$\begin{aligned}
&L(w_{t+1}) \leq \\
&L(w_t) + \eta_t \left\langle \nabla L(w_t), \Delta w_t \right\rangle + \sup_{\beta \in (0,1)} \frac{\eta_t^2}{2} \Delta w_t^\top \nabla^2 L\left(w_t + \beta \eta_t \Delta w_t\right) \Delta w_t \\
&\overset{(a)}{\leq} L(w_t) - \eta_t \left\|\nabla L(w_t)\right\|_\star^2 + \frac{\eta_t^2 B^2}{2} \sup_{\beta \in (0,1)} L\left(w_t + \beta \eta_t \Delta w_t\right) \left\|\Delta w_t\right\|^2 \\
&\overset{(b)}{\leq} L(w_t) - \eta_t \left\|\nabla L(w_t)\right\|_\star^2 + \frac{\eta_t^2 B^2}{2} L(w_t) \left\|\Delta w_t\right\|^2
\end{aligned} \tag{8.28}$$

where (a) uses $v^\top \nabla^2 L(w) v \leq L(w) B^2 \|v\|^2$ and (b) uses Proposition 8.3.4 part 1 and convexity to show $\sup_{\beta \in (0,1)} L\left(w_t + \beta \eta_t \Delta w_t\right) \leq L(w_t)$.

From eq . 8.28, using $\gamma_t = \|\nabla L(w_t)\|_\star = \|\Delta w_t\|$, we have that

$$\begin{aligned}
L(w_{t+1}) &\leq L(w_t) - \eta \gamma_t^2 + \frac{\eta^2 B^2 L(w_t) \gamma_t^2}{2} \\
&= L(w_t) \left[1 - \frac{\eta \gamma_t^2}{L(w_t)} + \frac{\eta^2 B^2 \gamma_t^2}{2}\right] \\
&\overset{(a)}{\leq} L(w_t) \exp\left(-\frac{\eta \gamma_t^2}{L(w_t)} + \frac{\eta^2 B^2 \gamma_t^2}{2}\right) \\
&\overset{(b)}{\leq} L(w_0) \exp\left(-\sum_{u \leq t} \frac{\eta_u \gamma_u^2}{L(w_u)} + \sum_{u \leq t} \frac{\eta^2 B^2 \gamma_u^2}{2}\right),
\end{aligned} \tag{8.29}$$

where we get (a) by using $(1+x) \leq \exp(x)$, and (b) by recursing the argument.

Next, we lower bound the unnormalized margin. From eq. (8.29), we have,

$$\begin{aligned}
&\max_{n \in [N]} \exp(-\left\langle w_{t+1}, x_n \right\rangle) \leq L(w_{(t+1)}) \\
&\qquad\qquad \leq L(w_0) \exp\left(-\sum_{u \leq t} \frac{\eta \gamma_u^2}{L(w_u)} + \sum_{u \leq t} \frac{\eta^2 B^2 \gamma_u^2}{2}\right)
\end{aligned} \tag{8.30}$$

By applying $-\log$,

$$\min_{n \in [N]} \left\langle w_{t+1}, x_n \right\rangle \geq \sum_{u \leq t} \frac{\eta \gamma_u^2}{L(w_u)} - \sum_{u \leq t} \frac{\eta^2 B^2 \gamma_u^2}{2} - \log L(w_0). \tag{8.31}$$

Step 2: Upper bound $\|w_{t+1}\|$. Using $\|\Delta w_u\| = \|\nabla L(w_u)\|_\star = \gamma_u$, we have,

$$\|w_{t+1}\| \leq \|w_0\| + \sum_{u \leq t} \eta \left\|\Delta w_u\right\| \leq \|w_0\| + \sum_{u \leq t} \eta \gamma_u. \tag{8.32}$$

To complete the proof, we simply combine Equations (8.31) and (8.32) to lower bound the normalized margin.

$$\begin{aligned}\frac{\langle w_{t+1}, x_n\rangle}{\|w_{t+1}\|} \geq & \frac{\sum_{u\leq t}\frac{\eta\gamma_u^2}{L(w_u)}}{\sum_{u\leq t}\eta\gamma_u + \|w_0\|} \quad - \left(\frac{\sum_{u\leq t}\frac{\eta^2B^2\gamma_u^2}{2}+\log L(w_0)}{\|w_{t+1}\|}\right). \\ := & (I) \qquad\qquad\qquad +(II). \end{aligned} \tag{8.33}$$

For term (I), from Proposition 8.3.3, we have $\gamma_u = \|\nabla L(w_u)\|_\star \geq \gamma L(w_u)$. Hence the numerator is lower bounded $\sum_{u\leq t}\frac{\eta\gamma_u^2}{L(w_u)} \geq \gamma\sum_{u\leq t}\eta\gamma_u$. We have

$$\frac{\sum_{u\leq t}\frac{\eta\gamma_u^2}{L(w_u)}}{\sum_{u\leq t}\eta\gamma_u + \|w_0\|} \geq \gamma\frac{\sum_{u\leq t}\eta\gamma_u}{\sum_{u\leq t}\eta\gamma_u + \|w_0\|} \to \gamma, \tag{8.34}$$

using $\sum_{u\leq t}\eta\gamma_u \to \infty$ and $\|w_0\| < \infty$ from Proposition 8.3.4.

For term (II), $\log L(w_0) < \infty$ and $\sum_{u\leq t}\frac{\eta^2B^2\gamma_u^2}{2} < \infty$ using Proposition 8.3.3. Thus $(II) \to 0$.

Using the above in Equation (8.33), we obtain

$$\lim_{t\to\infty}\frac{w_{t+1}^\top x_i}{\|w_{t+1}\|} \geq \gamma := \max_{\|w\|\leq 1}\min_i \frac{w^\top x_i}{\|w\|}.$$

□

8.4 Homogeneous Models with Exponential Tailed Loss

≪Suriya notes: Jason: I think we should give Kaifengs’ proof here. Its more general and concurrent work.≫ In this section, we consider the asymptotic behavior of gradient descent when the prediction function is homogeneous in the parameters. Consider the loss

$$L(w) = \sum_{i=1}^n \exp(-y_i f_i(w)), \tag{8.35}$$

where $f_i(cw) = c^\alpha f_i(w)$ is α-homogeneous. Typically, $f_i(w)$ is the output of the prediction function such as a deep network. Similar to the linear case in Section 5.5.1, there is a related maximum margin problem. Define the optimal margin as $\gamma = \max_{\|w\|_2=1}\min_i y_i f_i(w)$. The associated non-linear margin maximization is given by the following non-convex constrained optimization:

$$\min \|w\|^2 \text{ st } y_i f_i(w) \geq \gamma. \tag{Max-Margin}$$

Analogous to Section 5.5.1, we expect that gradient descent on Equation (8.35) converges to the optimum of the Max-Margin problem (Max-Margin). However, the max-margin problem itself is a constrained non-convex problem, so we cannot expect to attain a global optimum. Instead, we show that gradient descent iterates converge to first-order stationary points of the max-margin problem.

Definition 8.4.1 (First-order Stationary Point)**.** *The first-order optimality conditions of Max-Margin are:*

1. $\forall i$, $y_i f_i(w) \geq \gamma$

2. *There exists Lagrange multipliers* $\lambda \in \mathbb{R}_+^N$ *such that* $w = \sum_n \lambda_n \nabla f_n(w)$ *and* $\lambda_n = 0$ *for* $n \notin S_m(w) := \{i : y_i f_i(w) = \gamma\}$, *where* $S_m(w)$ *is the set of support vectors .*

We denote by $\mathcal{W}^\star$ *the set of first-order stationary points.*

Let w_t be the iterates of gradient flow (gradient descent with step-size tending to zero). Define $\ell_{it} = \exp(-f_i(w_t))$ and $\boldsymbol{\ell}_t$ be the vector with entries $\ell_i(t)$. The following two assumptions assume that the limiting direction $\frac{w_t}{\|w_t\|}$ exist and the limiting direction of the losses $\frac{\boldsymbol{\ell}_t}{\|\boldsymbol{\ell}_t\|_1}$ exist. Such assumptions are natural in the context of max-margin problems, since we want to argue that w_t converges to a max-margin direction, and also the losses $\boldsymbol{\ell}_t / \|\boldsymbol{\ell}_t\|_1$ converges to an indicator vector of the support vectors. We will directly assume these limits exist, though this is proved in [6].

[6]

Assumption 8.4.2 (Smoothness)**.** *We assume* $f_i(w)$ *is a* C^2 *function.*

Assumption 8.4.3 (Asymptotic Formulas)**.** *Assume that* $L(w_t) \to 0$, *that is we converge to a global minimizer. Further assume that* $\lim_{t\to\infty} \frac{w_t}{\|w_t\|_2}$ *and* $\lim_{t\to\infty} \frac{\boldsymbol{\ell}_t}{\|\boldsymbol{\ell}_t\|_1}$ *exist. Equivalently,*

$$\ell_{nt} = h_t a_n + h_t \epsilon_{nt} \tag{8.36}$$

$$w_t = g_t \bar{w} + g_t \delta_t, \tag{8.37}$$

with $\|\boldsymbol{a}\|_1 = 1$, $\|\bar{w}\|_2 = 1$, $\lim_{t\to\infty} h(t) = 0$, $\lim_{t\to\infty} \epsilon_{nt} = 0$, *and* $\lim_{t\to\infty} \delta_t t = 0$.

Assumption 8.4.4 (Linear Independence Constraint Qualification)**.** *Let* w *be a unit vector. LICQ holds at* w *if the vectors* $\{\nabla f_i(w)\}_{i \in S_m(w)}$ *are linearly independent.*

Constraint qualification allow the first-order optimality conditions of Definition 8.4.1 to be a necessary condition for optimality. Without constraint qualifications, event he global optimum may not satisfy the optimality conditions.

For example in linear SVM, LICQ is ensured if the support vectors x_i are linearly independent then LICQ holds. For data sampled from an absolutely continuous distribution, the linear SVM solution will always have linearly independent support vectors.

Theorem 8.4.5. *Define* $\bar{w} = \lim_{t\to\infty} \frac{w_t}{\|w_t\|}$. *Under Assumptions 8.4.2, 8.4.3, and 8.4.4,* $\bar{w} \in \mathcal{W}$ *is a first-order stationary point of* (Max-Margin).

Proof. Define $S = \{i : f_i(\bar{w}) = \gamma\}$, where γ is the optimal margin attainable by a unit norm w.

Lemma 8.4.6. *Under the setting of Theorem 8.4.5,*

$$\nabla f_i(w_t) = \nabla f_i(g_t\bar{w}) + O(Bg_t^{\alpha-1}\|\delta_t\|). \tag{8.38}$$

For $i \in S$ *, the second term is asymptotically negligible as a function of* t*,*

$$\nabla f_i(w_t) = \nabla f_i(g_t\bar{w}) + o(\nabla f_i(g_t\bar{w})).$$

Lemma 8.4.7. *Under the conditions of Theorem 8.4.5,* $a_i = 0$ *for* $i \notin S$*.*

From the gradient flow dynamics,

$$\begin{aligned}\dot{w}(t) &= \sum_i \exp(-f_i(w_t))\nabla f_i(w_t) \\ &= \sum_i (h_t a_i + h_t\epsilon_{it})(\nabla f_i(g_t\bar{w}) + \Delta_{it},\end{aligned}$$

where $\Delta_i(t) = \int_{s=0}^{s=1} \nabla^2 f_i(g_t\bar{w} + sg_t\delta_t)g_t\delta_t ds$. By expanding and using $a_i = 0$ for $n \notin S$ (Lemma 8.4.7) ,

$$\begin{aligned}\dot{w}_t = &\underbrace{\sum_{i\in S} h_t a_i \nabla f_i(g_t w)}_{I} \\ &+ \underbrace{h_t \sum_{i\in S} a_i\Delta_{it}}_{II} + \underbrace{h_t \sum_i \epsilon_{it}\nabla f_i(g_t\bar{w})}_{III} + \underbrace{\sum_i h_t\epsilon_{it}\Delta_{it}}_{IV}\end{aligned}$$

Via Assumption 8.4.4, term $I = \Omega(g_t^{\alpha-1}h_t)$ and from Lemma 8.4.6 , $II = o(I)$. Using these, the first term I is the largest and so after normalizing,

$$\frac{\dot{w}_t}{\|\dot{w}_t\|} = \sum_{i\in S} a_i \nabla f_i(g_t\bar{w}) + o(1). \tag{8.39}$$

Since $\lim_t \frac{w_t}{\|w_t\|} = \lim_t \frac{\dot{w}_t}{\|w_t\|}$ [GLSS18a], then

$$\lim_{t\to\infty} \frac{w_t}{\|w_t\|} = \sum_{i\in S} \nabla f_i(g_t\bar{w}). \tag{8.40}$$

Thus we have shown w satisfies the first-order optimality condition of Definition 8.4.1. □

8.5 Induced bias in function space

≪Suriya notes: Jason: can you introduce the idea of induced biases and give special results for linear convnets, any relevant results from yours+tengyu's margin paper, and infinite width 2 layer ReLU network?≫

9
Ultra-wide Neural Networks and Neural Tangent Kernels

This chapter concerns a model that seems ridiculous at first sight: one with infinitely many nodes at inner layers. To understand why it is actually interesting, let's recall the mysteries we're trying to understand.

Training a neural network is a non-convex optimization problem, and in the worst case, it is NP-hard [BR89]. On the other hand, empirically, simple gradient algorithms like stochastic gradient descent can often achieve zero training loss, i.e., the simple algorithm can find a neural network that fits all training data. Furthermore, one can still observe this phenomenon for nonsensical data, when the original labels are replaced with random labels [ZBH+16b].

Key role of overparametrization. The fact that networks can fit nonsensical data perfectly is not surprising because the nets are very over-parameterized. For example, Wide ResNet when trained on ImageNet has 100x more parameters than the number of training datapoints. Recall from Chapter 3 that under such conditions even linear regression (solved via gradient descent) can perfectly fit training data. But this does not explain real-life neural nets because we still need to prove that: (a) the low loss can be attained by a *gradient-based training* starting from a *random initialization* (b) that the trained net has good generalization when trained on proper data. Many traditional generalization bounds yield vacuous guarantees, as mentioned in Chapter 5.

Teacher/Student Nets. A difficulty that arises while addressing this research agenda is that clearly at some point the theory should take properties of the data into account, and real-life data (e.g., images) have no good description. One route taken by researchers is to assume that the labels for training data were computed via some ground truth net sometimes refered to as *teacher net*. Thus the net being trained is thought of as a *student net* and the goal of good generalization is to be able to produce labels broadly in agreement with the teacher net.

Given the importance of overparametrization in real life, it is natural to allow the student net to be much deeper or wider than the teacher net. [1]

Infinite nets and NTKs: Now we explain the model of infinite neural networks. The idea is to let the width in the inner layers be very large, essentially going to infinity. For example, imagine a standard net like AlexNet being allowed to expand its fully connected layers to have unlimited width and the convolutional layers have convolutional filters with unlimited number of channels. This hugely inflated version of AlexNet architecture still takes the same input as before but its training and generalization behavior could potentially change a lot from the usual version. Researchers studied these architectures and quickly realized that at least one way of initialization/training leads to the net turning into a kernel classifer, called *Neural Tangent Kernel* (NTK) [2]. This chapter is an introduction to infinitely wide nets and NTKs. We'll see that behavior of NTKs can be computed via efficient algorithms for computing the kernel inner product for NTK. NTKs do exhibit good optimization (i.e. convergence to low training loss) and reasonable generalization behavior but are not as good as their finite counterparts. For example the NTK corresponding to AlexNet generalizes reasonably OK on image data but with far worse accuracy than AlexNet. We will discuss some practical uses of NTK for small-data tasks.

9.1 *Evolution equation for net parameters*

This section derives evolution of nets during training under least squares loss. It applies to any net, and the simple expression will play a key role in NTK theory.

We denote by $f(w,x) \in \mathbb{R}$ the output of a neural network where $w \in \mathbb{R}^N$ is all the parameters in the network and $x \in \mathbb{R}^d$ is the input. Given a training dataset $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$, consider training the neural network by minimizing the squared loss over training data:

$$\ell(w) = \frac{1}{2}\sum_{i=1}^{n} \left(f(w,x_i) - y_i\right)^2 .$$

For simplicity, in this chapter, we study gradient flow, a.k.a., gradient decent with infinitesimally small learning rate. In this case, the dy-

[1] Indeed in experiments one finds that if synthetically labeled data is generated by passing inputs from a distribution through a teacher net, then teaching a new net from scratch to mimic the teacher is much easier in practice if the new net is allowed to be significantly bigger.

[2] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018

namics can be described by an ordinary differential equation (ODE):

$$\frac{dw(t)}{dt} = -\nabla \ell(w(t)).$$

Using this description of the dynamics of the parameters, the next lemma describes the dynamics of the predictions on training data points.

Lemma 9.1.1. *Let $u(t) = (f(w(t), x_i))_{i\in[n]} \in \mathbb{R}^n$ be the network outputs on all x_i's at time t, and $y = (y_i)_{i\in[n]}$ be the labels. Then $u(t)$ follows the following evolution, where $H(t)$ is an $n \times n$ positive semidefinite matrix whose (i, j)-th entry is $\left\langle \frac{\partial f(w(t),x_i)}{\partial w}, \frac{\partial f(w(t),x_j)}{\partial w} \right\rangle$:*

$$\frac{du(t)}{dt} = -H(t) \cdot (u(t) - y). \tag{9.1}$$

Proof of Lemma 9.1.1. The parameters w evolve according to the differential equation

$$\frac{dw(t)}{dt} = -\nabla \ell(w(t)) = -\sum_{i=1}^{n} (f(w(t), x_i) - y_i) \frac{\partial f(w(t), x_i)}{\partial w}, \tag{9.2}$$

where $t \geq 0$ is a continuous time index. Under Equation (9.2), the evolution of the network output $f(w(t), x_i)$ can be written as

$$\frac{df(w(t), x_i)}{dt} = -\sum_{j=1}^{n} (f(w(t), x_j) - y_j) \left\langle \frac{\partial f(w(t), x_i)}{\partial w}, \frac{\partial f(w(t), x_j)}{\partial w} \right\rangle. \tag{9.3}$$

Since $u(t) = (f(w(t), x_i))_{i\in[n]} \in \mathbb{R}^n$ is the network outputs on all x_i's at time t, and $y = (y_i)_{i\in[n]}$ is the desired outputs, Equation (9.3) can be written more compactly as

$$\frac{du(t)}{dt} = -H(t) \cdot (u(t) - y), \tag{9.4}$$

where $H(t) \in \mathbb{R}^{n\times n}$ is a kernel matrix defined as $[H(t)]_{i,j} = \left\langle \frac{\partial f(w(t),x_i)}{\partial w}, \frac{\partial f(w(t),x_j)}{\partial w} \right\rangle$ $(\forall i, j \in [n])$. □

9.1.1 *Behavior in the infinite limit*

Recall that we're interested in the limit of deep net training when the training set is fixed, the width goes to infinity, and for a suitable scale of initialization (which depends on the width). Under fairly general conditions it can be shown that the matrix $H(t)$ remains rough *constant* during training i.e., roughly equal to $H(0)$. Furthermore, the matrix $H(0)$, whose definition involves random weights used at initialization, converges in probability to the Gram Matrix H^* of the training dataset (see Chapter 3) with respect to certain kernel, called the *Neural Tangent Kernel*. Then Equation (9.1) becomes

$$\frac{du(t)}{dt} = -H^* \cdot (u(t) - y). \tag{9.5}$$

In other words, least squares kernel regression as described in Chapter 3, but with infinitesimally small learning rate. Recall that the final classifier is described as

$$f^*(x) = (k(x, x_1), \dots, k(x, x_n)) \cdot (H^*)^{-1} y. \tag{9.6}$$

9.2 NTK: Simple 2-layer example

In this section, we develop the theory in context of a simple two-layer neural network of the following form:

$$f(a, W, x) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma\left(w_r^\top x\right) \tag{9.7}$$

where $\sigma(\cdot)$ is the activation function. Here we assume $|\dot{\sigma}(z)|$ and $|\ddot{\sigma}(z)|$ are bounded by 1 for all $z \in \mathbb{R}$ and For example, soft-plus activation function, $\sigma(z) = \log(1 + \exp(z))$, satisfies this assumption. [3] We also assume input x is a unit vector, i.e., $\|x\|_2 = 1$. The scaling $1/\sqrt{m}$ will play an important role in proving $H(t)$ stays close to the Gram Matrix H^* for a fixed kernel. Throughout the section, to measure the closeness of two matrices A and B, we use the operator norm $\|\cdot\|_2$. We will use the fact that if corresponding entries are close then so are their spectral properties A, B. This follows from the fact that the sum of squared differences across coordinates, $\|A - B\|_F^2$, is also an upper bound on $\|A - B\|_2^2$.

We use random initialization $w_r(0) \sim N(0, I)$ and $a_r \sim \text{Unif}[\{-1, 1\}]$. For simplicity, we will only optimize the first layer, i.e., $W = [w_1, \dots, w_m]$. Note this is still a non-convex optimization problem.

We can first calculate $H(0)$ and show as $m \to \infty$, $H(0)$ converges to a fixed matrix H^*. Note $\frac{\partial f(a,W,x_i)}{\partial w_r} = \frac{1}{\sqrt{m}} a_r x_i \sigma'\left(w_r^\top x_i\right)$. Therefore, each entry of $H(0)$ admits the formula

$$\begin{aligned}
[H(0)]_{ij} &= \sum_{r=1}^{m} \left\langle \frac{\partial f(a, W(0), x_i)}{\partial w_r(0)}, \frac{\partial f(a, W(0), x_j)}{\partial w_r(0)} \right\rangle \\
&= \sum_{r=1}^{m} \left\langle \frac{1}{\sqrt{m}} a_r x_i \dot{\sigma}\left(w_r(0)^\top x_i\right), \frac{1}{\sqrt{m}} a_r x_j \sigma'\left(w_r(0)^\top x_i\right) \right\rangle \\
&= x_i^\top x_j \cdot \frac{\sum_{r=1}^{m} \sigma'\left(w_r(0)^\top x_i\right) \sigma'\left(w_r(0)^\top x_j\right)}{m}
\end{aligned}$$

Here the last step we used $a_r^2 = 1$ for all $r = 1, \dots, m$ because we initialize $a_r \sim \text{Unif}[\{-1, 1\}]$. Recall every $w_r(0)$ is i.i.d. sampled from

[3] Note rectified linear unit (ReLU) activation function does not satisfy this assumption. However, one can use a specialized analysis of ReLU to show $H(t) \approx H^*$ [DZPS18].

a standard Gaussian distribution. Therefore, one can view $[H(0)]_{ij}$ *as the average of m i.i.d. random variables.* If m is large, then by the law of large number, we know this average is close to the expectation of the random variable. Here the expectation is the NTK evaluated on x_i and x_j:

$$H^*_{ij} \triangleq x_i^\top x_j \cdot \mathop{\mathbb{E}}_{w \sim N(0,I)} \left[\sigma'\left(w^\top x_i\right) \sigma'\left(w^\top x_j\right) \right]$$

Problem 9.2.1. *If the activation σ is ReLU then (noting that it is differentiable everywhere except at one point) then show that*

$$H^*_{ij} = \mathbb{E}_{w \sim \mathcal{N}(0,I)} \left[\dot{\sigma} w^\top x \dot{\sigma} w^\top x' \right] = \frac{\pi - \arccos\left(\frac{x^\top x'}{\|x\|_2 \|x'\|_2}\right)}{2\pi}. \tag{9.8}$$

Using Hoeffding inequality and the union bound, one can easily obtain the following bound characterizing m and the closeness of $H(0)$ and H^*.

Lemma 9.2.2 (Perturbation on the Initialization, [DZPS19, SY19])**.** *Fix some $\epsilon > 0$. If $m = \Omega\left(\epsilon^{-2} n^2 \log\left(n/\delta\right)\right)$, then with probability at least $1 - \delta$ over $w_1(0), \ldots, w_m(0)$, we have*

$$\|H(0) - H^*\|_2 \leq \epsilon.$$

Proof of Lemma 9.2.2. We first fixed an entry (i, j). Note

$$\left| x_i^\top x_j \sigma'\left(w_t(0)^\top x_i\right) \sigma'\left(w_r(0)^\top x_j\right) \right| \leq 1.$$

Applying Hoeffding inequality, we have with probability $1 - \frac{\delta}{n^2}$,

$$|[H(0)]_{i,j} - H^*_{i,j}| \leq \left(\frac{2}{m} \log(2n^2/\delta)\right)^{1/2} \leq 4(\frac{\log(n/\delta)}{m})^{1/2} \leq \frac{\epsilon}{n}.$$

Next, applying the union bound over all pairs $(i, j) \in [n] \times [n]$, we have for all (i, j), $\left| [H(0)]_{i,j} - H^*_{i,j} \right| \leq \frac{\epsilon}{n^2}$. To establish the operator norm bound, we simply use the following chain of inequalities

$$\begin{aligned} \|H(0) - H^*\|_2 &\leq \|H(0) - H^*\|_F \\ &= \left(\sum_{ij} | \, [H(0)]_{i,j} - H^*_{i,j}|^2 \right)^{1/2} \\ &\leq (n^2 \cdot \frac{\epsilon^2}{n^2})^{1/2} = \epsilon. \end{aligned}$$

□

Now we proceed to show during training, $H(t)$ is close to $H(0)$. Formally, we prove the following lemma.

Lemma 9.2.3. *Assume $y_i = O(1)$ for all $i = 1, \ldots, n$. Given $t > 0$, suppose that for all $0 \le \tau \le t$, $u_i(\tau) = O(1)$ for all $i = 1, \ldots, n$. If $m = \Omega\left(\frac{n^6 t^2}{\epsilon^2}\right)$, we have*

$$\|H(t) - H(0)\|_2 \le \epsilon.$$

Proof of Lemma 9.2.3. The first key idea is to show that *every weight vector only moves little if m is large.* To show this, let us calculate the movement of a single weight vector w_r.

$$\begin{aligned}
\|w_r(t) - w_r(0)\|_2 =& \left\| \int_0^t \frac{dw_r(\tau)}{d\tau} d\tau \right\|_2 \\
=& \left\| \int_0^t \frac{1}{\sqrt{m}} \sum_{i=1}^n \left(u_i(\tau) - y_i\right) a_r x_i \dot{\sigma}\left(w_r(\tau)^\top x_i\right) d\tau \right\|_2 \\
\le& \frac{1}{\sqrt{m}} \int \left\| \sum_{i=1}^n \left(u_i(\tau) - y_i\right) a_r x_i \dot{\sigma}\left(w_r(\tau)^\top x_i\right) \right\|_2 d\tau \\
\le& \frac{1}{\sqrt{m}} \sum_{i=1}^n \int_0^t \left\| u_i(\tau) - y_i a_r x_i \dot{\sigma}\left(w_r(\tau)^\top x_i\right) \right\|_2 d\tau \\
\le& \frac{1}{\sqrt{m}} \sum_{i=1}^n \int_0^t O(1) d\tau \\
=& O\left(\frac{tn}{\sqrt{m}}\right).
\end{aligned}$$

This calculation shows that at any given time t, $w_r(t)$ is close to $w_r(0)$, as long as m is large. Next, we show this implies the kernel matrix $H(t)$ is close $H(0)$. We calculate the difference on a single entry.

$$
\begin{aligned}
&[H(t)]_{ij} - [H(0)]_{ij} \\
=&\left|\frac{1}{m}\sum_{r=1}^{m}\left(\dot{\sigma}\left(w_r(t)^\top x_i\right)\dot{\sigma}\left(w_r(t)^\top x_j\right) - \dot{\sigma}\left(w_r(0)^\top x_i\right)\dot{\sigma}\left(w_r(0)^\top x_j\right)\right)\right| \\
\leq&\frac{1}{m}\sum_{r=1}^{m}\left|\dot{\sigma}\left(w_r(t)^\top x_i\right)\left(\dot{\sigma}\left(w_r(t)^\top x_j\right) - \dot{\sigma}\left(w_r(0)^\top x_j\right)\right)\right| \\
&+\frac{1}{m}\sum_{r=1}^{m}\left|\dot{\sigma}\left(w_r(0)^\top x_j\right)\left(\dot{\sigma}\left(w_r(t)^\top x_j\right) - \dot{\sigma}\left(w_r(0)^\top x_i\right)\right)\right| \\
\leq&\frac{1}{m}\sum_{r=1}^{m}\left|\max_r \dot{\sigma}\left(w_r(t)^\top x_i\right)\|x_i\|_2\|w_r(t)-w_r(0)\|_2\right| \\
&+\frac{1}{m}\sum_{r=1}^{m}\left|\max_r \dot{\sigma}\left(w_r(t)^\top x_i\right)\|x_i\|_2\|w_r(t)-w_r(0)\|_2\right| \\
=&\frac{1}{m}\sum_{r=1}^{m} O\left(\frac{tn}{\sqrt{m}}\right) \\
=&O\left(\frac{tn}{\sqrt{m}}\right).
\end{aligned}
$$

Therefore, using the same argument as in Lemma 9.2.2, we have

$$\|H(t)-H(0)\|_2 \leq \sum_{i,j}\left|[H(t)]_{ij} - [H(0)]_{ij}\right| = O\left(\frac{tn^3}{\sqrt{m}}\right).$$

Plugging our assumption on m, we finish the proof. □

Several remarks are in sequel.

Remark 1: The assumption that $y_i = O(1)$ is a mild assumption because in practice most labels are bounded by an absolute constant.

Remark 2: The assumption on $u_i(\tau) = O(1)$ for all $\tau \leq t$ and m's dependency on t can be relaxed. This requires a more refined analysis. See [DZPS19].

Remark 3: One can generalize the proof for multi-layer neural network. See [ADH+19b] for more details.

Remark 4: While we only prove the continuous time limit, it is not hard to show with small learning rate (discrete time) gradient descent, $H(t)$ is close to H^*. See [DZPS19].

9.3 *Explaining Optimization and Generalization of Ultra-wide Neural Networks via NTK*

Now we have established the following approximation

$$\frac{du(t)}{dt} \approx -H^* \cdot (u(t) - y) \tag{9.9}$$

where H^* is the NTK matrix. Now we use this approximation to analyze the optimization and generalization behavior of ultra-wide neural networks.

Understanding Optimization The dynamics of $u(t)$ that follows

$$\frac{du(t)}{dt} = -H^* \cdot (u(t) - y)$$

is actually linear dynamical system. For this dynamics, there is a standard analysis. We denote the eigenvalue decomposition of H^* as

$$H^* = \sum_{i=1}^{n} \lambda_i v_i v_i^\top$$

where $\lambda_1 \geq \ldots \geq \lambda_n \geq 0$ are eigenvalues and $v_1, \ldots, v_n$ are eigenvectors. With this decomposition, we consider the dynamics of $u(t)$ on each eigenvector *separately*. Formally, fixing an eigenvevector v_i and multiplying both side by v_i, we obtain

$$\begin{aligned}\frac{dv_i^\top u(t)}{dt} &= -\, v_i^\top H^* \cdot (u(t) - y) \\ &= -\, \lambda_i \left(v_i^\top \left(u(t) - y\right)\right).\end{aligned}$$

Observe that the dynamics of $v_i^\top u(t)$ only depends on itself and λ_i, so this is actually a one dimensional ODE. Moreover, this ODE admits an analytical solution

$$v_i^\top \left(u(t) - y\right) = \exp\left(-\lambda_i t\right) \left(v_i^\top \left(u(0) - y\right)\right). \tag{9.10}$$

Now we use Equation (9.10) to explain why we can find a zero training error solution. We need to assume $\lambda_i > 0$ for all $i = 1, \ldots, n$, i.e., all eigenvalues of this kernel matrix are strictly positive. One can prove this under fairly general conditions. See [DZPS19, DLL+18].

Observe that $(u(t) - y)$ is the difference between predictions and training labels at time t and the algorithm finds a 0 training error solutions means as $t \to \infty$, we have $u(t) - y \to 0$. Equation (9.10) implies that each component of this difference, i.e., $v_i^\top \left(u(t) - y\right)$ is converging to 0 exponentially fast because of the $\exp(-\lambda_i t)$ term. Furthermore, notice that $\{v_1, \ldots, v_n\}$ forms an orthonormal basis of $\mathbb{R}^n$, so $(u(t) - y) = \sum_{i=1}^{n} v_i^\top \left(u(t) - y\right)$. Since we know each $v_i^\top \left(u_i(t) - y\right) \to 0$, we can conclude that $(u(t) - y) \to 0$ as well.

Equation (9.10) actually gives us more information about the convergence. Note each component $v_i^\top (u(t) - y)$ converges to 0 at a different rate. The component that corresponds to larger λ_i converges to 0 at a faster rate than the one with a smaller λ_i. For a set of labels, in order to have faster convergence, we would like the projections

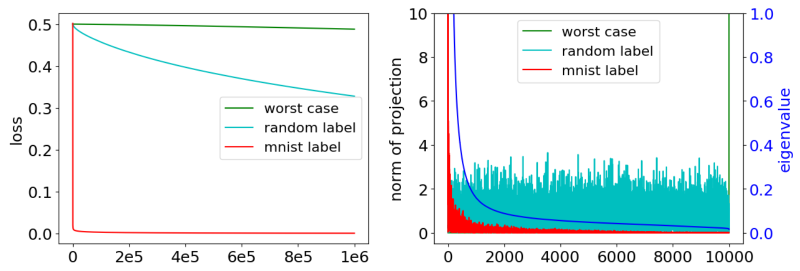


Figure 9.1: Convergence rate vs. projections onto eigenvectors of the kernel matrix.

of y onto the top eigenvectors to be larger.[4] Therefore, we obtain the following intuitive rule to compare the convergence rates in a qualitative manner (for fixed $\|y\|_2$):

- For a set of labels y, if they align with top eigenvectors, i.e., $(v_i^\top y)$ is large for large λ_i, then gradient descent converges quickly.

- For a set of labels, if the projections on eigenvectors $\{(v_i^\top y)\}_{i=1}^n$ are uniform, or labels align with eigenvectors with respect to small eigenvalues, then gradient descent converges with a slow rate.

We can verify this phenomenon experimentally. In Figure 9.1, we compare convergence rates of gradient descent between using original labels, random labels and the worst case labels (normalized eigenvector of H^* corresponding to λ_n. We use the neural network architecture defined in Equation (9.7) with ReLU activation function and only train the first layer. In the right figure, we plot the eigenvalues of H^* as well as projections of true, random, and worst case labels on different eigenvectors of H^*. The experiments use gradient descent on data from two classes of MNIST. The plots demonstrate that original labels have much better alignment with top eigenvectors, thus enjoying faster convergence.

9.3.1 *Understanding Generalization in 2-layer setting*

The approximation in Equation (9.9) implies the final prediction function of ultra-wide neural network is approximately the kernel prediction function defined in Equation (9.6). Therefore, we can just use the generalization theory for kernels to analyze the generalization behavior of ultra-wide neural networks. For the kernel prediction function defined in Equation (9.6), we can use Rademacher complexity bound to derive the following generalization bound for 1-Lipschitz loss function (which is an upper bound of classification error):

$$\frac{\sqrt{2y^\top \left(H^*\right)^{-1} y \cdot \operatorname{tr}\left(H^*\right)}}{n}. \tag{9.11}$$

[4] Here we ignore the effect of $u(0)$ for simplicity. See [ADH+19a] on how to mitigate the effect on $u(0)$.

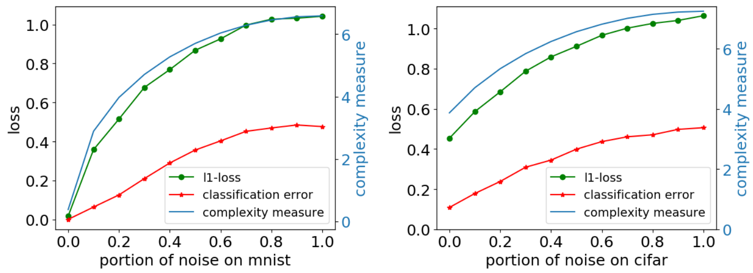


Figure 9.2: Generalization error vs. complexity measure.

This is a *data-dependent* complexity measure that upper bounds the generalization error.

We can check this complexity measure empirically. In Figure 9.2, we compare the generalization error (ℓ_1 loss and classification error) with this complexity measure. We vary the portion of random labels in the dataset to see how the generalization error and the complexity measure change. We use the neural network architecture defined in Equation (9.7) with ReLU activation function and only train the first layer. The left figure uses data from two classes of MNIST and the right figure uses two classes from CIFAR. This complexity measure almost matches the trend of generalization error as the portion of random labels increases.

Learning from simple teacher nets. Now we explain why NTK can learn some functions that can be expressed as two-layer nets. Since we know the optimization error goes to 0 for any label, so it is sufficient to study what teacher nets enables the generalization error to be small. Concretely, we give some examples of two-layer teach nets that make generalization bound (9.11) goes to 0 as $n \to \infty$.

Linear Function: We begin with a simple example. Assume the label $y = \beta^\top x$ for some vector β. Then one can show that (9.11) is upper bounded by $O\left(\frac{\|\beta\|_2}{\sqrt{n}}\right)$. Therefore, NTK can learn linear functions with bounded coefficients.

Two-layer nets with Polynomial Activation: Consider $y = \sum_{j=1}^{k} \alpha_j \left(\beta_j^\top x\right)^p$, i.e., a two-layer net with the activation function being z^p with p being an even number. Similar to the linear function, one can show that (9.11) is upper bounded by $O\left(\frac{p \sum_{j=1}^{k} |\alpha_j| \|\beta_j\|_2^p}{\sqrt{n}}\right)$. Therefore, we can argue NTK can learn two-layer polynomial nets with bounded coefficients.

Cosine activation Beyond polynomials, one can also show NTK can learn somewhat bizarre function. For example, if $y = \sum_{j=1}^{k} \alpha_j \left(\cos\left(\beta_j^\top x\right) - 1\right)$. then we can bound (9.11) is by $O\left(\frac{p \sum_{j=1}^{k} |\alpha_j| \|\beta_j\|_2 \sinh\left(\|\beta\|_2^2\right)}{\sqrt{n}}\right)$.

All the these examples can be proved by a general technique based on Taylor expansion of NTK. See [ADH+19a].

9.4 NTK formula for Multilayer Fully-connected Neural Network

In this section we show case the NTK formulas of fully-connected neural network. We first define a fully-connected neural net formally. Let $x \in \mathbb{R}^d$ be the input, and denote $g^{(0)}(x) = x$ and $d_0 = d$ for notational convenience. We define an L-hidden-layer fully-connected neural network recursively, for $h = 1, 2, \ldots, L$:

$$f^{(h)}(x) = W^{(h)} g^{(h-1)}(x) \in \mathbb{R}^{d_h}, g^{(h)}(x) = \sqrt{\frac{c_\sigma}{d_h}} \sigma\left(f^{(h)}(x)\right) \in \mathbb{R}^{d_h} \tag{9.12}$$

where $W^{(h)} \in \mathbb{R}^{d_h \times d_{h-1}}$ is the weight matrix in the h-th layer ($h \in [L]$), $\sigma : \mathbb{R} \to \mathbb{R}$ is a coordinate-wise activation function, and $c_\sigma = \left(\mathbb{E}_{z \sim \mathcal{N}(0,1)}\left[\sigma z^2\right]\right)^{-1}$. The last layer of the neural network is

$$\begin{aligned} f(w, x) &= f^{(L+1)}(x) = W^{(L+1)} \cdot g^{(L)}(x) \\ &= W^{(L+1)} \cdot \sqrt{\frac{c_\sigma}{d_L}} \sigma W^{(L)} \cdot \sqrt{\frac{c_\sigma}{d_{L-1}}} \sigma W^{(L-1)} \cdots \sqrt{\frac{c_\sigma}{d_1}} \sigma W^{(1)} x, \end{aligned}$$

where $W^{(L+1)} \in \mathbb{R}^{1 \times d_L}$ is the weights in the final layer, and $w = \left(W^{(1)}, \ldots, W^{(L+1)}\right)$ represents all the parameters in the network.

We initialize all the weights to be i.i.d. $\mathcal{N}(0,1)$ random variables[5] , and consider the limit of large hidden widths: $d_1, d_2, \ldots, d_L \to \infty$. The scaling factor $\sqrt{c_\sigma / d_h}$ in Equation (9.12) ensures that the norm of $g^{(h)}(x)$ for each $h \in [L]$ is approximately preserved at initialization (see [DLL+18]). In particular, for ReLU activation, we have $\mathbb{E}\left[\left\|g^{(h)}(x)\right\|_2^2\right] = \|x\|_2^2$ ($\forall h \in [L]$).

Recall from Lemma 9.1.1 that we need to compute the value that $\left\langle \frac{\partial f(w,x)}{\partial w}, \frac{\partial f(w,x')}{\partial w} \right\rangle$ converges to at random initialization in the infinite width limit. We can write the partial derivative with respect to a particular weight matrix $W^{(h)}$ in a compact form:

$$\frac{\partial f(w,x)}{\partial W^{(h)}} = b^{(h)}(x) \cdot \left(g^{(h-1)}(x)\right)^\top, \qquad h = 1, 2, \ldots, L+1,$$

where

$$b^{(h)}(x) = \begin{cases} 1 \in \mathbb{R}, & h = L+1, \\ \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x) \left(W^{(h+1)}\right)^\top b^{(h+1)}(x) \in \mathbb{R}^{d_h}, & h = 1, \ldots, L, \end{cases} \tag{9.13}$$

$$D^{(h)}(x) = \operatorname{diag}\left(\dot{\sigma}\left(f^{(h)}(x)\right)\right) \in \mathbb{R}^{d_h \times d_h}, \qquad h = 1, \ldots, L. \tag{9.14}$$

[5] This scaling is key to the NTK behavior. Initializing with much smaller variance leads to very different behavior.

Then, for any two inputs x and x', and any $h \in [L+1]$, we can compute

$$\begin{aligned}&\left\langle \frac{\partial f(w,x)}{\partial W^{(h)}}, \frac{\partial f(w,x')}{\partial W^{(h)}} \right\rangle \\ =& \left\langle b^{(h)}(x) \cdot \left(g^{(h-1)}(x)\right)^{\top}, b^{(h)}(x') \cdot \left(g^{(h-1)}(x')\right)^{\top} \right\rangle \\ =& \left\langle g^{(h-1)}(x), g^{(h-1)}(x') \right\rangle \cdot \left\langle b^{(h)}(x), b^{(h)}(x') \right\rangle.\end{aligned}$$

Note the first term $\left\langle g^{(h-1)}(x), g^{(h-1)}(x') \right\rangle$ is the covariance between x and x' at the h-th layer. When the width goes to infinity, $\left\langle g^{(h-1)}(x), g^{(h-1)}(x') \right\rangle$ will converge to a fix number, which we denote as $\Sigma^{(h-1)}(x,x')$. This covariance admits a recursive formula, for $h \in [L]$,

$$\begin{aligned}\Sigma^{(0)}(x,x') &= x^{\top}x', \\ \Lambda^{(h)}(x,x') &= \begin{pmatrix} \Sigma^{(h-1)}(x,x) & \Sigma^{(h-1)}(x,x') \\ \Sigma^{(h-1)}(x',x) & \Sigma^{(h-1)}(x',x') \end{pmatrix} \in \mathbb{R}^{2\times 2}, \\ \Sigma^{(h)}(x,x') &= c_\sigma \mathbb{E}_{(u,v)\sim\mathcal{N}\left(0,\Lambda^{(h)}\right)}\left[\sigma(u)\,\sigma(v)\right].\end{aligned} \tag{9.15}$$

Now we proceed to derive this formula. The intuition is that $\left[f^{(h+1)}(x)\right]_i = \sum_{j=1}^{d_h}\left[W^{(h+1)}\right]_{i,j}\left[g^{(h)}(x)\right]_j$ is a centered Gaussian process conditioned on $f^{(h)}$ ($\forall i \in [d_{h+1}]$), with covariance

$$\begin{aligned}&\mathbb{E}\left[\left[f^{(h+1)}(x)\right]_i \cdot \left[f^{(h+1)}(x')\right]_i \middle| f^{(h)}\right] \\ =& \left\langle g^{(h)}(x), g^{(h)}(x') \right\rangle \\ =& \frac{c_\sigma}{d_h}\sum_{j=1}^{d_h}\sigma\left(\left[f^{(h)}(x)\right]_j\right)\sigma\left(\left[f^{(h)}(x')\right]_j\right),\end{aligned} \tag{9.16}$$

which converges to $\Sigma^{(h)}(x,x')$ as $d_h \to \infty$ given that each $\left[f^{(h)}\right]_j$ is a centered Gaussian process with covariance $\Sigma^{(h-1)}$. This yields the inductive definition in Equation (9.15).

Next we deal with the second term $\left\langle b^{(h)}(x), b^{(h)}(x') \right\rangle$. From Equation (9.13) we get

$$\begin{aligned}&\left\langle b^{(h)}(x), b^{(h)}(x') \right\rangle \\ =& \left\langle \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x)\left(W^{(h+1)}\right)^{\top} b^{(h+1)}(x), \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x')\left(W^{(h+1)}\right)^{\top} b^{(h+1)}(x') \right\rangle.\end{aligned} \tag{9.17}$$

Although $W^{(h+1)}$ and $b_{h+1}(x)$ are dependent, the Gaussian initialization of $W^{(h+1)}$ allows us to replace $W^{(h+1)}$ with a fresh new sample $\widetilde{W}^{(h+1)}$ without changing its limit: (See [**?**] for the precise proof.)

$$
\begin{aligned}
&\left\langle \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x) \left(W^{(h+1)}\right)^\top b^{(h+1)}(x), \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x') \left(W^{(h+1)}\right)^\top b^{(h+1)}(x') \right\rangle \\
\approx &\left\langle \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x) \left(\widetilde{W}^{(h+1)}\right)^\top b^{(h+1)}(x), \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x') \left(\widetilde{W}^{(h+1)}\right)^\top b^{(h+1)}(x') \right\rangle \\
\rightarrow &\frac{c_\sigma}{d_h} \operatorname{tr} D^{(h)}(x) D^{(h)}(x') \left\langle b^{(h+1)}(x), b^{(h+1)}(x') \right\rangle \\
\rightarrow &\dot{\Sigma}^{(h)}\left(x, x'\right) \left\langle b^{(h+1)}(x), b^{(h+1)}(x') \right\rangle .
\end{aligned}
$$

Applying this approximation inductively in Equation (9.17), we get

$$
\left\langle b^{(h)}(x), b^{(h)}(x') \right\rangle \rightarrow \prod_{h'=h}^{L} \dot{\Sigma}^{(h')}(x, x').
$$

Finally, since $\left\langle \frac{\partial f(w,x)}{\partial w}, \frac{\partial f(w,x')}{\partial w} \right\rangle = \sum_{h=1}^{L+1} \left\langle \frac{\partial f(w,x)}{\partial W^{(h)}}, \frac{\partial f(w,x')}{\partial W^{(h)}} \right\rangle$, we obtain the final NTK expression for the fully-connected neural network:

$$
\Theta^{(L)}(x, x') = \sum_{h=1}^{L+1} \left(\Sigma^{(h-1)}(x, x') \cdot \prod_{h'=h}^{L+1} \dot{\Sigma}^{(h')}(x, x') \right).
$$

9.5 *NTK in Practice*

Up to now we have shown an ultra-wide neural network with certain initialization scheme and trained by gradient flow correspond to a kernel with a particular kernel function. A natural question is: *why don't we use this kernel classifier directly?*

A recent line of work showed that NTKs can be empirically useful, especially on small to medium scale datasets. Arora et al. [ADL$^+$19] tested the NTK classifier on 90 small to medium scale datasets from UCI database. [6] They found NTK can beat neural networks, other kernels like Gaussian and the best previous classifier, random forest under various metrics, including average rank, average accuracy, etc. This suggests the NTK classifier should belong in any list of off-the-shelf machine learning methods.

For every neural network architecture, one can derive a corresponding kernel function. Du et al. [DHS$^+$19] derived graph NTK (GNTK) for graph classification tasks. On various social network and bioinformatics datasets, GNTK can outperform graph neural networks.

Similarly, Arora et al. [**?**] derived convolutional NTK (CNTK) formula that corresponds to convolutional neural networks. For image classification task, in small-scale data and low-shot settings, CNTKs can be quite strong [ADL$^+$19]. However, for large scale data, Arora

[6] https://archive.ics.uci.edu/ml/datasets.php

et al. [?] found there is still a performance gap between CNTK and CNN. It is an open problem to explain this phenomenon theoretically. This may need to go beyond the NTK framework.

9.6 Exercises

1. NTK formula for ReLU activation function: prove

$$\mathbb{E}_{w\sim\mathcal{N}(0,I)}\left[(\dot{\sigma}(w^\top x)\dot{\sigma}(w^\top x')\right] = \frac{\pi - \arccos\left(\frac{x^\top x'}{\|x\|_2\|x'\|_2}\right)}{2\pi}.$$

2. Prove Equation (9.11). (Hint: The generalization bound depends upon the norm of the difference between the final W matrix and the initial W, which you can upper bound using a sum/integral similar to the analysis of kernel regression in Chapter 3.)

3. Why NTK can learn a linear function: in this question, you are asked to prove that NTK can learn a linear function (the technique here can be used to show NTK can learn other functions listed in Section 9.3.1). In this question, we assume we have n data points, $\{(x_i, y_i)\}_{i=1}^n$ where every input x_i has norm 1 and $y_i = \beta^\top x_i$ for some β. Each entry of neural tangent kernel matrix H^* (induced by the two-layer ReLU neural network) is $H^*_{ij} = \frac{\pi - \arccos\left(\frac{x^\top x'}{\|x\|_2\|x'\|_2}\right)}{2\pi}$.

   (a) Taylor Expansion: use the Taylor expansion of $\arccos(z) = \frac{\pi}{2} - z - \sum_{\ell=1}^{\infty} \frac{(2\ell-3)!!z}{(2\ell-2)!!}$ to show that H^* admits the following form

   $$H^* = \frac{K}{4} + \sum_{\ell=1}^{\infty} \frac{1}{2\pi}\frac{(2\ell-3)!!}{(2\ell-2)!!}\cdot\frac{K^{\circ 2\ell}}{2\ell-1}$$

   where $K_{ij} = x_i^\top x_j$ and $K^\ell_{ij} = \left(x_i^\top x_j\right)^\ell$.

   (b) Assume H^* and K are invertible. Show $\left(H^*\right)^{-1} \preceq 4K^{-1}$.

   (c) Show $\operatorname{tr}\left(H^*\right) \leq n$.

   (d) Using the assumption $y_i = x_i^\top \beta$ to show

   $$\frac{\sqrt{2y^\top(H^*)^{-1}y\cdot \operatorname{tr}(H^*)}}{n} \leq \frac{2\sqrt{2}\,\|\beta\|_2}{\sqrt{n}}.$$

10
Interpreting output of Deep Nets: Credit Attribution

This chapter considers methods that try to understand: *Why did the model give the answer it did?* The basic notions are quite old, but proper and efficient application to deep learning settings is fairly recent. Mathematically, what is needed is to do *credit attribution* for the final decision to various components of the system, including training data.

We consider two types of explanations. *Influence functions* try to understand how individual data points affect the model's answers on test data points. *Saliency methods* try to understand the model's answer on a test data point in terms of the contents of that same data point, typically in the form of a heat map (also called *saliency maps*) depicting the importance of individual coordinates. An elegant idea that often arises in these settings is *Shapley values*.

10.1 Influence Functions

For a fixed training dataset S, the *influence function* captures how adding or removing a datapoint x from the training set S affects the answer (or the loss) on a test datapoint z. Cook and Weisberg's text [1] is the standard reference on the topic. The naive way to compute the influence function is *leave-one-out retraining*: for every x, recompute the model on $S \setminus \{x\}$. But it is also possible to do a more direct computation using the model θ^* trained on S, which uses a more continuous notion of influence.

Let $\ell()$ be a twice-differentiable loss function with $\ell(x, \theta)$ denoting the () loss of model parameters θ on datapoint x. For succinctness we let $\ell(S, \theta)$ denote the average loss on a set S of data points. The formal definition of influence $I(x, z)$ involves[2] the thought experiment of modifying the weighting of a training point x from $1/|S|$ to $1/|S| + \epsilon$. For a fixed S let θ^* be a minimizer of $\ell(S, \theta)$ and $\theta^*_{x,\epsilon}$ be the minimizer after the perturbation.

[1] R D Cook and S Weisberg. Residuals and influence in regression. 1982

[2] Better notation might be $I_S(x, z)$, to clarify dependence on S.

Definition 10.1.1. $I(x,z) = \frac{\partial}{\partial \epsilon}\ell(z, \theta^*_{x,\epsilon})|_{\epsilon=0}$.

Influence functions were invented for convex models such as least-squares linear regression, and thus the theory assumes θ^* satisfies $\nabla_\theta(S,\theta^*) = 0$ and $\nabla^2_\theta(S,\theta^*)$ is positive semi-definite. [3]

Theorem 10.1.2. $I(x,z) = -\nabla_\theta(\ell(z,\theta^*))^T H^{-1}_{\theta^*}\nabla_\theta \ell(x,\theta^*)$.

Proof. By optimality, $\ell(S,\theta^* + \Delta\theta) \approx \ell(S,\theta^*) + \frac{1}{2}(\Delta\theta)^T H_{\theta^*}(\Delta\theta)$ for small perturbations $\Delta\theta$. Changing the weight on datapoint x from $1/\{S\}$ to $1/\{S\} + \epsilon$ and re-optimizing gives $\theta^*_{x,\epsilon} = \theta^* + \Delta\theta$ where $\Delta\theta$ is a minimizer of

$$\epsilon\ell(x,\theta^* + \Delta\theta) + \frac{1}{2}(\Delta\theta)^T H_{\theta^*}(\Delta\theta).$$

Since $\ell(x,\theta^* + \Delta\theta) \approx \ell(x,\theta^*) + \nabla\ell(x,\theta^*)\cdot\Delta\theta$ what we have is a quadratic expression and it is minimized by $\Delta\theta = -\epsilon H^{-1}_{\theta^*}\nabla_\theta(\ell(x,\theta^*))$. Since a change in parameters from θ^* to $\theta^* + \Delta\theta$ causes the loss on a test point z to change by $(\Delta\theta) \times \nabla_\theta(\ell(z,\theta^*))$, the theorem now follows. □

10.1.1 Computing Influence Functions

At first sight, computing influence functions appears difficult due to the inverse Hessian computation, which naively has cubic complexity in the number of parameters. Koh and Liang [4] designed much faster methods. The key idea is a simple identify in the following question.

Problem 10.1.3. *If A is any positive definite matrix with full rank and maximum eigenvalue less than 1 then show that*[5] $A^{-1} = \sum_{i=0}^{\infty}(I-A)^i$.

Agarwal et al.[6] noted how to use this identity for fast (but approximate) Hessian-vector computations.

Problem 10.1.4. *If S_r denotes the truncation of the series to its first r terms, then show that* $\lambda_{max}(A^{-1} - S_r) \leq ??$.

Theorem 10.1.2 shows that we need to compute $H^{-1}v$ for some vector v, but Problem 10.1.4 allows us to approximate it as $\sum_{i=0}^{r}(I - H)^i v$ for some reasonably small r. Since $(I-H)v = v - Hv$ we see that it suffices to do r Hessian-vector computations, each which takes computation time linear in the size of the deep net. (see Section 4.4.1).

10.2 Shapley Values

Shapley Values [7] is a concept from cooperative game theory dealing with the following setting. There is a population of N players (denoted $[N]$ for brevity) who are willing to cooperate towards a certain

[3] In a deep learning setting one hopes to get to a *stationary point*, i.e., $\nabla_\theta(S,\theta^*)$ is zero (or more-realistically, near-zero), which in addition is a *local optimum*, i.e., $\nabla^2_\theta(S,\theta^*)$ is positive semi-definite. In practice neither holds exactly, but $\nabla^2_\theta(S,\theta^*)$ usually does not have large negative eigenvalues. So the influence function computation in practice uses $(H_{\theta^*} + \lambda I)^{-1}$ for some small $\lambda > 0$.

[4] P W Koh and P Liang. Understanding black-box predictions via influence functions. In *Proc. ICML*, 2017

[5] Hint: Note that A is is diaganalizable. How do eigenvectors and eigenvalues of A^i relate to those of A?

[6] Agarwal N, Bullins B, and Hazan E. Second-order stochastic optimization for machine learning in linear time. 2017

[7] Lloyd Shapley. *"Notes on the n-Person Game – II: The Value of an n-Person Game"*. RAND Corporation

goal. A utility function U stipulates the reward/utility for each subset of players: If a subset S of the players end up cooperating, they receive utility $U(S)$ receive as a group. If all N of them end up cooperating, what is the appropriate and fair way to split the utility $U([N])$ among them? Under some reasonable conditions, there turn out to be unique payments $s_1, s_2, \ldots, s_N$, called *Shapley values* such that $\sum_i s_i = U([N])$ (Theorem 10.2.3).

In ML the following two settings are representative of use of Shapley values: (a) *Pricing of datapoints:* "players"could be individuals holding data that could be useful for training an ML model, and the Shapley values can be seen as payments for use of their data. (b) When we try to understand the output of a deep net on a single (test) datapoint in terms of the contributions (aka *saliency*) of individual coordinates towards the deep net's decision.

Shapley value of the ith player is defined using the thought experiment of the players adding themselves to the coalition in a random order, and looking at the expected increase in utility when the ith player joins the coalition.

Definition 10.2.1. Shapley value *of player i, denoted s_i, is defined as the following expected value, where π is a random permutation of $\{1, 2, \ldots, N\}$ and $\pi_{<i}$ is shorthand for the subset of players that appear before i in the permutation:*

$$s_i = E_\pi \left[U(\pi_{\leq i}) - U(\pi_{<i}) \right]. \tag{10.1}$$

The definition of s_i's is sort of natural, though one may quibble about the slightly artificial assumption of the players joining the coalition in a random order. Here is an equivalent definition that avoids the random order.

Problem 10.2.2. *Show that the following definition of Shapley value is equivalent:*

$$\begin{aligned} s_i \quad &= \textstyle\sum_{S \subseteq [N] \setminus \{i\}} \frac{\{S\}!(N - \{S\} - 1)!)}{N!} (U(S \cup \{i\}) - U(S)) \\ &= \textstyle\frac{1}{N} \sum_{S \subseteq [N] \setminus \{i\}} \frac{U(S \cup \{i\}) - U(S)}{\binom{N-1}{|S|}}. \end{aligned}$$

Since $\binom{N-1}{k}$ is the number of subsets of size k in a set of size $N-1$, Problem 10.2.2 in effect redefines Shapley value s_i as the expectation of the following random process: randomly pick an integer k from $[0, N-1]$, then a random subset S of size k but not containing i, and measure the change in utility upon adding i to S.

While this seems more natural, it is still a good idea to understand whether we are missing out on some radically different definition of how to distribute credit. Let's try to formalize natural properties for any method of credit attribution. The following axioms seem natural for any system of defining s_i's.

Efficiency: Sum of the players' values is $U([N])$.

Symmetry: If $U(S \cup \{i\}) = U(S \cup \{j\})$ for all S not containing i, j then their payments are the same. [8]

Linearity: If U_1, U_2 are any two utility functions then the payments for $U_1 + U_2$ are the sum of the payments for U_1 and the payments for U_2.

Null Player: If $U(S \cup \{i\}) = U(S)$ for all S not containing i, then the payment for i is zero.

You can quickly convince yourself that Shapley values satisfy the axioms.

Theorem 10.2.3. *The payment scheme in Definition 10.2.1 is the only one that satisfies the previous axioms.*

Problem 10.2.4. *Prove Theorem 10.2.3.* [9]

10.2.1 Algorithms to approximate Shapley values

Given a succinct description of the utility function U (e.g., as a circuit) it is NP-hard in general to compute Shapley values [10] meaning (assuming $P \neq NP$) that the running time is going to increase faster than any polynomial of N and the description of U. However, it is possible to compute them approximately if the utilities are bounded. Specifically, we assume an upper bound of R on the absolute value of $U(S \cup \{i\}) - U(S)$ for all S, i.
Naive approximation: Pick $O(R^2 N \log N/\epsilon^2)$ random permutations and use them to estimate the expectation in (10.1). (Note that the number of computations of $U(\cdot)$ is $O(R^2 N^2 \log N/\epsilon^2)$. This is the computational cost.) Then concentration bounds imply that the estimate $\widehat{s}_i$ of the expectation lies within $[s_i - \epsilon/\sqrt{N}, s_i + \epsilon/\sqrt{N}]$. Which implies that the vector of all Shapley values is estimated within ℓ_2 norm ϵ, namely, $\|s - \widehat{s}\|_2 \leq \epsilon$.
Better approximation: We give a method that uses $O(R^2 N \log N)$ evaluations of $U(\cdot)$. It uses the following fact about Shapley values.

Theorem 10.2.5. *Differences of Shapley values satisfy the following:*

$$s_i - s_j = \frac{1}{N-1} \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{U(S \cup \{i\}) - U(S \cup \{j\})}{\binom{N-2}{|S|}}.$$

Problem 10.2.6. *Prove Theorem 10.2.5 from Problem 10.2.2.*

Now we can describe the Method: (1) Use naive approximation to approximate s_1 within additive error $\epsilon/2\sqrt{N}$. (2) Use the characterization in Theorem 10.2.5 to sample sets S suitably to estimate all differences of type $s_1 - s_j$ within error $\epsilon/2\sqrt{N}$.

[8] Sometimes this is described as "payments should depend upon their contribution, not on their names."

[9] Hint: Take the difference of the two payment schemes.

[10] X Deng and C Papadimitriou. On the complexity of cooperative solution concepts. *Math of Operations Research*, 1994

Problem 10.2.7. *Figure out how to do step (2). (Hint: You pick S with a certain probability $p(|S|)$. This uses the observation that the same S can be used to estimate $s_1 - s_j$ for many j's.)*

10.3 Data Models

This is a method that also assigns credit to individual training datapoints, but uses a linear regression approach [11]. By training many models on subsets of p fraction of datapoints in the training set, the authors show that some interesting measures of test error (defined using logit values) behave as follows: the measure $f(x)$ is well-approximable as a (sparse) linear expression $\theta_0 + \sum_i \theta_i x_i$, where x is a binary vector denoting a sample of p fraction of training datapoints, with $x_i = 1$ indicating presence of i-th training point and $x_i = -1$ denoting absence. The coefficients θ_i are estimated via lasso regression. The surprise here is that $f(x)$ —which is the result of deep learning on dataset x—is well-approximated by $\theta_0 + \sum_i \theta_i x_i$. The authors note that the θ_i's can be viewed as heuristic estimates for the discrete influence of the ith datapoint. Note that the estimated θ_i depends on the value of p in the above procedure.

Why do data models work? The reason has to do with the fact that the models are being trained on a random subset of p fraction of the training set. One can understand it using interesting but elementary harmonic analysis [12].

10.4 Saliency Maps

Saliency methods try to understand the model's answer on a test data point in terms of the contents of that same data point, typically in the form of a heat map (also called *saliency maps*) depicting the importance of individual coordinates. For example, if the deep net is labeling images by their labels, then a saliency map for an image that has been labeled "dog" might involve highlighting the pixels that were determinative for assigning this label.

TO BE WRITTEN. SHAPLEY VALUES CAN BE USED TO DEFINE THE "CONTRIBUTION" OF EACH COORDINATE IN THE TEST DATAPOINT TO THE FINAL OUTPUT. THERE ARE OTHER METHODS.

Also gradient-based methods.

[11] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022

[12] Mark Braverman Sanjeev Aror Nikunj Saunshi, Arushi Gupta. Understanding influence functions and data models via harmonic analysis. *ICLR*, 2023

11

Inductive Biases due to Algorithmic Regularization

Many successful modern machine learning systems based on deep neural networks are over-parametrized, i.e., the number of parameters is typically much larger than the sample size. In other words, there exist (infinitely) many (approximate) minimizers of the empirical risk, many of which would not generalize well on the unseen data. For learning to succeed then, it is crucial to bias the learning algorithm towards "simpler" hypotheses by trading off empirical loss with a certain complexity term that ensures that empirical and population risks are close. Several explicit regularization strategies have been used in practice to help these systems generalize, including ℓ_1 and ℓ_2 regularization of the parameters [NH92].

Besides explicit regularization techniques, practitioners have used a spectrum of algorithmic approaches to improve the generalization ability of over-parametrized models. This includes early stopping of back-propagation [CLG01], batch normalization [IS15b], dropout [SHK$^+$14], and more[1]. While these heuristics have enjoyed tremendous success in training deep networks, a theoretical understanding of how these heuristics provide regularization in deep learning remains somewhat limited.

In this chapter, we investigate regularization due to Dropout, an algorithmic heurisitic recently proposed by [SHK$^+$14]. The basic idea when training a neural network using dropout, is that during a forward pass, we randomly drop neurons in the neural network, independently and identically according to a Bernoulli distribution. Specifically, at each round of the back-propagation algorithm, for each neuron, independently, with probability p we "drop" the neuron, so it does not participate in making a prediction for the given data point, and with probability $1 - p$ we retain that neuron [2].

Deep learning is a field where key innovations have been driven by practitioners, with several techniques motivated by drawing insights from other fields. For instance, Dropout was introduced as a way of breaking up "co-adaptation" among neurons, drawing

[1] We refer the reader to [KGC17] for an excellent exposition of over 50 of such proposals.

[2] The parameter p is treated as a hyperparameter which we typically tune for based on a validation set.

insights from the success of the sexual reproduction model in the evolution of advanced organisms. Another motivation that was cited by [SHK+14] was in terms of "balancing networks". Despite several theoretical works aimed at explaining Dropout [3], it remains unclear what kind of regularization does Dropout provide or what kinds of networks does Dropout prefer and how that helps with generalization. In this chapter, we work towards that goal by instantiating explicit forms of regularizers due to Dropout and how they provide capacity control in various machine learning including linear regression (Section 11.4), matrix sensing (Section 11.1.1), matrix completion (Section 11.1.2), and deep learning (Section 11.2).

[3]

11.1 Matrix Sensing

We begin with understanding dropout for matrix sensing, a problem which arguably is an important instance of a matrix learning problem with lots of applications, and is well understood from a theoretical perspective. Here is the problem setup.

Let $\mathrm{M}_* \in \mathbb{R}^{d_2 \times d_0}$ be a matrix with rank $r_* := \mathrm{Rank}(\mathrm{M}_*)$. Let $\mathrm{A}^{(1)}, \ldots, \mathrm{A}^{(n)}$ be a set of measurement matrices of the same size as M_*. The goal of matrix sensing is to recover the matrix M_* from n observations of the form $y_i = \langle \mathrm{M}_*, \mathrm{A}^{(i)} \rangle$ such that $n \ll d_2 d_0$. The learning algorithm we consider is empirical risk minimization, and we choose to represent the parameter matrix $\mathrm{M} \in \mathbb{R}^{d_2 \times d_0}$ in terms of product of its factors $\mathrm{U} \in \mathbb{R}^{d_2 \times d_1}, \mathrm{V} \in \mathbb{R}^{d_0 \times d_1}$:

$$\min_{\mathrm{U} \in \mathbb{R}^{d_2 \times d_1}, \mathrm{V} \in \mathbb{R}^{d_0 \times d_1}} \widehat{L}(\mathrm{U}, \mathrm{V}) := \frac{1}{n} \sum_{i=1}^{n} (y_i - \langle \mathrm{U}\mathrm{V}^\top, \mathrm{A}^{(i)} \rangle)^2. \tag{11.1}$$

When $d_1 \gg r_*$, there exist many "bad" empirical minimizers, i.e., those with a large true risk. However, recently, [LMZ18] showed that under restricted isometry property, despite the existence of such poor ERM solutions, gradient descent with proper initialization is *implicitly* biased towards finding solutions with minimum nuclear norm – this is an important result which was first conjectured and empirically verified by [GWB+17].

We propose solving the ERM problem (11.1) with algorithmic regularization due to dropout, where at training time, the corresponding columns of U and V are dropped independently and identically according to a Bernoulli random variable. As opposed to the *implicit* effect of gradient descent, this dropout heuristic *explicitly* regularizes the empirical objective. It is then natural to ask, in the case of matrix sensing, if dropout also biases the ERM towards certain low norm solutions. To answer this question, we begin with the observation that dropout can be viewed as an instance of SGD on the following objective:

$$\widehat{L}_{\text{drop}}(\mathrm{U},\mathrm{V}) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\mathrm{B}}(y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)}\rangle)^2, \tag{11.2}$$

where $\mathrm{B} \in \mathbb{R}^{d_1 \times d_1}$ is a diagonal matrix whose diagonal elements are Bernoulli random variables distributed as $\mathrm{B}_{jj} \sim \frac{1}{1-p}\mathrm{Ber}(1-p)$, for $j \in [d_1]$. In this case, we can show that for any $p \in [0,1)$:

$$\widehat{L}_{\text{drop}}(\mathrm{U},\mathrm{V}) = \widehat{L}(\mathrm{U},\mathrm{V}) + \frac{p}{1-p}\widehat{R}(\mathrm{U},\mathrm{V}), \tag{11.3}$$

where

$$\widehat{R}(\mathrm{U},\mathrm{V}) = \sum_{j=1}^{d_1}\frac{1}{n}\sum_{i=1}^{n}(\mathrm{u}_j^\top \mathrm{A}^{(i)}\mathrm{v}_j)^2 \tag{11.4}$$

is a data-dependent term that captures the *explicit* regularizer due to dropout.

Proof. Consider one of the summands in the Dropout objective in Equation 11.2. Then, we can write

$$\begin{aligned}\mathbb{E}_{\mathrm{B}}[(y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)}\rangle)^2] \quad &= \quad \Big(\mathbb{E}_{\mathrm{B}}[y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)}\rangle]\Big)^2 \\ &\quad + \mathrm{Var}(y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)}\rangle) \end{aligned} \tag{11.5}$$

For Bernoulli random variable B_{jj}, we have that $\mathbb{E}[\mathrm{B}_{jj}] = 1$ and $\mathrm{Var}(\mathrm{B}_{jj}) = \frac{p}{1-p}$. Thus, the first term on right hand side is equal to $(y_i - \langle \mathrm{UV}^\top, \mathrm{A}^{(i)}\rangle)^2$. For the second term we have

$$\begin{aligned}\mathrm{Var}(y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)}\rangle) &= \mathrm{Var}(\langle \mathrm{UBV}^\top, \mathrm{A}^{(i)}\rangle) \\ &= \mathrm{Var}(\langle \mathrm{B}, \mathrm{U}^\top \mathrm{A}^{(i)}\mathrm{V}\rangle) \\ &= \mathrm{Var}(\sum_{j=1}^{d_1}\mathrm{B}_{jj}\mathrm{u}_j^\top \mathrm{A}^{(i)}\mathrm{v}_j) \\ &= \sum_{j=1}^{d_1}(\mathrm{u}_j^\top \mathrm{A}^{(i)}\mathrm{v}_j)^2\,\mathrm{Var}(\mathrm{B}_{jj}) \\ &= \frac{p}{1-p}\sum_{j=1}^{d_1}(\mathrm{u}_j^\top \mathrm{A}^{(i)}\mathrm{v}_j)^2\end{aligned}$$

Using the facts above in Equation (11.2), we get

$$\begin{aligned}\widehat{L}_{\text{drop}}(\mathrm{U},\mathrm{V}) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \langle \mathrm{UV}^\top, \mathrm{A}^{(i)}\rangle)^2 + \frac{1}{n}\sum_{i=1}^{n}\frac{p}{1-p}\sum_{j=1}^{d_1}(\mathrm{u}_j^\top \mathrm{A}^{(i)}\mathrm{v}_j)^2 \\ &= \widehat{L}(\mathrm{U},\mathrm{V}) + \frac{p}{1-p}\widehat{R}(\mathrm{U},\mathrm{V}).\end{aligned}$$

which completes the proof. □

Provided that the sample size n is large enough, the *explicit* regularizer on a given sample behaves much like its expected value with respect to the underlying data distribution [4]. Further, given that we seek a minimum of $\widehat{L}_{\text{drop}}$, it suffices to consider the factors with the minimal value of the regularizer among all that yield the same empirical loss. This motivates studying the the following distribution-dependent *induced* regularizer:

$$\Theta(\mathrm{M}) := \min_{\mathrm{UV}^\top=\mathrm{M}} R(\mathrm{U},\mathrm{V}), \;\; \text{where} \;\; R(\mathrm{U},\mathrm{V}) := \mathbb{E}_{\mathrm{A}}[\widehat{R}(\mathrm{U},\mathrm{V})].$$

Next, we consider two two important examples of random sensing matrices.

11.1.1 Gaussian Sensing Matrices

We assume that the entries of the sensing matrices are independently and identically distributed as standard Gaussian, i.e., $\mathrm{A}^{(i)}_{k\ell} \sim \mathcal{N}(0,1)$. For Gaussian sensing matrices, we show that the induced regularizer due to Dropout provides nuclear-norm regularization. Formally, we show that

$$\Theta(\mathrm{M}) = \frac{1}{d_1}\|\mathrm{M}\|_*^2. \tag{11.6}$$

Proof. We recall the general form for the dropout regularizer for the matrix sensing problem in Equation 11.4, and take expectation with respect to the distribution on the sensing matrices. Then, for any pair of factors (U,V), it holds that the expected regularizer is given as follows.

$$\begin{aligned}
R(\mathrm{U},\mathrm{V}) &= \sum_{j=1}^{d_1} \mathbb{E}(\mathrm{u}_j^\top \mathrm{A} \mathrm{v}_j)^2 \\
&= \sum_{j=1}^{d_1} \mathbb{E}(\sum_{k=1}^{d_2}\sum_{\ell=1}^{d_0} \mathrm{U}_{kj}\mathrm{A}_{k\ell}\mathrm{V}_{\ell j})^2 \\
&= \sum_{j=1}^{d_1} \sum_{k,k'=1}^{d_2} \sum_{\ell,\ell'=1}^{d_0} \mathrm{U}_{kj}\mathrm{U}_{k'j}\mathrm{V}_{\ell j}\mathrm{V}_{\ell' j}\,\mathbb{E}[\mathrm{A}_{k\ell}\mathrm{A}_{k'\ell'}] \\
&= \sum_{j=1}^{d_1} \sum_{k=1}^{d_2} \sum_{\ell=1}^{d_0} \mathrm{U}_{kj}^2\mathrm{V}_{\ell j}^2\,\mathbb{E}[\mathrm{A}_{k\ell}^2] \\
&= \sum_{j=1}^{d_1} \sum_{k=1}^{d_2} \sum_{\ell=1}^{d_0} \mathrm{U}_{kj}^2\mathrm{V}_{\ell j}^2 \\
&= \sum_{j=1}^{d_1} \|\mathrm{u}_j\|^2\|\mathrm{v}_j\|^2
\end{aligned}$$

[4] Under mild assumptions, we can formally show that the dropout regularizer is well concentrated around its mean

Now, using the Cauchy-Schwartz inequality, we can bound the expected regularizer as

$$\begin{aligned} R(\mathrm{U},\mathrm{V}) &\geq \frac{1}{d_1}\left(\sum_{i=1}^{d_1}\|\mathrm{u}_i\|\|\mathrm{v}_i\|\right)^2 \\ &= \frac{1}{d_1}\left(\sum_{i=1}^{d_1}\|\mathrm{u}_i\mathrm{v}_i^\top\|_*\right)^2 \\ &\geq \frac{1}{d_1}\left(\|\sum_{i=1}^{d_1}\mathrm{u}_i\mathrm{v}_i^\top\|_*\right)^2 = \frac{1}{d_1}\|\mathrm{U}\mathrm{V}^\top\|_*^2 \end{aligned}$$

where the equality follows because for any pair of vectors a,b, it holds that $\|\mathrm{a}\mathrm{b}^\top\|_* = \|\mathrm{a}\mathrm{b}^\top\|_F = \|\mathrm{a}\|\|\mathrm{b}\|$, and the last inequality is due to triangle inequality.

Next, we need the following key result from [MAV18].

Theorem 11.1.1. For any pair of matrices $\mathrm{U} \in \mathbb{R}^{d_2\times d_1}, \mathrm{V} \in \mathbb{R}^{d_0\times d_1}$, there exists a rotation matrix $\mathrm{Q} \in \mathrm{SO}(d_1)$ such that matrices $\widetilde{\mathrm{U}} := \mathrm{U}\mathrm{Q}, \widetilde{\mathrm{V}} := \mathrm{V}\mathrm{Q}$ satisfy $\|\widetilde{\mathrm{u}}_i\|\|\widetilde{\mathrm{v}}_i\| = \frac{1}{d_1}\|\mathrm{U}\mathrm{V}^\top\|_*$, for all $i \in [d_1]$.

Using Theorem 11.1.1 on (U,V), the expected dropout regularizer at $\mathrm{U}\mathrm{Q},\mathrm{V}\mathrm{Q}$ is given as

$$\begin{aligned} R(\mathrm{U}\mathrm{Q},\mathrm{V}\mathrm{Q}) &= \sum_{i=1}^{d_1}\|\mathrm{U}\mathrm{q}_i\|^2\|\mathrm{V}\mathrm{q}_i\|^2 \\ &= \sum_{i=1}^{d_1}\frac{1}{d_1^2}\|\mathrm{U}\mathrm{V}^\top\|_*^2 \\ &= \frac{1}{d_1}\|\mathrm{U}\mathrm{V}^\top\|_*^2 \\ &\leq \Theta(\mathrm{U}\mathrm{V}^\top) \end{aligned}$$

which completes the proof. □

For completeness we provide a proof of Theorem 11.1.1.

Proof. Define $\mathrm{M} := \mathrm{U}\mathrm{V}^\top$. Let $\mathrm{M} = \mathrm{W}\Sigma\mathrm{Y}^\top$ be compact SVD of M. Define $\widehat{\mathrm{U}} := \mathrm{W}\Sigma^{1/2}$ and $\widehat{\mathrm{V}} := \mathrm{Y}\Sigma^{1/2}$. Let $\mathrm{G}_\mathrm{U} = \widehat{\mathrm{U}}^\top\widehat{\mathrm{U}}$ and $\mathrm{G}_\mathrm{V} = \widehat{\mathrm{V}}^\top\widehat{\mathrm{V}}$ be respective Gram matrices. Observe that $\mathrm{G}_\mathrm{U} = \mathrm{G}_\mathrm{V} = \Sigma$. We will show that there exists a rotation Q such that for $\widetilde{\mathrm{U}} = \widehat{\mathrm{U}}\mathrm{Q}$, $\widetilde{\mathrm{V}} = \widehat{\mathrm{V}}\mathrm{Q}$, it holds that

$$\|\widetilde{\mathrm{u}}_j\|^2 = \frac{1}{d_1}\|\widetilde{\mathrm{U}}\|_F^2 = \frac{1}{d_1}\mathrm{Tr}(\widetilde{\mathrm{U}}^\top\widetilde{\mathrm{U}}) = \frac{1}{d_1}\mathrm{Tr}(\Sigma) = \frac{1}{d_1}\|\mathrm{M}\|_*$$

and

$$\|\widetilde{\mathrm{v}}_j\|^2 = \frac{1}{d_1}\|\widetilde{\mathrm{V}}\|_F^2 = \frac{1}{d_1}\mathrm{Tr}(\widetilde{\mathrm{V}}^\top\widetilde{\mathrm{V}}) = \frac{1}{d_1}\mathrm{Tr}(\Sigma) = \frac{1}{d_1}\|\mathrm{M}\|_*$$

Consequently, it holds that $\|\tilde{u}_i\|\|\tilde{v}_i\| = \frac{1}{d_1}\|M\|_*$.

All that remains is to give a construction of matrix Q. We note that a rotation matrix Q satisfies the desired properties above if and only if all diagonal elements of $Q^\top G_U Q$ are equal[5], and equal to $\frac{\operatorname{Tr} G_U}{d_1}$. The key idea is that for the trace zero matrix $G_1 := G_U - \frac{\operatorname{Tr} G_U}{d_1} I_{d_1}$, if $G_1 = \sum_{i=1}^r \lambda_i e_i e_i^\top$ is an eigendecomposition of G_1, then for the average of the eigenvectors, i.e. for $w_{11} = \frac{1}{\sqrt{r}} \sum_{i=1}^r e_i$, it holds that $w_{11}^\top G_1 w_{11} = 0$. We use this property recursively to exhibit an orthogonal transformation Q, such that $Q^\top G_1 Q$ is zero on its diagonal.

To verify the claim, first notice that w_{11} is unit norm

$$\|w_{11}\|^2 = \|\frac{1}{\sqrt{r}} \sum_{i=1}^r e_i\|^2 = \frac{1}{r} \sum_{i=1}^r \|e_i\|^2 = 1.$$

Further, it is easy to see that

$$w_{11}^\top G w_{11} = \frac{1}{r} \sum_{i,j=1}^r e_i G e_j = \frac{1}{r} \sum_{i,j=1}^r \lambda_j e_i^\top e_j = \frac{1}{r} \sum_{i=1}^r \lambda_i = 0.$$

Complete $W_1 := [w_{11}, w_{12}, \cdots, w_{1d}]$ be such that $W_1^\top W_1 = W_1 W_1^\top = I_d$. Observe that $W_1^\top G_1 W_1$ has zero on its first diagonal elements

$$W_1^\top G_1 W_1 = \begin{bmatrix} 0 & b_1^\top \\ b_1 & G_2 \end{bmatrix}$$

The principal submatrix G_2 also has a zero trace. With a similar argument, let $w_{22} \in \mathbb{R}^{d-1}$ be such that $\|w_{22}\| = 1$ and $w_{22}^\top G_2 w_{22} = 0$ and define $W_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ o & w_{22} & w_{23} & \cdots & w_{2d} \end{bmatrix} \in \mathbb{R}^{d \times d}$ such that $W_2^\top W_2 = W_2 W_2^\top = I_d$, and observe that

$$(W_1 W_2)^\top G_1 (W_1 W_2) = \begin{bmatrix} 0 & \cdot & \cdots \\ \cdot & 0 & \cdots \\ \vdots & \vdots & G_3 \end{bmatrix}.$$

This procedure can be applied recursively so that for the matrix $Q = W_1 W_2 \cdots W_d$ we have

$$Q^\top G_1 Q = \begin{bmatrix} 0 & \cdot & \cdots & \cdot \\ \cdot & 0 & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots \\ \cdot & \cdot & \cdot & 0 \end{bmatrix},$$

so that $\operatorname{Tr}(\widetilde{U}\widetilde{U}^\top) = \operatorname{Tr}(Q^\top G_U Q) = \operatorname{Tr}(\Sigma) = \operatorname{Tr}(Q^\top G_V Q) = \operatorname{Tr}(\widetilde{V}^\top \widetilde{V})$.

□

[5] since $(Q^\top G_U Q)_{jj} = \|\tilde{u}_j\|^2$

11.1.2 Matrix Completion

Next, we consider the problem of matrix completion which can be formulated as a special case of matrix sensing with sensing matrices that random indicator matrices. Formally, we assume that for all $j \in [n]$, let $\mathrm{A}^{(j)}$ be an indicator matrix whose (i,k)-th element is selected randomly with probability $p(i)q(k)$, where $p(i)$ and $q(k)$ denote the probability of choosing the i-th row and the j-th column, respectively.

We will show next that in this setup Dropout induces the *weighted trace-norm* studied by [SS10] and [FSSS11]. Formally, we show that

$$\Theta(\mathrm{M}) = \frac{1}{d_1}\|\mathrm{diag}(\sqrt{p})\mathrm{U}\mathrm{V}^\top\mathrm{diag}(\sqrt{q})\|_*^2. \tag{11.7}$$

Proof. For any pair of factors (U,V) it holds that

$$\begin{aligned}
R(\mathrm{U},\mathrm{V}) &= \sum_{j=1}^{d_1}\mathbb{E}(\mathrm{u}_j^\top \mathrm{A}\mathrm{v}_j)^2\\
&= \sum_{j=1}^{d_1}\sum_{k=1}^{d_2}\sum_{\ell=1}^{d_0} p(k)q(\ell)(\mathrm{u}_j^\top \mathrm{e}_k\mathrm{e}_\ell^\top \mathrm{v}_j)^2\\
&= \sum_{j=1}^{d_1}\sum_{k=1}^{d_2}\sum_{\ell=1}^{d_0} p(k)q(\ell)\mathrm{U}(k,j)^2\mathrm{V}(\ell,j)^2\\
&= \sum_{j=1}^{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{u}_j\|^2\|\sqrt{\mathrm{diag}(\mathrm{q})}\mathrm{v}_j\|^2\\
&\geq \frac{1}{d_1}\left(\sum_{j=1}^{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{u}_j\|\|\sqrt{\mathrm{diag}(\mathrm{q})}\mathrm{v}_j\|\right)^2\\
&= \frac{1}{d_1}\left(\sum_{j=1}^{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{u}_j\mathrm{v}_j^\top\sqrt{\mathrm{diag}(\mathrm{q})}\|_*\right)^2\\
&\geq \frac{1}{d_1}\left(\|\sqrt{\mathrm{diag}(\mathrm{p})}\sum_{j=1}^{d_1}\mathrm{u}_j\mathrm{v}_j^\top\sqrt{\mathrm{diag}(\mathrm{q})}\|_*\right)^2\\
&= \frac{1}{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}\mathrm{V}^\top\sqrt{\mathrm{diag}(\mathrm{q})}\|_*^2
\end{aligned}$$

where the first inequality is due to Cauchy-Schwartz and the second inequality follows from the triangle inequality. The equality right after the first inequality follows from the fact that for any two vectors a,b, $\|\mathrm{a}\mathrm{b}^\top\|_* = \|\mathrm{a}\mathrm{b}^\top\|_F = \|\mathrm{a}\|\|\mathrm{b}\|$. Since the inequalities hold for any U,V, it implies that

$$\Theta(\mathrm{U}\mathrm{V}^\top) \geq \frac{1}{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}\mathrm{V}^\top\sqrt{\mathrm{diag}(\mathrm{q})}\|_*^2.$$

Applying Theorem 11.1.1 on $(\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}, \sqrt{\mathrm{diag}(\mathrm{q})}\mathrm{V})$, there exists a rotation matrix Q such that

$$\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}\mathrm{q}_j\|\|\sqrt{\mathrm{diag}(\mathrm{q})}\mathrm{V}\mathrm{q}_j\| = \frac{1}{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}\mathrm{V}^\top\sqrt{\mathrm{diag}(\mathrm{q})}\|_*.$$

We evaluate the expected dropout regularizer at UQ, VQ:

$$\begin{aligned} R(\mathrm{UQ},\mathrm{VQ}) &= \sum_{j=1}^{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}\mathrm{q}_j\|^2\|\sqrt{\mathrm{diag}(\mathrm{q})}\mathrm{V}\mathrm{q}_j\|^2 \\ &= \sum_{j=1}^{d_1}\frac{1}{d_1^2}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}\mathrm{V}^\top\sqrt{\mathrm{diag}(\mathrm{q})}\|_*^2 \\ &= \frac{1}{d_1}\|\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}\mathrm{V}^\top\sqrt{\mathrm{diag}(\mathrm{q})}\|_*^2 \\ &\leq \Theta(\mathrm{U}\mathrm{V}^\top) \end{aligned}$$

which completes the proof. □

The results above are interesting because they connect Dropout, an algorithmic heuristic in deep learning, to strong complexity measures that are empirically effective as well as theoretically well understood. To illustrate, here we give a generalization bound for matrix completion with dropout in terms of the value of the *explicit* regularizer at the minimum of the empirical problem.

Theorem 11.1.2. Without loss of generality, assume that $d_2 \geq d_0$ and $\|\mathrm{M}_*\| \leq 1$. Furthermore, assume that $\min_{i,j} p(i)q(j) \geq \frac{\log(d_2)}{n\sqrt{d_2 d_0}}$. Let (U,V) be a minimizer of the dropout ERM objective in equation (11.2), and assume that $\max_i \|\mathrm{U}(i,:)\|^2 \leq \gamma$, $\max_i \|\mathrm{V}(i,:)\|^2 \leq \gamma$. Let α be such that $\widehat{R}(\mathrm{U},\mathrm{V}) \leq \alpha/d_1$. Then, for any $\delta \in (0,1)$, the following generalization bounds holds with probability at least $1-2\delta$ over a sample of size n:

$$L(\mathrm{U},\mathrm{V}) \leq \widehat{L}(\mathrm{U},\mathrm{V}) + C(1+\gamma)\sqrt{\frac{\alpha d_2\log(d_2)}{n}} + C'(1+\gamma^2)\sqrt{\frac{\log(2/\delta)}{2n}}$$

as long as $n = \Omega\left((d_1\gamma^2/\alpha)^2\log(2/\delta)\right)$, where C, C' are some absolute constants.

The proof of Theorem 11.1.2 follows from standard generalization bounds for ℓ_2 loss [MRT18] based on the Rademacher complexity [BM02] of the class of functions with weighted trace-norm bounded by $\sqrt{\alpha}$, i.e. $\mathcal{M}_\alpha := \{\mathrm{M} : \|\mathrm{diag}(\sqrt{\mathrm{p}})\mathrm{M}\mathrm{diag}(\sqrt{\mathrm{q}})\|_*^2 \leq \alpha\}$. A bound on the Rademacher complexity of this class was established by [FSSS11]. The technicalities here include showing that the explicit regularizer is well concentrated around its expected value, as well as deriving a bound on the supremum of the predictions. A few remarks are in order.

We require that the sampling distributions be non-degenerate, as specified by the condition $\min_{i,j} p(i)q(j) \geq \frac{\log(d_2)}{n\sqrt{d_2 d_0}}$. This is a natural requirement for bounding the Rademacher complexity of $\mathcal{M}_\alpha$, as discussed in [FSSS11].

We note that for large enough sample size, $\widehat{R}(\mathrm{U},\mathrm{V}) \approx R(\mathrm{U},\mathrm{V}) \approx \Theta(\mathrm{UV}^\top) = \frac{1}{d_1}\|\mathrm{diag}(\sqrt{p})\mathrm{UV}^\top\mathrm{diag}(\sqrt{\mathrm{q}})\|_*^2$, where the second approximation is due the fact that the pair (U,V) is a minimizer. That is, compared to the weighted trace-norm, the value of the explicit regularizer at the minimizer roughly scales as $1/d_1$. Hence the assumption $\widehat{R}(\mathrm{U},\mathrm{V}) \leq \alpha/d_1$ in the statement of the corollary.

In practice, for models that are trained with dropout, the training error $\widehat{L}(\mathrm{U},\mathrm{V})$ is negligible. Moreover, given that the sample size is large enough, the third term can be made arbitrarily small. Having said that, the second term, which is $\widetilde{O}(\gamma\sqrt{\alpha d_2/n})$, dominates the right hand side of generalization error bound in Theorem 11.1.2.

The assumption $\max_i \|\mathrm{U}(i,:)\|^2 \leq \gamma$, $\max_i \|\mathrm{V}(i,:)\|^2 \leq \gamma$ is motivated by the practice of deep learning; such *max-norm* constraints are typically used with dropout, where the norm of the vector of incoming weights at each hidden unit is constrained to be bound by a constant [SHK+14]. In this case, if a dropout update violates this constraint, the weights of the hidden unit are projected back to the constraint norm ball. In proofs, we need this assumption to give a concentration bound for the empirical explicit regularizer, as well as bound the supremum deviation between the predictions and the true values. We remark that the value of γ also determines the complexity of the function class. On one hand, the generalization gap explicitly depends on and increases with γ. However, when γ is large, the constraints on U,V are milder, so that $\widehat{L}(\mathrm{U},\mathrm{V})$ can be made smaller.

Finally, the required sample size heavily depends on the value of the explicit regularizer at the optima (α/d_1), and hence, on the dropout rate p. In particular, increasing the dropout rate increases the regularization parameter $\lambda := \frac{p}{1-p}$, thereby intensifies the penalty due to the explicit regularizer. Intuitively, a larger dropout rate p results in a smaller α, thereby a tighter generalization gap can be guaranteed. We show through experiments that that is indeed the case in practice.

11.2 *Deep neural networks*

Next, we focus on neural networks with multiple hidden layers. Let $\mathcal{X} \subseteq \mathbb{R}^{d_0}$ and $\mathcal{Y} \subseteq \mathbb{R}^{d_k}$ denote the input and output spaces, respectively. Let $\mathcal{D}$ denote the joint probability distribution on $\mathcal{X} \times \mathcal{Y}$. Given n examples $\{(\mathrm{x}_i, \mathrm{y}_i)\}_{i=1}^n \sim \mathcal{D}^n$ drawn i.i.d. from the joint distribution and a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, the goal of learning is to find a hypothesis $f_{\mathrm{w}} : \mathcal{X} \to \mathcal{Y}$, parameterized by w, that has a small *population risk* $L(\mathrm{w}) := \mathbb{E}_{\mathcal{D}}[\ell(f_{\mathrm{w}}(\mathrm{x}), \mathrm{y})]$.

We focus on the squared ℓ_2 loss, i.e., $\ell(\mathrm{y}, \mathrm{y}') = \|\mathrm{y} - \mathrm{y}'\|^2$, and study the generalization properties of the dropout algorithm for minimizing the *empirical risk* $\widehat{L}(\mathrm{w}) := \frac{1}{n}\sum_{i=1}^{n}[\|\mathrm{y}_i - f_{\mathrm{w}}(\mathrm{x}_i)\|^2]$. We consider the hypothesis class associated with feed-forward neural networks with k layers, i.e., functions of the form $f_{\mathrm{w}}(\mathrm{x}) = \mathrm{W}_k\sigma(\mathrm{W}_{k-1}\sigma(\cdots \mathrm{W}_2\sigma(\mathrm{W}_1\mathrm{x})\cdots))$, where $\mathrm{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$, for $i \in [k]$, is the weight matrix at i-th layer. The parameter w is the collection of weight matrices $\{\mathrm{W}_k, \mathrm{W}_{k-1}, \ldots, \mathrm{W}_1\}$ and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function applied entrywise to an input vector.

In modern machine learning systems, rather than talk about a certain network topology, we should think in terms of layer topology where each layer could have different characteristics – for example, fully connected, locally connected, or convolutional. In convolutional neural networks, it is a common practice to apply dropout only to the fully connected layers and not to the convolutional layers. Furthermore, in deep regression, it has been observed that applying dropout to only one of the hidden layers is most effective [LMAPH19]. In our study, dropout is applied on top of the learned representations or *features*, i.e. the output of the top hidden layer. In this case, dropout updates can be viewed as stochastic gradient descent iterates on the *dropout objective*:

$$\widehat{L}_{\mathrm{drop}}(\mathrm{w}) := \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\mathrm{B}}\|\mathrm{y}_i - \mathrm{W}_k\mathrm{B}\sigma(\mathrm{W}_{k-1}\sigma(\cdots\mathrm{W}_2\sigma(\mathrm{W}_1\mathrm{x}_i)\cdots))\|^2 \tag{11.8}$$

where B is a diagonal random matrix with diagonal elements distributed identically and independently as $\mathrm{B}_{ii} \sim \frac{1}{1-p}\mathrm{Bern}(1-p)$, $i \in [d_{k-1}]$, for some *dropout rate* p. We seek to understand the *explicit* regularizer due to dropout:

$$\widehat{R}(\mathrm{w}) := \widehat{L}_{\mathrm{drop}}(\mathrm{w}) - \widehat{L}(\mathrm{w}) \qquad \text{(explicit regularizer)}$$

We denote the output of the i-th hidden node in the j-th hidden layer on an input vector x by $a_{i,j}(\mathrm{x}) \in \mathbb{R}$; for example, $a_{1,2}(\mathrm{x}) = \sigma(\mathrm{W}_2(1,:)^\top\sigma(\mathrm{W}_1\mathrm{x}))$. Similarly, the vector $\mathrm{a}_j(\mathrm{x}) \in \mathbb{R}^{d_j}$ denotes the activation of the j-th layer on input x. Using this notation, we can conveniently rewrite the Dropout objective (see Equation 11.8) as $\widehat{L}_{\mathrm{drop}}(\mathrm{w}) := \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\mathrm{B}}\|\mathrm{y}_i - \mathrm{W}_k\mathrm{B}\mathrm{a}_{k-1}(\mathrm{x}_i)\|^2$. It is then easy to show that the explicit regularizer due to dropout is given as follows.

Proposition 11.2.1 (Dropout regularizer in deep regression).

$$\widehat{L}_{\mathrm{drop}}(\mathrm{w}) = \widehat{L}(\mathrm{w}) + \widehat{R}(\mathrm{w}), \text{ where } \widehat{R}(\mathrm{w}) = \lambda\sum_{j=1}^{d_{k-1}}\|\mathrm{W}_k(:,j)\|^2\widehat{a}_j^2.$$

where $\widehat{a}_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_{j,k-1}(\mathrm{x}_i)^2}$ and $\lambda = \frac{p}{1-p}$ is the regularization parameter.

Proof. Recall that $\mathbb{E}[\mathrm{B}_{ii}] = 1$ and $\mathrm{Var}(\mathrm{B}_{ii}) = \frac{p}{1-p}$. Conditioned on x, y in the current mini-batch, we have that

$$\mathbb{E}_{\mathrm{B}}[\|\mathrm{y} - \mathrm{W}_k\mathrm{B}\mathrm{a}_{k-1}(\mathrm{x})\|^2] = \sum_{i=1}^{d_k} \mathbb{E}_{\mathrm{B}}(y_i - \mathrm{W}_k(i,:)^\top \mathrm{B}\mathrm{a}_{k-1}(\mathrm{x}))^2. \qquad (11.9)$$

The following holds for each of the summands above:

$$\begin{aligned}\mathbb{E}_{\mathrm{B}}(y_i - \mathrm{W}_k(i,:)^\top \mathrm{B}\mathrm{a}_{k-1}(\mathrm{x}))^2 \quad &= \quad \Big(\mathbb{E}_{\mathrm{B}}[y_i - \mathrm{W}_k(i,:)^\top \mathrm{B}\mathrm{a}_{k-1}(\mathrm{x})]\Big)^2 \\ &\quad + \mathrm{Var}(y_i - \mathrm{W}_k(i,:)^\top \mathrm{B}\mathrm{a}_{k-1}(\mathrm{x})).\end{aligned}$$

Since $\mathbb{E}[\mathrm{B}] = \mathrm{I}$, the first term on right hand side is equal to $(y_i - \mathrm{W}_k(:,i)^\top \mathrm{a}_{k-1}(\mathrm{x}))^2$. For the second term we have

$$\begin{aligned}\mathrm{Var}(y_i - \mathrm{W}_k(i,:)^\top \mathrm{B}\mathrm{a}_{k-1}(\mathrm{x})) &= \mathrm{Var}(\mathrm{W}_k(i,:)^\top \mathrm{B}\mathrm{a}_{k-1}(\mathrm{x})) \\ &= \mathrm{Var}(\sum_{j=1}^{d_{k-1}} \mathrm{W}_k(i,j)\mathrm{B}_{jj} a_{j,k-1}(\mathrm{x})) \\ &= \sum_{j=1}^{d_{k-1}} (\mathrm{W}_k(i,j) a_{j,k-1}(\mathrm{x}))^2 \,\mathrm{Var}(\mathrm{B}_{jj}) \\ &= \frac{p}{1-p} \sum_{j=1}^{d_{k-1}} \mathrm{W}_k(i,j)^2 a_{j,k-1}(\mathrm{x})^2\end{aligned}$$

Plugging the above into Equation (11.9)

$$\begin{aligned}\mathbb{E}_{\mathrm{B}}[\|\mathrm{y} - \mathrm{W}_k\mathrm{B}\mathrm{a}_{k-1}(\mathrm{x})\|^2] \quad &= \quad \|\mathrm{y} - \mathrm{W}_k\mathrm{a}_{k-1}(\mathrm{x})\|^2 \\ &\quad + \frac{p}{1-p}\sum_{j=1}^{d_{k-1}} \|\mathrm{W}_k(:,j)\|^2 a_{j,k-1}(\mathrm{x})^2\end{aligned}$$

Now taking the empirical average with respect to x, y, we get

$$\widehat{L}_{\mathrm{drop}}(\mathrm{w}) = \widehat{L}(\mathrm{w}) + \frac{p}{1-p}\sum_{j=1}^{d_{k-1}} \|\mathrm{W}_k(:,j)\|^2 \widehat{a}_j^2 = \widehat{L}(\mathrm{w}) + \widehat{R}(\mathrm{w})$$

which completes the proof. □

The explicit regularizer $\widehat{R}(\mathrm{w})$ is the summation over hidden nodes, of the product of the squared norm of the outgoing weights with the empirical second moment of the output of the corresponding neuron. For a two layer neural network with ReLU, when the input distribution is symmetric and isotropic, the expected regularizer is equal to the squared ℓ_2 path-norm of the network [NTS15b]. Such a connection has been previously established for deep linear networks [MAV18, MA19]; here we extend that result to single hidden layer ReLU networks.

Proposition 11.2.2. Consider a two layer neural network $f_{\mathrm{w}}(\cdot)$ with ReLU activation functions in the hidden layer. Furthermore, assume that the marginal input distribution $\mathbb{P}_{\mathcal{X}}(\mathrm{x})$ is symmetric and isotropic, i.e., $\mathbb{P}_{\mathcal{X}}(\mathrm{x}) = \mathbb{P}_{\mathcal{X}}(-\mathrm{x})$ and $\mathbb{E}[\mathrm{x}\mathrm{x}^\top] = \mathrm{I}$. Then the expected explicit regularizer due to dropout is given as

$$R(\mathrm{w}) := \mathbb{E}[\widehat{R}(\mathrm{w})] = \frac{\lambda}{2} \sum_{i_0,i_1,i_2=1}^{d_0,d_1,d_2} \mathrm{W}_2(i_2,i_1)^2 \mathrm{W}_1(i_1,i_0)^2, \tag{11.10}$$

Proof of Proposition 11.2.2. Using Proposition 11.2.1, we have that:

$$R(\mathrm{w}) = \mathbb{E}[\widehat{R}(\mathrm{w})] = \lambda \sum_{j=1}^{d_1} \|\mathrm{W}_2(:,j)\|^2\, \mathbb{E}[\sigma(\mathrm{W}_1(j,:)^\top \mathrm{x})^2]$$

It remains to calculate the quantity $\mathbb{E}_{\mathrm{x}}[\sigma(\mathrm{W}_1(j,:)^\top \mathrm{x})^2]$. By symmetry assumption, we have that $\mathbb{P}_{\mathcal{X}}(\mathrm{x}) = \mathbb{P}_{\mathcal{X}}(-\mathrm{x})$. As a result, for any $\mathrm{v} \in \mathbb{R}^{d_0}$, we have that $\mathbb{P}(\mathrm{v}^\top \mathrm{x}) = \mathbb{P}(-\mathrm{v}^\top \mathrm{x})$ as well. That is, the random variable $z_j := \mathrm{W}_1(j,:)^\top \mathrm{x}$ is also symmetric about the origin. It is easy to see that $\mathbb{E}_z[\sigma(z)^2] = \frac{1}{2}\mathbb{E}_z[z^2]$.

$$\begin{aligned}
\mathbb{E}_z[\sigma(z)^2] &= \int_{-\infty}^{\infty} \sigma(z)^2 d\mu(z) \\
&= \int_{0}^{\infty} \sigma(z)^2 d\mu(z) = \int_{0}^{\infty} z^2 d\mu(z) \\
&= \frac{1}{2}\int_{-\infty}^{\infty} z^2 d\mu(z) = \frac{1}{2}\mathbb{E}_z[z^2].
\end{aligned}$$

Plugging back the above identity in the expression of $R(\mathrm{w})$, we get that

$$R(\mathrm{w}) = \frac{\lambda}{2} \sum_{j=1}^{d_1} \|\mathrm{W}_2(:,j)\|^2\, \mathbb{E}[(\mathrm{W}_1(j,:)^\top \mathrm{x})^2] = \frac{\lambda}{2} \sum_{j=1}^{d_1} \|\mathrm{W}_2(:,j)\|^2 \|\mathrm{W}_1(j,:)\|^2$$

where the second equality follows from the assumption that the distribution is isotropic. □

11.3 Landscape of the Optimization Problem

While the focus in Section 11.2 was on understanding the implicit bias of dropout in terms of the global optima of the resulting regularized learning problem, here we focus on computational aspects of dropout as an optimization procedure. Since dropout is a first-order method and the landscape of the Dropout objective (e.g., Problem 11.11) is highly non-convex, we can perhaps only hope to find a *local* minimum, that too provided if the problem has no degenerate saddle points [LSJR16, GHJY15b]. Therefore, in this section, we pose the following questions: *What is the implicit bias of dropout in terms of* *local minima? Do local minima share anything with global minima structurally or in terms of the objective? Can dropout find a local optimum?*

For the sake of simplicity of analysis, we focus on the case of single hidden layer *linear* autoencoders with tied weights, i.e. $\mathrm{U} = \mathrm{V}$. We assume that the input distribution is isotropic, i.e. $\mathrm{C_x} = \mathrm{I}$. In this case, the population risk reduces to

$$\begin{aligned}\mathbb{E}[\|\mathrm{y} - \mathrm{UU}^\top \mathrm{x}\|^2] &= \mathrm{Tr}\,(\mathrm{C_y}) - 2\langle \mathrm{C_{yx}}, \mathrm{UU}^\top\rangle + \|\mathrm{UU}^\top\|_F^2 \\ &= \|\mathrm{M} - \mathrm{UU}^\top\|_F^2 + \mathrm{Tr}\,(\mathrm{C_y}) - \|\mathrm{C_{yx}}\|_F^2\end{aligned}$$

where $\mathrm{M} = \frac{\mathrm{C_{yx}}+\mathrm{C_{xy}}}{2}$. Ignoring the terms that are independent of the weights matrix U, the goal is to minimize $L(\mathrm{U}) = \|\mathrm{M} - \mathrm{UU}^\top\|_F^2$. Using Dropout amounts to solving the following problem:

$$\min_{\mathrm{U}\in\mathbb{R}^{d_0\times d_1}} L_\theta(\mathrm{U}) := \|\mathrm{M} - \mathrm{UU}^\top\|_F^2 + \underbrace{\lambda \sum_{i=1}^{d_1} \|\mathrm{u}_i\|^4}_{R(\mathrm{U})} \tag{11.11}$$

We can characterize the global optima of the problem above as follows.

Theorem 11.3.1. For any $j \in [r]$, let $\kappa_j := \frac{1}{j}\sum_{i=1}^{j} \lambda_i(\mathrm{C_{yx}})$. Furthermore, define $\rho := \max\{j \in [r] : \lambda_j(\mathrm{C_{yx}}) > \frac{\lambda j \kappa_j}{r+\lambda j}\}$. Then, if U_* is a global optimum of Problem 11.11, it satisfies that $\mathrm{U}_*\mathrm{U}_*^\top = \mathcal{S}_{\frac{\lambda\rho\kappa_\rho}{r+\lambda\rho}}(\mathrm{C_{yx}})$.

Next, it is easy to see that the gradient of the objective of Problem 11.11 is given by

$$\nabla L_\theta(\mathrm{U}) = 4(\mathrm{UU}^\top - \mathrm{M})\mathrm{U} + 4\lambda \mathrm{U}\mathrm{diag}(\mathrm{U}^\top\mathrm{U}).$$

We also make the following important observation about the critical points of Problem 11.11. Lemma 11.3.2 allows us to bound different norms of the critical points, as will be seen later in the proofs.

Lemma 11.3.2. If U is a critical point of Problem 11.11, then it holds that $\mathrm{UU}^\top \preceq \mathrm{M}$.

Proof of Lemma 11.3.2. Since $\nabla L_\theta(\mathrm{U}) = 0$, we have that

$$(\mathrm{M} - \mathrm{UU}^\top)\mathrm{U} = \lambda \mathrm{U}\mathrm{diag}(\mathrm{U}^\top\mathrm{U})$$

multiply both sides from right by $\mathrm{U}^\top$ and rearrange to get

$$\mathrm{MUU}^\top = \mathrm{UU}^\top\mathrm{UU}^\top + \lambda \mathrm{U}\mathrm{diag}(\mathrm{U}^\top\mathrm{U})\mathrm{U}^\top \tag{11.12}$$

Note that the right hand side is symmetric, which implies that the left hand side must be symmetric as well, i.e.

$$\mathrm{MUU}^\top = (\mathrm{MUU}^\top)^\top = \mathrm{UU}^\top\mathrm{M},$$

so that M and $\mathrm{UU}^\top$ commute. Note that in Equation (11.12), $\mathrm{U}\mathrm{diag}(\mathrm{U}^\top\mathrm{U})\mathrm{U}^\top \succeq 0$. Thus, $\mathrm{MUU}^\top \succeq \mathrm{UU}^\top\mathrm{UU}^\top$. Let $\mathrm{UU}^\top = \mathrm{W}\Gamma\mathrm{W}^\top$ be a compact eigendecomposition of $\mathrm{UU}^\top$. We get

$$\mathrm{MUU}^\top = \mathrm{MW}\Gamma\mathrm{W}^\top \succeq \mathrm{UU}^\top\mathrm{UU}^\top = \mathrm{W}\Gamma^2\mathrm{W}^\top.$$

Multiplying from right and left by $\mathrm{W}\Gamma^{-1}$ and $\mathrm{W}^\top$ respectively, we get that $\mathrm{W}^\top\mathrm{MW} \succeq \Gamma$ which completes the proof. □

We show in Section 11.3.1 that (a) local minima of Problem 11.11 inherit the same implicit bias as the global optima, i.e. all local minima are equalized. Then, in Section 11.3.2, we show that for sufficiently small regularization parameter, (b) there are no spurious local minima, i.e. all local minima are global, and (c) all saddle points are non-degenerate (see Definition 11.3.4).

11.3.1 Implicit bias in local optima

Recall that the population risk $L(\mathrm{U})$ is rotation invariant, i.e. $L(\mathrm{UQ}) = L(\mathrm{U})$ for any rotation matrix Q. Now, if the weight matrix U were not equalized, then there exist indices $i, j \in [r]$ such that $\|\mathrm{u}_i\| > \|\mathrm{u}_j\|$. We show that it is easy to design a rotation matrix (equal to identity everywhere expect for columns i and j) that moves mass from u_i to u_j such that the difference in the norms of the corresponding columns of UQ decreases strictly while leaving the norms of other columns invariant. In other words, this rotation strictly reduces the regularizer and hence the objective. Formally, this implies the following result.

Lemma 11.3.3. All local minima of Problem 11.11 are equalized, i.e. if U is a local optimum, then $\|\mathrm{u}_i\| = \|\mathrm{u}_j\| \ \forall i, j \in [r]$.

Lemma 11.3.3 unveils a fundamental property of dropout. As soon as we perform dropout in the hidden layer – *no matter how small the dropout rate* – all local minima become equalized. We illustrate this using a toy problem in Figure 11.1.

Proof of Lemma 11.3.3. We show that if U is not equalized, then any ϵ-neighborhood of U contains a point with dropout objective strictly smaller than $L_\theta(\mathrm{U})$. More formally, for any $\epsilon > 0$, we exhibit a rotation Q_ϵ such that $\|\mathrm{U} - \mathrm{UQ}_\epsilon\|_F \leq \epsilon$ and $L_\theta(\mathrm{UQ}_\epsilon) < L_\theta(\mathrm{U})$. Let U be a critical point of Problem 11.11 that is not equalized, i.e. there exists two columns of U with different norms. Without loss of generality, let $\|\mathrm{u}_1\| > \|\mathrm{u}_2\|$. We design a rotation matrix Q such that it is almost an isometry, but it moves mass from u_1 to u_2. Consequently, the new factor becomes "less un-equalized" and achieves a smaller

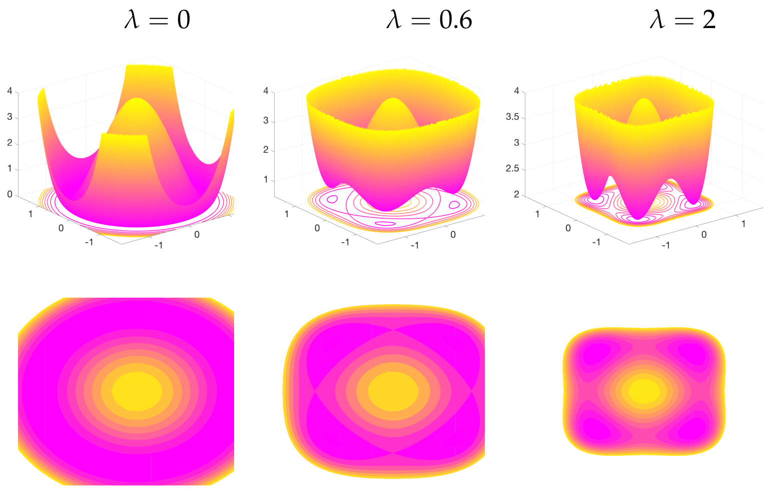

$\lambda = 0$ $\lambda = 0.6$ $\lambda = 2$

Figure 11.1: Optimization landscape (top) and contour plot (bottom) for a single hidden-layer linear autoencoder network with one dimensional input and output and a hidden layer of width $r = 2$ with dropout, for different values of the regularization parameter λ. Left: for $\lambda = 0$ the problem reduces to squared loss minimization, which is rotation invariant as suggested by the level sets. Middle: for $\lambda > 0$ the global optima shrink toward the origin. All local minima are global, and are equalized, i.e. the weights are parallel to the vector $(\pm 1, \pm 1)$. Right: as λ increases, global optima shrink further.

regularizer, while preserving the value of the loss. To that end, define

$$\mathrm{Q}_\delta := \begin{bmatrix} \sqrt{1-\delta^2} & -\delta & 0 \\ \delta & \sqrt{1-\delta^2} & 0 \\ 0 & 0 & \mathrm{I}_{r-2} \end{bmatrix}$$

and let $\widehat{\mathrm{U}} := \mathrm{U}\mathrm{Q}_\delta$. It is easy to verify that Q_ϵ is indeed a rotation. First, we show that for any ϵ, as long as $\delta^2 \leq \frac{\epsilon^2}{2\,\mathrm{Tr}(\mathrm{M})}$, we have $\widehat{\mathrm{U}} \in \mathcal{B}_\epsilon(\mathrm{U})$:

$$\begin{aligned} \|\mathrm{U} - \widehat{\mathrm{U}}\|_F^2 &= \sum_{i=1}^{r} \|\mathrm{u}_i - \hat{\mathrm{u}}_i\|^2 \\ &= \|\mathrm{u}_1 - \sqrt{1-\delta^2}\mathrm{u}_1 - \delta \mathrm{u}_2\|^2 + \|\mathrm{u}_2 - \sqrt{1-\delta^2}\mathrm{u}_2 + \delta \mathrm{u}_1\|^2 \\ &= 2(1 - \sqrt{1-\delta^2})(\|\mathrm{u}_1\|^2 + \|\mathrm{u}_2\|^2) \\ &\leq 2\delta^2\,\mathrm{Tr}(\mathrm{M}) \leq \epsilon^2 \end{aligned}$$

where the second to last inequality follows from Lemma 11.3.2, because $\|\mathrm{u}_1\|^2 + \|\mathrm{u}_2\|^2 \leq \|\mathrm{U}\|_F^2 = \mathrm{Tr}(\mathrm{U}\mathrm{U}^\top) \leq \mathrm{Tr}(\mathrm{M})$, and also the fact that $1 - \sqrt{1-\delta^2} = \frac{1-1+\delta^2}{1+\sqrt{1-\delta^2}} \leq \delta^2$.

Next, we show that for small enough δ, the value of L_θ at $\widehat{\mathrm{U}}$ is strictly smaller than that of U. Observe that

$$\begin{aligned} \|\hat{\mathrm{u}}_1\|^2 &= (1-\delta^2)\|\mathrm{u}_1\|^2 + \delta^2\|\mathrm{u}_2\|^2 + 2\delta\sqrt{1-\delta^2}\mathrm{u}_1^\top \mathrm{u}_2 \\ \|\hat{\mathrm{u}}_2\|^2 &= (1-\delta^2)\|\mathrm{u}_2\|^2 + \delta^2\|\mathrm{u}_1\|^2 - 2\delta\sqrt{1-\delta^2}\mathrm{u}_1^\top \mathrm{u}_2 \end{aligned}$$

and the remaining columns will not change, i.e. for $i = 3, \cdots, r$, $\hat{\mathrm{u}}_i = \mathrm{u}_i$. Together with the fact that Q_δ preserves the norms, i.e. $\|\mathrm{U}\|_F = \|\mathrm{U}\mathrm{Q}_\delta\|_F$, we get

$$\|\hat{\mathrm{u}}_1\|^2 + \|\hat{\mathrm{u}}_2\|^2 = \|\mathrm{u}_1\|^2 + \|\mathrm{u}_2\|^2. \tag{11.13}$$

Let $\delta = -c \cdot \text{sgn}(\text{u}_1^\top \text{u}_2)$ for a small enough $c > 0$ such that $\|\text{u}_2\| < \|\hat{\text{u}}_2\| \leq \|\hat{\text{u}}_1\| < \|\text{u}_1\|$. Using Equation (11.13), This implies that $\|\hat{\text{u}}_1\|^4 + \|\hat{\text{u}}_2\|^4 < \|\text{u}_1\|^4 + \|\text{u}_2\|^4$, which in turn gives us $R(\widehat{\text{U}}) < R(\text{U})$ and hence $L_\theta(\widehat{\text{U}}) < L_\theta(\text{U})$. Therefore, a non-equalized critical point cannot be local minimum, hence the first claim of the lemma. □

11.3.2 *Landscape properties*

Next, we characterize the solutions to which dropout converges. We do so by understanding the optimization landscape of Problem 11.11. Central to our analysis, is the following notion of *strict saddle property*.

Definition 11.3.4 (Strict saddle point/property)**.** Let $f : \mathcal{U} \to \mathbb{R}$ be a twice differentiable function and let $\text{U} \in \mathcal{U}$ be a critical point of f. Then, U is a *strict saddle point* of f if the Hessian of f at U has at least one negative eigenvalue, i.e. $\lambda_{\min}(\nabla^2 f(\text{U})) < 0$. Furthermore, f satisfies *strict saddle property* if all saddle points of f are strict saddle.

Strict saddle property ensures that for any critical point U that is not a local optimum, the Hessian has a significant negative eigenvalue which allows first order methods such as gradient descent (GD) and stochastic gradient descent (SGD) to escape saddle points and converge to a local minimum [LSJR16, GHJY15b]. Following this idea, there has been a flurry of works on studying the landscape of different machine learning problems, including low rank matrix recovery [BNS16b], generalized phase retrieval problem [SQW16b], matrix completion [GLM16b], deep linear networks [Kaw16], matrix sensing and robust PCA [GJZ17b] and tensor decomposition [GHJY15b], making a case for global optimality of first order methods.

For the special case of no regularization (i.e. $\lambda = 0$; equivalently, no dropout), Problem 11.11 reduces to standard squared loss minimization which has been shown to have no spurious local minima and satisfy strict saddle property (see, e.g. [BH89, JGN+17]). However, the regularizer induced by dropout can potentially introduce new spurious local minima as well as degenerate saddle points. Our next result establishes that that is not the case, at least when the dropout rate is sufficiently small.

Theorem 11.3.5. Let $r := \text{Rank}(\text{M})$. Assume that $d_1 \leq d_0$ and that the regularization parameter satisfies $\lambda < \frac{r\lambda_r(\text{M})}{(\sum_{i=1}^r \lambda_i(\text{M})) - r\lambda_r(\text{M})}$. Then it holds for Problem 11.11 that

1. all local minima are global,

2. all saddle points are strict saddle points.

A few remarks are in order. First, the assumption $d_1 \leq d_0$ is by no means restrictive, since the network map $\text{UU}^\top \in \mathbb{R}^{d_0 \times d_0}$ has rank at most d_0, and letting $d_1 > d_0$ does not increase the expressivity of the function class represented by the network. Second, Theorem 11.3.5 guarantees that any critical point U that is not a global optimum is a strict saddle point, i.e. $\nabla^2 L(\mathrm{U}, \mathrm{U})$ has a negative eigenvalue. This property allows first order methods, such as dropout, to escape such saddle points. Third, note that the guarantees in Theorem 11.3.5 hold when the regularization parameter λ is sufficiently small. Assumptions of this kind are common in the literature (see, for example [GJZ17b]). While this is a *sufficient* condition for the result in Theorem 11.3.5, it is not clear if it is *necessary*.

Proof of Theorem 11.3.5. Here we outline the main steps in the proof of Theorem 11.3.5.

1. In Lemma 11.3.3, we show that the set of non-equalized critical points does not include any local optima. Furthermore, Lemma 11.3.6 shows that all such points are strict saddles.

2. In Lemma 11.3.7, we give a closed-form characterization of all the equalized critical points in terms of the eigendecompostion of M. We then show that if λ is chosen appropriately, all such critical points that are not global optima, are strict saddle points.

3. It follows from Item 1 and Item 2 that if λ is chosen appropriately, then all critical points that are not global optimum, are strict saddle points.

□

Lemma 11.3.6. *All critical points of Problem 11.11 that are not equalized, are strict saddle points.*

Proof of Lemma 11.3.6. By Lemma 11.3.3, the set of non-equalized critical points does not include any local optima. We show that all such points are strict saddles. Let U be a critical point that is not equalized. To show that U is a strict saddle point, it suffices to show that the Hessian has a negative eigenvalue. In here, we exhibit a curve along which the second directional derivative is negative. Assume, without loss of generality that $\|\mathrm{u}_1\| > \|\mathrm{u}_2\|$ and consider the curve

$$\Delta(t) := [(\sqrt{1-t^2}-1)\mathrm{u}_1 + t\mathrm{u}_2, (\sqrt{1-t^2}-1)\mathrm{u}_2 - t\mathrm{u}_1, 0_{d,r-2}]$$

It is easy to check that for any $t \in \mathbb{R}$, $L(\mathrm{U} + \Delta(t)) = L(\mathrm{U})$ since $\mathrm{U} + \Delta(t)$ is essentially a rotation on U and L is invariant under rotations.

Observe that

$$\begin{aligned} g(t) &:= L_\theta(\mathrm{U} + \Delta(t)) \\ &= L_\theta(\mathrm{U}) + \|\sqrt{1-t^2}\mathrm{u}_1 + t\mathrm{u}_2\|^4 - \|\mathrm{u}_1\|^4 + \|\sqrt{1-t^2}\mathrm{u}_2 - t\mathrm{u}_1\|^4 - \|\mathrm{u}_2\|^4 \\ &= L_\theta(\mathrm{U}) - 2t^2(\|\mathrm{u}_1\|^4 + \|\mathrm{u}_2\|^4) + 8t^2(\mathrm{u}_1\mathrm{u}_2)^2 + 4t^2\|\mathrm{u}_1\|^2\|\mathrm{u}_2\|^2 \\ &\qquad + 4t\sqrt{1-t^2}\mathrm{u}_1^\top\mathrm{u}_2(\|\mathrm{u}_1\|^2 - \|\mathrm{u}_2\|^2) + O(t^3). \end{aligned}$$

The derivative of g then is given as

$$\begin{aligned} g'(t) &= -4t(\|\mathrm{u}_1\|^4 + \|\mathrm{u}_2\|^4) + 16t(\mathrm{u}_1\mathrm{u}_2)^2 + 8t\|\mathrm{u}_1\|^2\|\mathrm{u}_2\|^2 \\ &\qquad + 4(\sqrt{1-t^2} - \frac{t^2}{\sqrt{1-t^2}})(\mathrm{u}_1^\top\mathrm{u}_2)(\|\mathrm{u}_1\|^2 - \|\mathrm{u}_2\|^2) + O(t^2). \end{aligned}$$

Since U is a critical point and L_θ is continuously differentiable, it should hold that

$$g'(0) = 4(\mathrm{u}_1^\top\mathrm{u}_2)(\|\mathrm{u}_1\|^2 - \|\mathrm{u}_2\|^2) = 0.$$

Since by assumption $\|\mathrm{u}_1\|^2 - \|\mathrm{u}_2\|^2 > 0$, it should be the case that $\mathrm{u}_1^\top\mathrm{u}_2 = 0$. We now consider the second order directional derivative:

$$\begin{aligned} g''(0) &= -4(\|\mathrm{u}_1\|^4 + \|\mathrm{u}_2\|^4) + 16(\mathrm{u}_1\mathrm{u}_2)^2 + 8\|\mathrm{u}_1\|^2\|\mathrm{u}_2\|^2 \\ &= -4(\|\mathrm{u}_1\|^2 - \|\mathrm{u}_2\|^2)^2 < 0 \end{aligned}$$

which completes the proof. □

We now focus on the critical points that are equalized, i.e. points U such that $\nabla L_\theta(\mathrm{U}) = 0$ and $\mathrm{diag}(\mathrm{U}^\top\mathrm{U}) = \frac{\|\mathrm{U}\|_F^2}{d_1}\mathrm{I}$.

Lemma 11.3.7. *Let $r := \mathrm{Rank}(M)$. Assume that $d_1 \leq d_0$ and $\lambda < \frac{r\lambda_r}{\sum_{i=1}^r(\lambda_i - \lambda_r)}$. Then all equalized local minima are global. All other equalized critical points are strict saddle points.*

Proof of Lemma 11.3.7. Let U be a critical point that is equalized. Furthermore, let r' be the rank of U, and $\mathrm{U} = \mathrm{W}\Sigma\mathrm{V}^\top$ be its rank-r' SVD, i.e. $\mathrm{W} \in \mathbb{R}^{d_0 \times r'}, \mathrm{V} \in \mathbb{R}^{d_1 \times r'}$ are such that $\mathrm{U}^\top\mathrm{U} = \mathrm{V}^\top\mathrm{V} = \mathrm{I}_{r'}$ and $\Sigma \in \mathbb{R}^{r' \times r'}$, is a positive definite diagonal matrix whose diagonal entries are sorted in descending order. We have:

$$\begin{aligned} &\nabla L_\theta(\mathrm{U}) = 4(\mathrm{U}\mathrm{U}^\top - \mathrm{M})\mathrm{U} + 4\lambda\mathrm{U}\mathrm{diag}(\mathrm{U}^\top\mathrm{U}) = 0 \\ \Longrightarrow\ &\mathrm{U}\mathrm{U}^\top\mathrm{U} + \frac{\lambda\|\mathrm{U}\|_F^2}{d_1}\mathrm{U} = \mathrm{M}\mathrm{U} \\ \Longrightarrow\ &\mathrm{W}\Sigma^3\mathrm{V}^\top + \frac{\lambda\|\Sigma\|_F^2}{d_1}\mathrm{W}\Sigma\mathrm{V}^\top = \mathrm{M}\mathrm{W}\Sigma\mathrm{V}^\top \\ \Longrightarrow\ &\Sigma^2 + \frac{\lambda\|\Sigma\|_F^2}{d_1}\mathrm{I} = \mathrm{W}^\top\mathrm{M}\mathrm{W} \end{aligned}$$

Since the left hand side of the above equality is diagonal, it implies that $W \in \mathbb{R}^{d_0 \times r'}$ corresponds to some r' eigenvectors of M. Let $\mathcal{E} \subseteq [d_0]$, $|\mathcal{E}| = r'$ denote the set of eigenvectors of M that are present in W. The above equality is equivalent of the following system of linear equations:

$$(I + \frac{\lambda}{d_1} 11^\top)\mathrm{diag}(\Sigma^2) = \vec{\lambda},$$

where $\vec{\lambda} = \mathrm{diag}(W^\top M W)$. The solution to the linear system of equations above is given by

$$\mathrm{diag}(\Sigma^2) = (I - \frac{\lambda}{d_1 + \lambda r'})\vec{\lambda} = \vec{\lambda} - \frac{\lambda \sum_{i=1}^{r'} \lambda_i}{d_1 + \lambda r'} 1_{r'}. \tag{11.14}$$

Thus, the set $\mathcal{E}$ belongs to one of the following categories:

0. $\mathcal{E} = [r'],\ r' > \rho$

1. $\mathcal{E} = [r'],\ r' = \rho$

2. $\mathcal{E} = [r'],\ r' < \rho$

3. $\mathcal{E} \neq [r']$

We provide a case by case analysis for the above partition here.
Case 0. [$\mathcal{E} = [r'],\ r' > \rho$]. We show that $\mathcal{E}$ cannot belong to this class, i.e. when $\mathcal{E} = [r']$, it should hold that $r' \leq \rho$. To see this, consider the r'-th linear equation in Equation (11.14):

$$\sigma_{r'}^2 = \lambda_{r'} - \frac{\lambda \sum_{i=1}^{r'} \lambda_i}{d_1 + \lambda r'}.$$

Since Rank $U = r'$, it follows that $\sigma_{r'} > 0$, which in turn implies that

$$\lambda_{r'} > \frac{\lambda \sum_{i=1}^{r'} \lambda_i}{d_1 + \lambda r'} = \frac{\lambda r' \kappa_{r'}}{d_1 + \lambda r'}.$$

It follows from maximality of ρ in Theorem 11.3.1 that $r' \leq \rho$.
Case 1. [$\mathcal{E} = [r'],\ r' = \rho$] When W corresponds to the top-ρ eigenvectors of M, we retrieve a global optimum described by Theorem 11.3.1. Therefore, all such critical points are global minima.
Case 2. [$\mathcal{E} = [r'],\ r' < \rho$] Let $W_{d_0} := [W, W_\perp]$ be a complete eigenbasis for M corresponding to eigenvalues of M in descending order, where $W_\perp \in \mathbb{R}^{d_0 \times d_0 - r'}$ constitutes a basis for the orthogonal subspace of W. For rank deficient M, $W_\perp$ contains the null-space of M, and hence eigenvectors corresponding to zero eigenvalues of M. Similarly, let $V_\perp \in \mathbb{R}^{d_1 \times d_1 - r'}$ span the orthogonal subspace of V, such that $V_{d_1} := [V, V_\perp]$ forms an orthonormal basis for $\mathbb{R}^{d_1}$. Note that both $W_\perp$ and $V_\perp$ are well-defined since $r' \leq \min\{d_0, d_1\}$. Define

$\mathbf{U}(t) = \mathbf{W}_{d_0}\Sigma'\mathbf{V}_{d_1}^\top$ where $\Sigma' \in \mathbb{R}^{d_0 \times d_1}$ is diagonal with non-zero diagonal elements given as $\sigma_i' = \sqrt{\sigma_i^2 + t^2}$ for $i \le d_1$. Observe that

$$\mathbf{U}(t)^\top \mathbf{U}(t) = \mathbf{V}\Sigma^2\mathbf{V}^\top + t^2\mathbf{V}_{d_1}^\top \mathbf{V}_{d_1} = \mathbf{U}^\top\mathbf{U} + t^2\mathbf{I}_{d_1}.$$

Thus, the parametric curve $\mathbf{U}(t)$ is equalized for all t. The population risk at $\mathbf{U}(t)$ equals:

$$\begin{aligned} L(\mathbf{U}(t)) &= \sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2 - t^2)^2 + \sum_{i=d_1+1}^{d_0}\lambda_i^2 \\ &= L(\mathbf{U}) + d_1 t^4 - 2t^2\sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2). \end{aligned}$$

Furthermore, since $\mathbf{U}(t)$ is equalized, we obtain the following form for the regularizer:

$$\begin{aligned} R(\mathbf{U}(t)) &= \frac{\lambda}{d_1}\|\mathbf{U}(t)\|_F^4 = \frac{\lambda}{d_1}\left(\|\mathbf{U}\|_F^2 + d_1 t^2\right)^2 \\ &= R(\mathbf{U}) + \lambda d_1 t^4 + 2\lambda t^2\|\mathbf{U}\|_F^2. \end{aligned}$$

Define $g(t) := L(\mathbf{U}(t)) + R(\mathbf{U}(t))$. We have that

$$g(t) = L(\mathbf{U}) + R(\mathbf{U}) + d_1 t^4 - 2t^2\sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2) + \lambda d_1 t^4 + 2\lambda t^2\|\mathbf{U}\|_F^2.$$

It is easy to verify that $g'(0) = 0$. Moreover, the second derivative of g at $t = 0$ is given as:

$$g''(0) = -4\sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2) + 4\lambda\|\mathbf{U}\|_F^2 = -4\sum_{i=1}^{d_1}\lambda_i + 4(1+\lambda)\|\mathbf{U}\|_F^2 \tag{11.15}$$

We use $\|\mathbf{U}\|_F^2 = \sum_{i=1}^{r'}\sigma_i^2$ and Equation (11.14) to arrive at

$$\|\mathbf{U}\|_F^2 = \mathrm{tr}\Sigma^2 = \sum_{i=1}^{r'}(\lambda_i - \frac{\lambda\sum_{j=1}^{r'}\lambda_j}{d_1 + \lambda r'}) = (\sum_{i=1}^{r'}\lambda_i)(1 - \frac{\lambda r'}{d_1 + \lambda r'}) = \frac{d_1\sum_{i=1}^{r'}\lambda_i}{d_1 + \lambda r'}$$

Plugging back the above equality in Equation (11.15), we get

$$g''(0) = -4\sum_{i=1}^{d_1}\lambda_i + 4\frac{d_1 + d_1\lambda}{d_1 + \lambda r'}\sum_{i=1}^{r'}\lambda_i = -4\sum_{i=r'+1}^{d_1}\lambda_i + 4\frac{(d_1 - r')\lambda}{d_1 + \lambda r'}\sum_{i=1}^{r'}\lambda_i$$

To get a sufficient condition for $\mathbf{U}$ to be a strict saddle point, it suffices that $g''(t)$ be negative at $t = 0$, i.e.

$$
\begin{aligned}
g''(0) < 0 &\implies \frac{(d_1 - r')\lambda}{d_1 + \lambda r'} \sum_{i=1}^{r'} \lambda_i < \sum_{i=r'+1}^{d_1} \lambda_i \\
&\implies \lambda < \frac{(d_1 + \lambda r') \sum_{i=r'+1}^{r} \lambda_i}{(d_1 - r') \sum_{i=1}^{r'} \lambda_i} \\
&\implies \lambda(1 - \frac{r' \sum_{i=r'+1}^{d_1} \lambda_i}{(d_1 - r') \sum_{i=1}^{r'} \lambda_i}) < \frac{d_1 \sum_{i=r'+1}^{d_1} \lambda_i}{(d_1 - r') \sum_{i=1}^{r'} \lambda_i} \\
&\implies \lambda < \frac{d_1 \sum_{i=r'+1}^{d_1} \lambda_i}{(d_1 - r') \sum_{i=1}^{r'} \lambda_i - r' \sum_{i=r'+1}^{d_1} \lambda_i} \\
&\implies \lambda < \frac{d_1 h(r')}{\sum_{i=1}^{r'} (\lambda_i - h(r'))}
\end{aligned}
$$

where $h(r') := \frac{\sum_{i=r'+1}^{d_1} \lambda_i}{d_1 - r'}$ is the average of the tail eigenvalues $\lambda_{r'+1}, \dots, \lambda_{d_1}$. It is easy to see that the right hand side is monotonically decreasing with r', since $h(r')$ monotonically decreases with r'. Hence, it suffices to make sure that λ is smaller than the right hand side for the choice of $r' = r - 1$, where $r := \mathrm{Rank}(\mathrm{M})$. That is, $\lambda < \frac{r\lambda_r}{\sum_{i=1}^{r}(\lambda_i - \lambda_r)}$.

Case 3. [$\mathcal{E} \neq [r']$] We show that all such critical points are strict saddle points. Let w' be one of the top r' eigenvectors that are missing in W. Let $j \in \mathcal{E}$ be such that w_j is not among the top r' eigenvectors of M. For any $t \in [0, 1]$, let $\mathrm{W}(t)$ be identical to W in all the columns but the j^{th} one, where $\mathrm{w}_j(t) = \sqrt{1 - t^2}\mathrm{w}_j + t\mathrm{w}'$. Note that $\mathrm{W}(t)$ is still an orthogonal matrix for all values of t. Define the parametrized curve $\mathrm{U}(t) := \mathrm{W}(t)\Sigma\mathrm{V}^\top$ for $t \in [0, 1]$ and observe that:

$$
\begin{aligned}
\|\mathrm{U} - \mathrm{U}(t)\|_F^2 &= \sigma_j^2 \|\mathrm{w}_j - \mathrm{w}_j(t)\|^2 \\
&= 2\sigma_j^2 (1 - \sqrt{1 - t^2}) \leq t^2 \operatorname{Tr} \mathrm{M}
\end{aligned}
$$

That is, for any $\epsilon > 0$, there exist a $t > 0$ such that $\mathrm{U}(t)$ belongs to the ϵ-ball around U. We show that $L_\theta(\mathrm{U}(t))$ is strictly smaller than $L_\theta(\mathrm{U})$, which means U cannot be a local minimum. Note that this construction of $\mathrm{U}(t)$ guarantees that $R(\mathrm{U}') = R(\mathrm{U})$. In particular, it is easy to see that $\mathrm{U}(t)^\top \mathrm{U}(t) = \mathrm{U}^\top \mathrm{U}$, so that $\mathrm{U}(t)$ remains equalized for all values of t. Moreover, we have that

$$
\begin{aligned}
&L_\theta(\mathrm{U}(t)) - L_\theta(\mathrm{U}) = \|\mathrm{M} - \mathrm{U}(t)\mathrm{U}(t)^\top\|_F^2 - \|\mathrm{M} - \mathrm{U}\mathrm{U}^\top\|_F^2 \\
&= -2 \operatorname{Tr}(\Sigma^2 \mathrm{W}(t)^\top \mathrm{M} \mathrm{W}(t)) + 2 \operatorname{Tr}(\Sigma^2 \mathrm{W}^\top \mathrm{M} \mathrm{W}) \\
&= -2\sigma_j^2 t^2 (\mathrm{w}_j(t)^\top \mathrm{M} \mathrm{w}_j(t) - \mathrm{w}_j^\top \mathrm{M} \mathrm{w}_j) < 0,
\end{aligned}
$$

where the last inequality follows because by construction $\mathrm{w}_j(t)^\top \mathrm{M} \mathrm{w}_j(t) > \mathrm{w}_j^\top \mathrm{M} \mathrm{w}_j$. Define $g(t) := L_\theta(\mathrm{U}(t)) = L(\mathrm{U}(t)) + R(\mathrm{U}(t))$. To see that such saddle points are non-degenerate, it suffices to show $g''(0) < 0$.

It is easy to check that the second directional derivative at the origin is given by

$$g''(0) = -4\sigma_j^2(\mathrm{w}_j(t)^\top \mathrm{M}\mathrm{w}_j(t) - \mathrm{w}_j^\top \mathrm{M}\mathrm{w}_j) < 0,$$

which completes the proof. □

11.4 Role of Parametrization

For least squares linear regression (i.e., for $k = 1$ and $\mathrm{u} = \mathrm{W}_1^\top \in \mathbb{R}^{d_0}$ in Problem 11.8), we can show that using dropout amounts to solving the following regularized problem:

$$\min_{\mathrm{u}\in\mathbb{R}^{d_0}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathrm{u}^\top \mathrm{x}_i)^2 + \lambda \mathrm{u}^\top \widehat{\mathrm{C}}\mathrm{u}.$$

All the minimizers of the above problem are solutions to the following system of linear equations $(1+\lambda)\mathrm{X}^\top\mathrm{X}\mathrm{u} = \mathrm{X}^\top\mathrm{y}$, where $\mathrm{X} = [\mathrm{x}_1, \cdots, \mathrm{x}_n]^\top \in \mathbb{R}^{n\times d_0}, \mathrm{y} = [y_1, \cdots, y_n]^\top \in \mathbb{R}^{n\times 1}$ are the design matrix and the response vector, respectively. Unlike Tikhonov regularization which yields solutions to the system of linear equations $(\mathrm{X}^\top\mathrm{X} + \lambda\mathrm{I})\mathrm{u} = \mathrm{X}^\top\mathrm{y}$ (a useful prior, discards the directions that account for small variance in data even when they exhibit good discriminability), the dropout regularizer manifests merely as a scaling of the parameters. This suggests that parametrization plays an important role in determining the nature of the resulting regularizer. However, a similar result was shown for deep linear networks [MA19] that the data dependent regularization due to dropout results in merely scaling of the parameters. At the same time, in the case of matrix sensing we see a richer class of regularizers. One potential explanation is that in the case of linear networks, we require a convolutional structure in the network to yield rich inductive biases. For instance, matrix sensing can be written as a two layer network in the following convolutional form:

$$\langle \mathrm{U}\mathrm{V}^\top, \mathrm{A}\rangle = \langle \mathrm{U}^\top, \mathrm{V}^\top\mathrm{A}^\top\rangle = \langle \mathrm{U}^\top, (\mathrm{I}\otimes\mathrm{V}^\top)\mathrm{A}^\top\rangle.$$

11.4.1 Related Work

12
SDE approximation of SGD and its implications

In Chapter 2 the analysis of convergence of gradient descent assumed that learning rate η is set small enough that the loss decreases at each iteration. Thus if η is taken to zero, the optimization still works just as well. When $\eta \to 0$ the trajectory becomes continuous and the process is called *gradient flow* (GF), given by the differential equation

$$\frac{dx}{dt} = -\nabla f(x) \quad \textit{(Gradient Flow)}.$$

If we are trying to understand how the model parameters evolve during training, gradient flow is the most natural process to analyse, as we did for instance in Chapter 8. In Chapter 2 we also studied SGD, which can be more efficient because it can use noisy estimates of the gradient from random minibatches. But if $\eta \to 0$ in SGD then the updates again are infinitesimally small and so the stochastic gradient averages out to true gradient. So as $\eta \to 0$, GD and SGD both reduce to GF.

But as we have discussed in subsequent chapters, the three algorithms can have very different behavior with respect to generalization. In other words, the trajectory matters. And yet in the limit $\eta \to 0$ they become identical! This chapter presents a way to model SGD as an process with infinitesimal steps: *stochastic differential equations, or SDEs*. These augment differential equations using noise from a stochastic process. We shall see that they yield so-called scaling rules for large-batch training; understanding how quickly SGD may escape saddle points[1] and the norm dynamics of normalized networks[2]. More recently, these techniques have been applied to study adaptive optimization algorithms, such as RMSprop and Adam, and derive scaling rules for them as well.

In this section we use x for the vector of parameters instead of our usual w, because it is more standard in SDE literature.

[1] Wenqing Hu, Chris Junchi Li, Lei Li, and Jian-Guo Liu. On the diffusion approximation of nonconvex stochastic gradient descent. *Annals of Mathematical Sciences and Applications*, 4(1), 2019

[2] Zhiyuan Li, Kaifeng Lyu, and Sanjeev Arora. Reconciling modern deep learning with traditional optimization analyses: The intrinsic learning rate. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020

12.1 Understanding gradient noise in SGD

Suppose there are m training points and let $f_i(x)$ be the loss function on example i when x denotes the model parameters. In SGD, we sample a batch of size B to compute $f_1, ..., f_B$ and use the batch gradient $\nabla f^{(B)}(x) = \frac{1}{B}\sum_{i=1}^{B} \nabla f_i(x)$ to update the parameters. [3] Hence, the noise in this estimate is $z^{(B)} = \nabla f^{(B)}(x) - \nabla f(x) = \frac{1}{B}\sum_{i=1}^{B} z_i$. Clearly, it has mean zero, implying the expectation of the stochastic gradient is the true gradient $\overline{\nabla} = \frac{1}{m}\nabla f_i(x)$.

The convergence analysis for SGD in Section 2.5.3 assumed also that there is a known upper bound on the magnitude of variance. But as mentioned, in addition to the magnitude of the noise, the nature of noise appears to be important for good generalization[4] and we now try to better understand it. If a vector random variable z has mean zero, its *covariance* matrix is the expectation $\mathbb{E}[zz^T]$. This is a measure of the "shape" of its distribution.

Problem 12.1.1. *Show that the covariance matrix of $z^{(B)}(x)$ is*

$$\Sigma^{(B)}(x) = \frac{1}{B}\mathop{\mathbb{E}}_{i}(\nabla f_i(x) - \overline{\nabla})(\nabla f_i(x) - \overline{\nabla})^T.$$

The noise $z^{(B)}$ is zero-mean, but its covariance $\Sigma^{(B)}(x)$ depends upon the current parameters and scales inversely with B. To highlight this we use $\Sigma(x)$ for the covariance with $B = 1$, and thus $\Sigma^{(B)}(x) = \frac{1}{B}\Sigma(x)$. We abstract this formula for gradient which implicitly assumes a loss function, and highlights the importance of both the shape and the scale of the noise.

Definition 12.1.2 (Noisy Gradient Oracle with Scale)**.** *A noisy gradient oracle with scale parameter $\sigma > 0$ (NGOS) takes a parameter x and returns a stochastic gradient $g = \nabla f(x) + \sigma z$, where z is drawn from a mean-zero distribution $\mathcal{Z}_\sigma(x)$ with covariance $\Sigma(x)$. $\mathcal{Z}_\sigma(x)$ can change with σ, but $\Sigma(x)$ must remain fixed.*

Improvements in multi-GPU parallelism have encouraged practitioners try large batch sizes: if the architecture can handle batch size B, then training time (keeping number of epochs constant) scales as $1/B$. This is an attractive idea but runs into the trouble that generalization error rises with B. Clearly, the magnitude of the noise covariance plays an important role in generalization just as shape does. The following was used in [5] to raise the effective scale of the noise [6] and it turned out to preserve generalization error for fairly large batch sizes (although it also fails when the batch size gets too large). Later we will mathematically justify it.

Definition 12.1.3 (Linear Scaling Rule)**.** *When running SGD with batch size $B' = \kappa B$, use learning rate $\eta' = \kappa\eta$.*

[3] In practice the training data is randomly partitioned into batches for each epoch, instead of the statistically correct method of drawing a fresh random batch for each gradient estimation.

[4] For instance, computing the full gradient and adding uniform Gaussian noise to it does not have the same beneficial effect as the noise in SGD.
Bin Shi, Weijie J Su, and Michael I Jordan. On learning rates and schrödinger operators. *arXiv preprint arXiv:2004.06977*, 2020; and Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017

[5] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017

[6] The linear scaling rule seems specific to SGD and does not apply to, say, adaptive gradient methods. It also fails for extremely large batch sizes, which has also been studied mathematically.

12.1.1 *Motivating example: Loss with Fixed Gradient*

We now consider a simple case where the gradient is fixed at $\bar{g}$ for every time step and the NGOS has isotropic noise: $g_k \sim \mathcal{N}(\bar{g}, \sigma^2 I)$. Then, after k steps of SGD, $x_k \sim \mathcal{N}(-\eta\bar{g}k, \eta^2\sigma^2 k)$. If there is a continuous approximation for the SGD trajectory, then x_k must be able to be approximated by the value of a continuous trajectory at a fixed time t independent of σ (i.e., regardless of batch size). To prevent the distribution of x_k from changing with σ, we must adjust η and k so that $\eta \sim 1/\sigma^2$ and $k \sim 1/\eta$. The first observation yields the linear scaling rule, which can be seen by noting that $\sigma \sim 1/\sqrt{B}$. The latter observation motivates a continuous time-scaling of $t \sim k\eta$. We note, however, that the scaling rule and the continuous time scale can only be made rigorous by formally showing the quality of a corresponding SDE formulation, as in Theorem 12.3.4. In order to proceed in a more rigorous fashion, we must now precisely describe what it means for a continuous trajectory to approximate a discrete one.

12.2 *Stochastic processes: Informal Treatment*

Stochastic processes can be thought of as the continuous-time version of a random walk.

Definition 12.2.1 (Lévy process). *A stochastic process $W = \{W_t : t \geq 0\}$ is a Lévy process if it satisfies the following properties.* [7]

1. $W_0 = 0$ *almost surely*
2. *Continuity: For any $\epsilon > 0$ and $t \geq 0$, $\lim_{h\to 0} \Pr[|W_{t+h} - W_t| > \epsilon] = 0$.*
3. *Stationary increments: For $s < t$, the distribution of $W_t - W_s$ is equal to the distribution of W_{t-s}.*
4. *Independent increments: for every $t > 0$, future increments $W_{t+\delta} - W_t$ for $\delta \geq 0$ are independent of past values W_s for $s \leq t$.*
5. *W_t is continuous in t.*[8]

Definition 12.2.2 (Wiener process). [9] *A Wiener process in $\Re^d$ is a Lévy process that has Gaussian increments: $W_{t+\delta} - W_t \sim \mathcal{N}(0, \delta I_{d\times d})$ for $\delta \geq 0$.*

Note that the trajectory defined by a single run of these processes is not differentiable in the normal sense. *Ito Calculus* gives a way to define a derivative of sorts, denoted, dW_t, whose integral from time 0 through T is W_T. We omit details, since those will not be needed below.

[7] Think of t as time.

[8] i.e., trajectory of W_t has no discontinuities.

[9] Wiener process is also called *Brownian motion*, used by Einstein to explain the observed movement of tiny particles suspended in a fluid. The overall trajectory results from very tiny movements in random directions due to collisions with molecules.

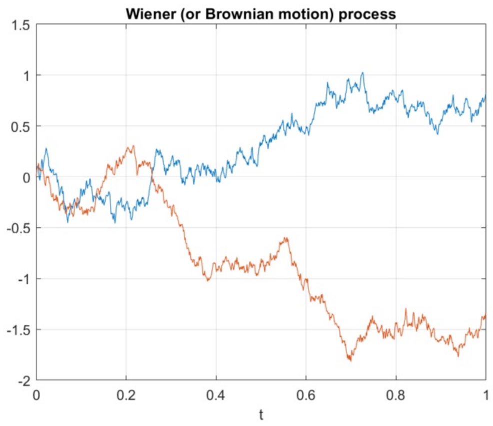


Figure 12.1: Two trajectories generated by two runs of a one-dimensional Wiener process from the same starting point. They make independent random moves and quickly diverge.

A *Stochastic Differential Equation* has the form

$$dx_t = \mu(x_t, t)dt + \sigma(x_t, t)dW_t, \quad (SDE) \tag{12.1}$$

where dW_t denotes a Weiner process and $\mu()$ and $\sigma()$ depend upon current time as well as location x_t. The heuristic interpretation is as follows:
In a small time interval $[t, t+\delta]$ the process moves by a random amount according to he Gaussian of mean $\mu(x_t, t)\delta$ and covariance matrix $\sigma(x_t, t)^2\delta I$.

Example 12.2.3 (Time change). *Suppose we change the scale of time, so that the new time τ is t/a. How does this change the equation? Using the above intuition, it becomes*

$$dx_\tau = a\mu(x_\tau, \tau)d\tau + \sqrt{a}\sigma(x_\tau, \tau)dW_\tau.$$

The fundamental reason is that the Weiner process (being a geometric random walk) only goes a distance proportional to $\sqrt{\delta}$ in time δ.

12.2.1 SDEs and SGD

The simplest SDE to model for SGD on loss $f()$ is

$$dx_t = -\eta\nabla f(x_t) + \eta\sigma dW_t \tag{12.2}$$

where dW_t is the standard Weiner process and η, σ are fixed constants. This is modeling noise in batch gradients as a uniform Gaussian[10]. A more realistic SDE for the NGOS in Definition 12.1.2 is

$$dx_t = -\eta\nabla f(x) + \eta\sigma\Sigma(x)^{1/2}dW_t \tag{12.3}$$

By the reasoning of Example 12.2.3 this is equivalent to

[10] Bin Shi, Weijie J Su, and Michael I Jordan. On learning rates and schrödinger operators. *arXiv preprint arXiv:2004.06977*, 2020; and Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017

$$dx_t = -\nabla f(x) + \sqrt{\eta}\sigma\Sigma(x)^{1/2} dW_t \quad \text{(Canonical SDE for SGD)} \quad (12.4)$$

which is a bit nicer (albeit a bit disconcerting at first sight) because the learning rate η does not appear before the gradient; only in the noise term.

Problem 12.2.4. *Define the loss function $f(x) = x^2 + 9$, and let $f_1(x) = (x-3)^2$, $f_2(x) = (x+3)^2$, and $f_3(x) = x^2 + 9$. Each iteration of SGD will uniformly sample f_1, f_2, or f_3 and use the corresponding gradient to compute the next iterate with learning rate η: $x_{k+1} = x_k - \eta\nabla f_i(x)$, where $i \sim \mathcal{U}(\{1,2,3\})$. Write down the precise SDE approximation for the trajectory SGD takes in this setting.*

Note that we have defined the canonical SDE approximation heuristically; we have not shown that this continuous approximation actually tracks the corresponding SGD. Nevertheless, we can now give an informal justification for LSR, which we later try to make a bit more rigorous in Theorem 12.3.5.

Informal justification for Linear Scaling Rule: [11]*Canonical SDE captures generalization properties. If we simultaneously scale the batch size B and learning rate η in SGD by a factor κ then the noise scale σ changes by $1/\sqrt{\kappa}$ and thus the Canonical SDE does not change.*

12.3 *Notion of closeness between stochastic processes*

SGD and the corresponding SDE in (12.4) are both stochastic processes, one discrete and the other continuous. Each induces a distribution over trajectories in the parameter space. We will use k for discrete time steps and t for continuous time. By our above discussion, if the SDE trajectory evolves for time T then this corresponds to $K = \lfloor T/\eta_e \rfloor$ discrete steps in the SGD. [12] Corresponding trajectories involve sequences of parameter vectors, denote $\{x_k^{\eta_e}\}_{k=0}^K$ for SGD and by $\{X\}_T$ for SDE[13].

What does it mean to say that two stochastic processes are close? We need formulations of a *distance* between distributions over the parameter vectors that they induce (as in Chapters 14 and 16). A common way to measure closeness is to compare difference in expectations of certain *test functions* on the two distributions[14]. For example a natural test function in our context is *test error* of the trained net.

Definition 12.3.1 (Distance between Distributions)**.** *For a function class F, define the distance between distributions $\{\widetilde{X}_k\}$ and $\{x_k\}$ as*

$$d_F(\{x_k\},\{\widetilde{X}_k\}) = \sup_{f \in F} | \mathop{\mathbb{E}}_{X \sim \{\widetilde{X}_k\}} f(X) - \mathop{\mathbb{E}}_{x \sim \{x_k\}} f(x)|$$

[11] S Jastrzębski, Z Kenton, D Arpit, N Ballas, A Fischer, Y Bengio, and A Storkey. Three Factors Influencing Minima in SGD. *ICANN*, 2018

[12] Although $\eta_e = \eta$ for SGD, we use η_e instead of η because a different optimization algorithm may require a different continuous time scaling (e.g., Section 12.4).

[13] The set notation captures that we are discussing the family of stochastic trajectories driven by random seeds for a fixed choice of η_e, but we interchangeably discuss the family and a single trajectory without loss of generality.

[14] The *discriminator net* in Chapter 16 is an example of a test function, and we saw there that classes of test functions can define transportations metrics on the space of distributions.

Completely general test functions do not make sense in such contexts, because they can magnify a negligible difference in distribution to a large difference in expectation. To prevent this we restrict the function class to have at most polynomial growth[15]. Class G of continuous functions $\mathbb{R}^d \to \mathbb{R}$ has *polynomial growth* if $\forall g \in G$ there exist positive integers $\kappa_1, \kappa_2 > 0$ such that for all $x \in \mathbb{R}^d$, $|g(x)| \leq \kappa_1(1 + |x|^{2\kappa_2})$. For $\alpha \in \mathbb{N}^+$, we denote by G^α the set of α-times continuously differentiable functions g where all partial derivatives of the form $\frac{\partial^{\overline{\alpha}} g}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}$ s.t. $\sum_{i=1}^d \alpha_i = \overline{\alpha} \leq \alpha$, are also in G. We can extend the standard definition of a distance between the positional distributions to the distributions of the entire trajectory by taking the maximum over the positional distances.

Definition 12.3.2 (Trajectory Distance)**.** *The trajectory distance over a finite number of steps $K > 0$ between a discrete trajectory $\{x\}_K$ and the corresponding rescaled continuous trajectory $\{\widetilde{X}\}_K$ under a class of test functions G is*

$$D_G(\{x\}_K, \{\widetilde{X}\}_K, K) = \max_{k=0,\ldots,K} d_G(\{x_k\}, \{\widetilde{X}_k\})$$

We can thus define a notion of *weak approximation*, which guarantees an upper bound on the maximum distinguishing expectation for a class of test functions with at most polynomial growth.

Definition 12.3.3 (Order-α Weak Approximation)**.** *We say $\{X\}_t$ and $\{x\}_k$ are order-α weak approximations*[16] *of each other if for every test function $g \in G^2$, there exists a constant $C > 0$ independent of η_e such that*

$$D_G(\{x\}_k, \{\widetilde{X}\}_k, T) \leq C\eta_e^\alpha$$

12.3.1 Formal Approximation

Now we explain in what sense

Theorem 12.3.4 (SDE is an order-1 weak approximation of SGD)**.** *Assume the NGOS and loss function f satisfy*

1. *$\nabla f(x)$ is Lipschitz and C^∞-smooth*
2. *All partial derivatives of ∇f and $\Sigma^{1/2}$ up to and including the 4th order have polynomial growth*
3. *Low skewness: there exists a function $K(x)$ of polynomial growth independent of σ such that $|\mathbb{E}_{z \sim \mathcal{Z}_\sigma(x)}[z^{\otimes 3}]| \leq K(x)/\sigma$ for all $x \in \mathbb{R}^d$ and all noise scales σ.*

[15] Test functions relevant to deep learning, such as generalization error, are probably not polynomial growth on the entire space of parameter vectors. But when we apply the definition to measure closeness of SDE and SGD, we are only interested in parameter vectors occuring on training trajectories, where the generalization error may be better behaved.

[16] Qianxiao Li, Cheng Tai, and E Weinan. Stochastic modified equations and dynamics of stochastic gradient algorithms i: Mathematical foundations. *J. Mach. Learn. Res.*, 20:40–1, 2019

4. *Bounded moments: for all integers $m \geq 1$ and all noise scales σ, there exists a constant C_{2m} independent of σ such that $\mathbb{E}_{z\sim\mathcal{Z}_\sigma(x)}[\|z\|_2^{2m}]^{\frac{1}{2m}} \leq C_{2m}(1+\|x\|_2)$ for all $x \in \mathbb{R}^d$.*

Let $\{x\}_K$ be a family of discrete SGD trajectories with learning rate η and $\{X\}_T$ be the corresponding family of SDE trajectories given by Equation (12.4). Then, $\{x\}_K$ and $\{X\}_T$ are order-1 weak approximations (Definition 12.3.3) of each other for any $T > 0$ and with $\eta_e = \eta$.[17]

Normalized networks can violate the Lipschitzness condition on the gradient because the derivatives are unbounded, but if the trajectory is bounded away from the origin and infinity, then the condition is still satisfied. The low skewness condition requires the NGOS to have a small third-order moment, and the bounded moments condition ensures the NGOS is not heavy-tailed. Together, these two conditions allow the stochastic noise to be modeled by a Wiener process. The above theorem can be extended to show the validity of LSR.

Theorem 12.3.5 (Validity of Linear Scaling Rule)**.** *Let $\{x\}_K^{(B)}$ be a family of discrete SGD trajectories with batch size B and learning rate η and $\{x\}_{\lfloor K/\kappa\rfloor}^{(\kappa B)}$ be the family with batch size κB and learning rate $\kappa\eta$. Furthermore, define the time-rescaled discrete trajectory $\{\tilde{x}\}_K^{(\kappa B)}$ where $\{\tilde{x}_k\}^{(\kappa B)} = \{x_{\lfloor k/\kappa\rfloor}\}^{(\kappa B)}$. Then, if $\{x\}_K^{(B)}$ and $\{\tilde{x}\}_K^{(\kappa B)}$ have the same initial condition, for any g with at most polynomial growth and any number of time steps $K > 0$,*

$$M_g(\{x\}_K^{(B)}, \{\tilde{x}\}_K^{(\kappa B)}, T) = C(1+\kappa)\eta$$

Proof. The linearity of covariance implies that scaling the batch size by κ only modifies the NGOS by scaling σ by $1/\sqrt{\kappa}$. Therefore, the SDE is unchanged when modifying the hyperparameters according to LSR. The weak approximation of the SDE to SGD is in terms of η, and since η is scaled by κ in LSR, the same method gives an upper bound of $C\kappa\eta$. We also account for the case when $\kappa < 1$ and hence get a bound of $C(1+\kappa)\eta$. □

Through the proof mechanism, we see that the linear scaling rule holds when the SDE approximation does. It is also possible for LSR to hold when the Itô SDE approximation fails (e.g., when the noise distribution violates the Gaussian-like assumption[18]). If $(1+\kappa)$ is treated as a constant, then we recover the same weak approximation as before. When κ becomes large, the bound becomes loose, and indeed, in practice, we observe that the linear scaling rule breaks for very large batches.[19]

[17] Qianxiao Li, Cheng Tai, and E Weinan. Stochastic modified equations and dynamics of stochastic gradient algorithms i: Mathematical foundations. *J. Mach. Learn. Res.*, 20:40–1, 2019

[18] Zhiyuan Li, Sadhika Malladi, and Sanjeev Arora. On the validity of modeling sgd with stochastic differential equations (sdes). *Advances in Neural Information Processing Systems*, 34, 2021

[19] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017; and Zhiyuan Li, Sadhika Malladi, and Sanjeev Arora. On the validity of modeling sgd with stochastic differential equations (sdes). *Advances in Neural Information Processing Systems*, 34, 2021

12.3.2 Proof Sketch

For ease of notation, we drop the set notation used to describe the positional distributions of the two trajectories and instead use x_k and $\widetilde{X}_k$. We follow two broad steps to show that the weak approximation in Theorem 12.3.4 holds. First, we divide a T-length time interval into a series of one-step intervals. We define $x_k(x, k_0)$ as the value of the discrete trajectory at time k with initial condition $x_{k_0} = x$ and do the same for X_t and $\widetilde{X}_k$. Under this notation, $\widetilde{X}_k(x_k, k) = x_k$ (i.e., SGD after k steps), and $\widetilde{X}_k(x_0, 0) = \widetilde{X}_k = X_{k\eta_e}$ (i.e., the SDE after $k\eta_e$ continuous time). We start from the SGD trajectory and sequentially replace each interval with the SDE trajectory starting from the corresponding initial condition, as shown in Figure 12.2.

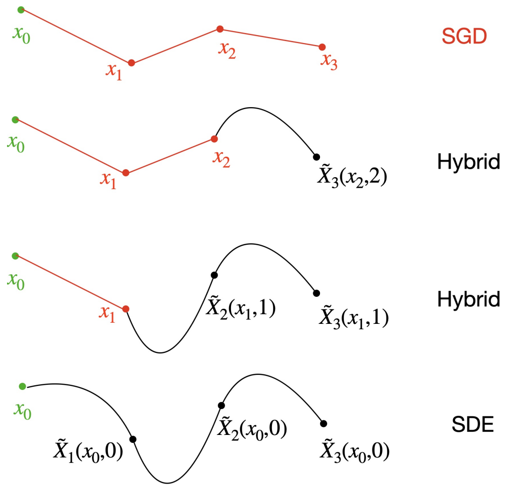


Figure 12.2: Example hybrid trajectories interpolating between SGD and SDE.

These hybrid trajectories yield the following error decomposition for any $1 \leq k \leq K$.

$$|\mathbb{E}[g(x_k)] - \mathbb{E}[g(\widetilde{X}_k)]| \leq \sum_{j=0}^{k-1} \left| \mathbb{E}[g(\widetilde{X}_k(x_{j+1}, j+1))] - \mathbb{E}[g(\widetilde{X}_k(x_j, j))] \right|$$

Problem 12.3.6. *Show that the above error decomposition holds.*

As such, the error over the entire time interval is related to the sum of the single-step errors.

The second step is to show the single-step error of the approximation is sufficiently small through Taylor expansion. Define the single-step movements with initial condition x for the discrete trajectory as $\Delta(x) = x_1 - x$ and for the continuous trajectory $\widetilde{\Delta}(x) = \widetilde{X}_1 - x$. The Taylor expansion of the discrete trajectory is straightforward:

$$\mathbb{E}[g(x + \Delta)] = g(x) + \langle \mathbb{E}[\Delta], \nabla g(x) \rangle + \mathbb{E}\left[\left\langle \frac{\Delta\Delta^\top}{2}, \nabla^2 g(x) \right\rangle \right] + \dots$$

To Taylor expand $g(x+\widetilde{\Delta})$, we introduce a key technical tool from stochastic calculus, *Itô's Lemma*, which is also known as the stochastic counterpart to the standard chain rule.

Definition 12.3.7 (Itô's Lemma)**.** *For a general Itô SDE* $\mathrm{d}X_t = b(X_t)\mathrm{d}t + \sigma(X_t)\mathrm{d}W_t$*, where* W_t *is a Wiener process, and a twice differentiable function* $h : \mathbb{R}^d \to \mathbb{R}$*,*

$$\mathrm{d}h(X_t) = \langle \nabla h(X_t), b(X_t) \rangle \mathrm{d}t + \langle \nabla h(X_t), \sigma(X_t)\mathrm{d}W_t \rangle + \frac{1}{2}\operatorname{Tr}[\nabla^2 h(X_t)\sigma^\top(X_t)\sigma(X_t)]\mathrm{d}t$$

The first two terms are the standard calculus chain rule, and the last term is a correction term to account for stochasticity. We omit a complete calculation of the stochastic Taylor expansion, but we note that the NGOS conditions are exploited in this step to show that the higher order terms in both Taylor expansions are small.

Now, we can compare the Taylor expansions to show that the single-step approximation error is $O(\eta_e^2)$, and since there are T/η_e intervals, the approximation error over a finite interval of time T is $O(\eta_e)$, as desired.

12.4 *Stochastic Variance Amplified Gradient (SVAG)*

The approximation error bound in Definition 12.3.3 relies on a small learning rate. However, real-life deep networks are often trained with larger learning rates, especially when following LSR and using a large batch size, so it is unclear if the SDE approximation is valid for practical settings. Directly simulating the SDE (e.g., through a standard discretization method, like Euler-Maruyama) is computationally intractable, because it requires computing the gradient covariance $\Sigma(X_t)$ and the full gradient $\nabla f(X_t)$ repeatedly for fine-grained intervals. In this section, we discuss a computationally efficient simulation of the SDE: SVAG.[20]

Definition 12.4.1 (SVAG Algorithm)**.** *For a given NGOS* g*, learning rate* η*, and a chosen hyperparameter* $\ell > 0$*, the SVAG algorithm computes the stochastic gradient as*

$$\widehat{g} = \frac{1+\sqrt{2\ell-1}}{2} g_{\gamma_1} + \frac{1-\sqrt{2\ell-1}}{2} g_{\gamma_2}$$

where g_γ *indicates a stochastic gradient sampled with* γ *as the random seed, and uses learning rate* η/ℓ *with the same update rule as SGD.*

The SVAG algorithm can be implemented by sampling two batches and computing a weighted average of the losses as above. Because the learning rate is scaled by $1/\ell$, we must run SVAG for ℓ steps in order to approximate the SDE value at η continuous time (i.e., the

[20] Zhiyuan Li, Sadhika Malladi, and Sanjeev Arora. On the validity of modeling sgd with stochastic differential equations (sdes). *Advances in Neural Information Processing Systems*, 34, 2021

SDE value corresponding to a single discrete step of SGD). We can formally show that the SVAG trajectory is a weak approximation of the SDE for SGD with $\eta_e = \eta/\ell$.

Theorem 12.4.2 (SVAG algorithm approximates SDE)**.** *Let $\{X\}_T$ be the SDE for SGD with hyperparameter η, and let $\{x\}_K$ be the analogous SVAG trajectory with hyperparameter ℓ. Furthermore, define $\{\widetilde{X}\}_K$ such that $\{\widetilde{X}_k\} = \{X_{k\eta/\ell}\}$ and set $\eta_e = \eta/\ell$. Assume conditions 1, 2, and 4 hold from Theorem 12.3.4. Then, for any test function $g \in G^4$ and finite time interval $T > 0$:*

$$D_G(\{x\}_K, \{\widetilde{X}\}_K, T) \leq C\eta/\ell$$

Proof. The proof relies on showing that if conditions 1, 2, and 4 hold from Theorem 12.3.4, then applying the SVAG algorithm results in an NGOS that satisfies condition 3. With all the conditions satisfied, one can regard the SVAG algorithm as SGD with a smaller learning rate and the guarantee that the NGOS satisfies the Gaussian-like and non-heavy-tailed assumptions on the noise distribution. Hence, we can directly apply the standard approximation theorem between SGD and the corresponding SDE (i.e., Theorem 12.3.4) to conclude that SVAG is an order-1 weak approximation of the SDE for SGD. □

We note here that the bound is η/ℓ, so it can be made small for a fixed η by increasing ℓ. Increasing ℓ requires taking more gradient steps to simulate the SDE, but it was found that the SVAG trajectory seems to converge and often match the SDE for computationally tractable values of ℓ.[21]

Problem 12.4.3. *Let $\widehat{\mathcal{Z}}_{\ell\sigma}(x)$ be the distribution of*

$$\widehat{z} = \frac{1}{\ell}\left(\frac{1+\sqrt{2\ell-1}}{2}z_1 + \frac{1-\sqrt{2\ell-1}}{2}z_2\right)$$

Let $\widehat{g}$ be defined as in Definition 12.4.1. Show that $\widehat{g}$ has the same distribution as $\nabla f(x) + \ell\sigma\widehat{z}$.

[21] Zhiyuan Li, Sadhika Malladi, and Sanjeev Arora. On the validity of modeling sgd with stochastic differential equations (sdes). *Advances in Neural Information Processing Systems*, 34, 2021

13
Effect of Normalization in Deep Learning

Around 2014, efforts to build upon new deep learning successes like AlexNet were stymied by the inability to make nets deeper. Training was too finicky, often failing to lower the loss very much. Ioffe and Szegedy [1] introduced *Batch Normalization*, a form of normalizing the layer parameters, which they found to make training 10x faster, and improved generalization as well. Since then other related methods have been invented, including *Layer Normalization* [2] and *Group Normalization* [3]. Today most deep architectures utilize some form of normalization.

In this chapter you will quickly note –e.g., Theorem 13.3.1— that normalization causes modern training to be fairly incompatible with traditional analyses of optimization that were surveyed in Chapter 2 and other chapters. The chapter introduces new analyses that take normalization into account. The key new mathematical notion is *scale invariance*.

13.1 *Warmup Example: How Normalization Helps Optimization*

Since deep nets are believed to be very over-parametrized for the tasks they are being used for, the net can in principle implement the desired input-output behavior in multiple ways. Let's consider a simple scenario with 1-dimensional non-separable dataset $\{(x_i, y_i) : i = 1, \ldots, n\}$ where $x_i \in \Re, y_i \in \{+1, -1\}$. The standard logistic loss $\widehat{\ell}(W)$ would be $\sum_i \log(1 + \exp(-Wx_iy_i))$. Suppose we overparametrize it, allowing k variables $(w_1, \ldots, w_k)$ and

$$\ell(w_1, \ldots, w_k) = \widehat{l}\left(\prod_{i=1}^{k} w_i\right) = \sum_i \log(1 + \exp(-x_iy_i \prod_{i=1}^{k} w_i)).$$

This is logically equivalent to the standard loss, but not equivalent with respect to behavior of gradient descent. Whereas GD quickly optimizes the original loss using a fixed learning rate, here the following happens.

[1] S Ioffe and C Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015

[2] J Ba, J R Kiros, and G E Hinton. Layer normalization. *NeurIPS*, 2016

[3] Y Wu and K He. Group Normalization. *ECCV*, 2017

Problem 13.1.1. *Suppose at initialization, all w_i's are the same, $\prod_i w_i = W_0$ and k is even. Show that if learning rate $\eta > \frac{2}{|\nabla \widehat{l}(W_0)|} |W_0|^{1/k-1}$ and $W_0 > W^*$ where $W^* > 0$ is the minimizer of $\widehat{l}$, then $\prod_i w_i$ will monotonically increase (i.e., explode).*

This example (also see the less trivial Example 1.2 in [4]) illustrates that making nets deep can have the effect of producing big numbers in the gradient, which can complicate training.

Definition 13.1.2 (Degree of homogeneity). *A function $f : \Re^d \to \Re^{d'}$ has* degree of homogeneity *k if for all $c > 0$ and all x, $f(c \cdot x) = c^k f(x)$. We also call such functions k-homogeneous.*

Problem 13.1.3. *Show that if any mapping from parameters to outputs with inputs fixed, $f \colon \Re^d \to \Re^{d'}$, is computed by a feed-forward net with depth k and only ReLU gates with zero bias [5] then it has degree of homogeneity k.*

In a k-homogeneous network for a high k, small changes in parameters can lead to large swings in gradient and Hessian. Normalization schemes reduce this effect.

13.2 Normalization schemes and scale invariance

Normalization can be done in many ways, and the following variant is possibly the easiest to understand.

Layer normalization: Let a_i denote the ith coordinate of the input of some layer in a usual feed-forward deep net (possibly with convolutions) with a fixed training datapoint. Layer normalization will change this architecture by first computing $\mu = \frac{1}{H}\sum_i a_i$ and $\sigma^2 = \frac{1}{H}\sum_i (a_i - \mu)^2$. The ith coordinate of the output of Layer Normalization layer is defined as

$$\text{LayerNorm}(a)_i = \gamma_i \cdot \frac{a_i - \mu}{\sigma} + \beta_i,$$

where γ_i, β_i are learnable parameters associated with this Layer Normalization layer.

Group normalization is a generalization of Layer Normalization where the statistics μ and σ are allowed to be computed only for subgroups of the layer. *Batch normalization* is similar to Layer Normalization, except the average μ and variance σ^2 is computed at the node with respect to all datapoints in the current training batch.

In general it is not clear how to analyse optimization once the net incorporates normalization. The paper of Arora et al [6] suggests a way forward by identifying a property called *scale invariance*.

[4] Z Li, S Bhojanapalli, M Zaheer, Reddi S, and Kumar S. Robust training of neural networks using scale invariant architectures. *arxiv*, 2022

[5] In other words, ReLU(z) = $\max\{0, z\}$.

[6] S Arora, Z Li, and K Lyu. Theoretical analysis of auto rate-tuning by batch normalization. *ICLR*, 2019

Definition 13.2.1 (Scale Invariance). *A function $f : \Re^d \to \Re^{d'}$ is* scale invariant *if $f(c \cdot x) = f(x)$ for all $c > 0$.* [7]

Note that if h_1, h_2 are k-homogeneous then so are $h_1 + h_2$ and ReLU(h_1), whereas h_1/h_2 is scale-invariant.

Lemma 13.2.2. *In the above description of layer normalization, if w denotes the parameter vector for the entire network, then for each layer $l \geq 1$, the output of ReLU, $x^{(l)}$ has degree of homogeneity 1 with respect to w, where L is the total number of layers, $x^{(0)}$ is the input of the network and $x^{(l)} := ReLU(LayerNorm(W^{(l-1)}x^{(l-1)}))$ for $1 \leq l \leq L$.*

If the parameters after the last normalization, $\gamma_i^{(L-1)}, \beta_i^{(L-1)}, W^{(L)}$ are fixed during training then the function computed is scale-invariant.

Proof. The proof is by induction on the height of the layer, l. Recall that the layer starts by computing a linear functions of the output of the previous layer. Thus in the above description, the function represented by each $x_i^{(l)}$ has degree of homogeneity 1 (except for $x_i^{(0)}$, which is 0-homogeneous), as do $\mu^{(l)}$ and $\sigma^{(l)}$. The normalized value $(x_i^{(l)} - \mu^{(l)})/\sigma^{(l)}$ is then scale-invariant. However, $\gamma_i^{(l)}(x_i^{(l)} - \mu^{(l)})/\sigma^{(l)} + \beta_i^{(l)}$ is again 1-homogeneous, and it remains 1-homogeneous after passing through the ReLU. This completes the induction. We conclude that if the output of the previous layer is 1-homogeneous then so is the output of the next layer. □

A simple fix makes the network scale-invariant: randomly fix the parameters after the last normalization, $\gamma_i^{(L-1)}, \beta_i^{(L-1)}, W^{(L)}$ at the start of training. Then train as usual. By the above lemma, the training loss becomes scale-invariant with respect to network parameters. Experiments in [8] show that fixing the top layer does not hurt classification accuracy etc. While above we focused on a simple architecture, the basic idea can be modified to show scale invariance for most known deep architectures with normalization including ResNets and language models; see [9], [10].

In the rest of the chapter, we assume scale invariance while proving convergence rates for optimization.

Lemma 13.2.3 (Properties of scale-invariant functions). *If L is scale-invariant then the following hold:*

1. $\langle w, \nabla L(w) \rangle = 0$.
2. $\nabla L(c \cdot w) = \frac{1}{c} \nabla L(w)$.
3. $\nabla^2 L(c \cdot w) = \frac{1}{c^2} \nabla^2 L(w)$.

Proof. 1) follows by differentiating $L(c \cdot w) = L(w)$ with respect to c and then setting $c = 1$. 2) follows by taking gradient of $L(c \cdot w) = L(w)$ with respect to w, and (3) follows by differentiating twice. □

[7] This means its degree of homogeneity is zero.

[8] S Arora, Z Li, and K Lyu. Theoretical analysis of auto rate-tuning by batch normalization. *ICLR*, 2019

[9] Z Li and S Arora. An exponential learning rate schedule for deep learning. *ICLR*, 2019

[10] Z Li, S Bhojanapalli, M Zaheer, Reddi S, and Kumar S. Robust training of neural networks using scale invariant architectures. *arxiv*, 2022

13.3 Exponential learning rate schedules

Usually training deep nets involves careful learning rate adjustments, with the rate being reduced over the course of training. However, in past few years several exotic learning rate schedules such as *cosine* have been successfully used. This appears to be a mystery at first. However, it can be shown provably that certain learning schedules that are nonsensical in a classical viewpoint become effective in normalized nets. We describe a result of [11] that even raising the learning rate at an exponential rate (i.e., multiply η by $(1+c)$ for some $c > 0$ at each iteration) is at least as powerful as usual training.

This happens because in practice normalization is used together with *weight decay (WD)* and *momentum*. For simplicity we ignore momentum (see the above-mentioned paper for a full analysis). The basic update with LR (Learning Rate) η and WD parameter λ is as follows, where $\nabla L_t(\cdot)$ denotes gradient computed using the minibatch in the t-th iteration:

$$w_{t+1} \leftarrow (1-\eta\lambda)w_t - \eta\nabla L_t(w_t). \quad \text{(GD + WD)} \tag{13.1}$$

Now we show that there is an alternative GD-based algorithm that can achieve the same effect but whose LR increases by a multiplicative factor of $(1+\alpha)$ in each iteration. This shows exponentially increasing LR is at least as effective as the standard GD+WD training.

Theorem 13.3.1. *If training loss is scale-invariant, the effect of (13.1) for T steps can also be obtained by the following alternative protocol:*

$$\widehat{w}_{t+1} \leftarrow \widehat{w}_t - \eta_t \nabla L_t(\widehat{w}_t). \tag{13.2}$$

with learning rate at step t being $\eta_t = (1-\eta\lambda)^{-(2t+1)}$.

Proof. Let w_t denote the parameter vector after t steps of GD + WD, and $\widehat{w}_t$ the parameter vector after t steps of our alternative protocol.

We show by induction that $\widehat{w}_t = w_t/(1-\eta\lambda)^t$. [12] This holds for $t = 0$ by design. Assuming it held for t, (13.2) gives

$$\widehat{w}_{t+1} \leftarrow \frac{w_t}{(1-\eta\lambda)^t} - \frac{\eta}{(1-\eta\lambda)^{2t+1}} \nabla L_t(\frac{w_t}{(1-\eta\lambda)^t}).$$

which by Lemma 13.2.3 part 2 simplifies to $(1-\eta\lambda)^{-(t+1)}w_{t+1} \leftarrow (1-\eta\lambda)^{-(t+1)}\left((1-\eta\lambda)w_t - \eta\nabla L_t(w_t)\right)$, thus completing the induction. □

13.4 Convergence analysis for GD on Scale-Invariant Loss

Now we analyze convergence rate for GD +WD (13.1) on scale-invariant loss $L(\cdot)$. The basic iteration is

$$w_{t+1} = (1-\eta\lambda)w_t - \eta\nabla L(w_t). \tag{13.3}$$

[11] Z Li and S Arora. An exponential learning rate schedule for deep learning. *ICLR*, 2019

[12] Recall that the loss is invariant to scalings of parameter vector.

Here we denote the unit-norm vector $w/\|w\|_2$ as $\overline{w}$.

The standard convergence analysis as in Theorem 2.5.1 of Section 2.5.2 doesn't work for a couple of reasons. First, Lemma 13.2.3 part 2 shows that making the gradient norm smaller need not imply low loss or even approach to a local optimum: increasing the scale of the parameter vector reduces the gradient but does not affect loss. Second, LR cannot be set using the reciprocal of the smoothness (i.e., largest eigenvalue of the Hessian) since the smoothness becomes unbounded as the parameter vector moves toward the origin. We present the first convergence analysis for fixed LR (taken from in [13]) in this setting, which has the added benefit of showing that the scale of initialization doesn't much matter —as one would intuitively expect in the scale-invariant setting.

Definition 13.4.1. $\rho = \max_{w:\|w\|_2=1} \|\nabla^2 L(w)\|_2 = \max_w \|\nabla^2 L(\overline{w})\|_2$.

Theorem 13.4.2 (Main)**.** For $\eta\lambda < \frac{1}{2}$, there is $t \leq \frac{1}{2\lambda\eta}\left(\left|\ln \frac{\|w_0\|_2^2}{\rho\pi^2\eta}\right| + 3\right)$ such that $\|\nabla L(\overline{w}_t)\|_2^2 \leq 8\pi^4\rho^2\lambda\eta$. [14]

To understand whether the norm upper bound guaranteed by the above theorem is meaningful, we try to understand the scale of the various quantities.

Lemma 13.4.3. *1.* $\|\nabla L(w)\| \leq \pi\rho$ *for all* w *of unit* ℓ_2 *norm.*

2. $L(w) - \min_w L(w) \leq \pi^2\rho/2$ *for all* $w \neq 0$*.*

Proof. Part 1: Let w^* be any local minimizer of L on the unit sphere. Let $\gamma\colon [0,1] \to \Re^d$ be the geodesic curve on the unit sphere with $\gamma(0) = w^*$ and $\gamma(1) = w$. We know the length of s is at most π and hence and

$$\|\nabla L(\gamma(1))\|_2 = \|\int_{t=0}^{1} \nabla^2(L(\gamma(t)))\frac{d\gamma}{dt}dt\|_2 \leq \int_{t=0}^{1} \|\nabla^2(L(\gamma(t))\|_2 \|\frac{d\gamma}{dt}\|_2 dt \leq \pi\rho.$$

Part 2 follows similarly and is left as exercise.

□

Problem 13.4.4. *Prove part 2 of Lemma 13.4.3.*

Theorem 13.4.2 guarantees that the algorithm quickly finds a solution where $\|\nabla L(\overline{w})\|_2$ is at most a $O(\sqrt{\lambda\eta})$ factor of the maximum possible value on the unit sphere. This is meaningful since in practice $\lambda\eta$ is tiny, like $10^{-4} \sim 10^{-6}$.

Lemma 13.4.5. *A twice-differential scale-invariant function* $L : \Re^d \to \Re$ *with* $\rho = \max_{\|x\|_2=1} \|\nabla^2 L(x)\|_2$ *satisfies for every pair of orthogonal vectors* x, v

$$L(x+v) - L(x) \leq \langle v, \nabla L(x)\rangle + \frac{\rho\|v\|_2^2}{2\|x\|_2^2}.$$

[13] Z Li, S Bhojanapalli, M Zaheer, Reddi S, and Kumar S. Robust training of neural networks using scale invariant architectures. *arxiv*, 2022

[14] The number of iterations has only logarithmic dependence on $\|w_0\|$, highlighting how normalization makes optimization fairly robust to the scale of initialization.

Proof. Define a function $\gamma\colon [0,1] \to \Re^d$ as $\gamma(s) = x + s \cdot v$ and $F(s) := L(\gamma(s))$. Then $L(\gamma(0)) = L(x)$ and $L(\gamma(1)) = L(x+v)$. By Taylor expansion and intermediate value theorem $F(1) = F(0) + F'(0) + F''(s^*)/2$ for some $s^* \in [0,1]$. Furthermore, $F'(0) = \langle \nabla L(x), v\rangle$ and scale invariance implies:

$$F''(s^*) = \gamma'(s^*)\nabla^2(\gamma(s^*))\gamma'(s^*) \leq \frac{\rho}{\|\gamma(s^*)\|_2^2}\|\gamma'(s^*)\|_2^2.$$

The lemma now follows by noting that $\gamma'(s^*) = v$ and $\|\gamma(s^*)\|_2 \geq \|x\|_2^2$ thanks to the orthogonality. □

The next theorem lower bounds the change in loss using the norm squared of the gradient, and is analogous to similar bounds in the simpler setting of Section 2.5.2.

Theorem 13.4.6. *If w_{t+1}, w_t are as in (13.3) and $\eta\lambda \leq 1/2$ then*

$$L(w_t) - L(w_{t+1}) \geq \eta(1 - \frac{2\rho\eta}{\|w_t\|_2^2})\|\nabla L(w_t)\|_2^2.$$

Problem 13.4.7. *Prove Theorem 13.4.6 from Lemma 13.4.5. (Hint: Use $(1-\eta\lambda)w_t)$ as x and $-\eta\nabla L(w_t)$ as v.)*

As pointed out earlier, $\nabla^2 L(w)$ can blow up as w approaches the zero vector. Accordingly, the analysis has to separate out two cases depending on $\|w_0\|_2$. First we show if the initial norm is too small then it quickly becomes large enough so the argument in Lemma 13.4.9 will apply.

Lemma 13.4.8. *In any sequence of $\frac{1}{6\lambda\eta}$ successive iterations there must exist some step T where $\|w_T\|_2^2 \geq \pi^2\rho\eta$ or $\|\nabla L(\overline{w}_T)\|_2^2 \leq 8\pi^4\rho^2\lambda\eta$. Furthermore, $\|w_T\|_2^2 \leq \frac{2(\pi^2\rho\eta)^2}{\|w_0\|_2^2}$.*

Proof. So long as $\|w_t\|_2^2 \leq \pi^2\rho\eta$ and $\|\nabla L(\overline{w}_t)\|_2^2 \geq 8\pi^4\rho^2\lambda\eta$ then using Pythagoras theorem and the fact that $\nabla L(w_t)$ is perpendicular to w_t one can conclude

$$\|w_{t+1}\|_2^2 - (1-\eta\lambda)^2\|w_t\|_2^2 = \eta^2\|\nabla L(w_t)\|_2^2. \tag{13.4}$$

which yields

$$\|w_{t+1}\|_2^2 - \|w_t\|_2^2 \geq \eta^2\|\nabla L(\overline{w}_t)\|_2^2/\|w_t\|_2^2 - 2\eta\lambda\|w_t\|_2^2 \geq 6\pi^2\rho\lambda\eta^2.$$

Summing up these inequalities over t shows that the left hand side is $\|w_t\|^2 - \|w_0\|^2$, which is at most $\pi^2\rho\eta$. On the other hand, the right hand side scales linearly with t, namely $6t\pi^2\rho\lambda\eta^2$. We conclude t cannot be more than $1/(6\lambda\eta)$. So there must be a first T before this point where $\|w_T\|_2^2 > \pi^2\rho\eta \geq \|w_{T-1}\|_2^2$. Applying Pythagoras theorem again, we have

$$\begin{aligned}\|w_T\|_2^2 \leq& \|w_{T-1}\|_2^2 + \eta^2\|\nabla L(\overline{w}_{T-1})\|_2^2/\|w_{T-1}\|_2^2 \\ \leq& \pi^2\rho\eta + \eta^2\|\nabla L(\overline{w}_{T-1})\|_2^2/\|w_0\|_2^2 \\ \leq& \frac{2(\pi^2\rho\eta)^2}{\|w_0\|_2^2},\end{aligned}$$

which yields the desired upper bound on $\|w_T\|_2^2$. □

Leveraging the previous lemma we can focus on the case where initial norm large enough.

Lemma 13.4.9. *For $\eta\lambda < \frac{1}{2}$, if $\|w_0\|_2^2 > \pi^2\rho\eta$ and $T_0 = \frac{1}{2\eta\lambda}\ln\frac{2\|w_0\|_2^2}{\rho\pi^2\eta}$ then some $t < T_0$ must satisfy*

$$\|\nabla L(\overline{w}_t)\|_2^2 \leq 8\pi^4\rho^2\lambda\eta. \tag{13.5}$$

Proof. First we show there exists some $T \leq T_0$ that $\|w_T\|_2^2 \leq \pi^2\rho\eta$. Otherwise, a simple induction using (13.4) gives

$$\begin{aligned}\|w_{T_0}\|_2^2 - (1-\eta\lambda)^{2T_0}\|w_0\|_2^2 =& \sum_{t=0}^{T_0-1}\eta^2(1-\eta\lambda)^{2(T_0-t)}\|\nabla L(w_t)\|_2^2 \\ \leq& \sum_{t=0}^{T_0-1}\frac{\eta^2}{2}\|\nabla L(w_t)\|_2^2.\end{aligned}$$

Summing the basic inequality proved in Theorem 13.4.6 over $t = 0$ to $T_0 - 1$ and the assumption that $\|w_t\| \geq \pi^2\rho\eta$ together show that the right hand side is upper bounded by $\eta(L(w_0) - L(w_{T_0}))$ which is at most $\pi^2\eta\rho/2$. Finally, by choice of T_0 we have $(1-\eta\lambda)^{2T_0}\|w_0\|_2^2 < \pi^2\eta\rho/2$. Substituting in the expression of T_0, we conclude $\|w_{T_0}\|_2^2 \leq \pi^2\eta\rho$. Contradiction! So there must be a first step $T \leq T_0$ where $\|w_T\|_2^2 \leq \pi^2\eta\rho < \|w_{T-1}\|_2^2$. (Note: $T \geq 0$ since the norm exceeds $\pi^2\eta\rho$ at initialization.) Since $\|w_T\|_2 \geq (1-\eta\lambda)\|w_{T-1}\|_2$, using (13.4) we conclude that

$$\begin{aligned}\|\nabla L(\overline{w}_{T-1})\|_2^2 \leq& \eta^{-2}\left(\|w_T\|_2^2 - (1-\eta\lambda)^2\|w_{T-1}\|_2^2\right)\|w_{T-1}\|_2^2 \\ \leq& \eta^{-2}\cdot 2\lambda\eta\|w_T\|_2^2\cdot\frac{\|w_T\|_2^2}{(1-\eta\lambda)^2} \\ \leq& 8\pi^4\rho^2\lambda\eta.\end{aligned}$$

which implies the desired upper bound on $\|\nabla L(\overline{w}_{T-1})\|_2$. □

The main theorem, Theorem 13.4.2 is proved by a straightforward combination of Lemmas 13.4.8 and 13.4.9.

14
Unsupervised learning: Distribution Learning

Much of the book so far concerned supervised learning —i.e., where training dataset consists of datapoints and a label indicating which class they belong to, and the model has to learn to produce the right label given an input. This chapter is an introduction to unsupervised learning, where one has randomly sampled datapoints but no labels or classes. We survey possible goals for this form of learning, and then focus on *distribution learning*, which addresses many of these goals.

14.1 *Possible goals of unsupervised learning*

Learn hidden/latent structure of data. An example would be *Principal Component Analysis (PCA)*, concerned with finding the most important directions in the data. Other examples of structure learning can include sparse coding (aka dictionary learning) or nonnegative matrix factorization (NMF).

Learn the distribution of the data. A classic example is Pearson's 1893 contribution to theory of evolution by studying data about the crab population on Malta island. Biologists had sampled a thousand crabs in the wild, and measured 23 attributes (e.g., length, weight, etc.) for each. The presumption was that these datapoints should exhibit Gaussian distribution, but Pearson could not find a good fit to a Gaussian. He was able to show however that the distribution was actually *mixture* of two Gaussians. Thus the population consisted of two distinct species, which had diverged not too long ago in evolutionary terms.

In general, in density estimation the hypothesis is that the unlabeled dataset consists of iid samples from a fixed distribution, and model θ learns representation of some distribution $p_\theta(\cdot)$ that assigns a probability $p_\theta(x)$ to datapoint x. This is the general problem of *density estimation.*

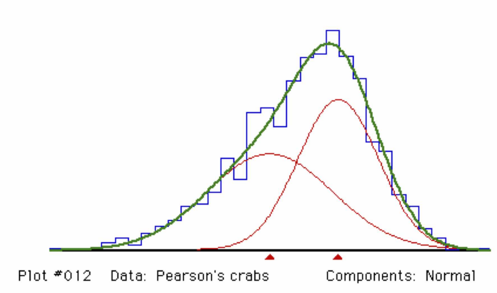


Figure 14.1: Visualization of Pearson's Crab Data as mixture of two Gaussians. (Credit: MIX homepage at McMaster University.)

One form of density estimation is to learn a *generative model*, where the learnt distribution has the form $p_\theta(h, x)$ where x is the observable (i.e., datapoint) and h consists of a vector of hidden variables, often called *latent* variables. Then the density distribution of x is $\int p_\theta(h, x)dh$. In the crab example, the distribution a mixture of Gaussians $\mathcal{N}(\mu_1, \Sigma_1), \mathcal{N}(\mu_2, \Sigma_2)$ where the first contributes ρ_1 fraction of samples and the other contributes $1 - \rho_1$ fraction. Then θ vector consists of parameters of the two Gaussians as well as ρ_1. The visible part x consists of attribute vector for a crab. Hidden vector h consists of a bit, indicating which of the two Gaussians this x was generated from, as well as the value of the gaussian random variable that generated x.

Learning good representation/featurization of data For example, the pixel representation of images may not be very useful in other tasks and one may desire a more "high level" representation that allows downstream tasks to be solved in a data-efficient way. One would hope to learn such featurization using unlabeled data.

In some settings, featurization is learnt via generative models: one assumes a data distribution $p_\theta(h, x)$ as above and the featurization of the visible samplepoint x is assumed to be the hidden variable h that was used to generate it. More precisely, the hidden variable is a sample from the conditional distribution $p(h|x)$. This view of representation learning is used in the *autoencoders* described later.

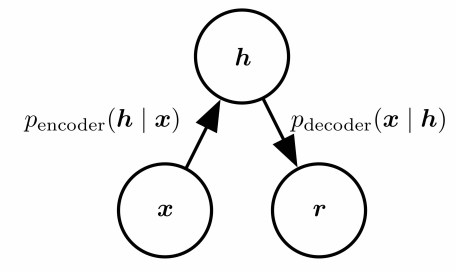

$\boldsymbol{h}$
$p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$
$p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$
$\boldsymbol{x}$
$\boldsymbol{r}$

Figure 14.2: Autoencoder defined using a density distribution $p(h, x)$, where h is the latent feature vector corresponding to visible vector x. The process of computing h given x is called "encoding" and the reverse is called "decoding." In general applying the encoder on x followed by the decoder would not give x again, since the composed transformation is a sample from a distribution.

For example, topic models are a simple probabilistic model of text generation, where x is some piece of text, and h is the proportion of specific topics ("sports," "politics" etc.). Then one could imagine that h is some short and more high-level descriptor of x.

Many techniques for density estimation —such as variational methods, described later —also give a notion of a representation: the method for learning the distribution often also come with a candidate distribution for $p(h|x)$. This why students sometimes conflate representation learning with density estimation. But many of today's approaches to representation learning do not boil down to

14.2 *Training Objective for Learning Distributions: Log Likelihood*

We wish to infer the best θ given the set S of i.i.d. samples ("evidence") from the distribution. One standard way to quantify "best" is pick θ is according to the *maximum likelihood principle*, which says that the best model is one that assigns the highest probability to the training dataset.[1]

$$\max_{\theta} \prod_{x^{(i)} \in S} p_{\theta}(x^{(i)}) \tag{14.1}$$

Because log is monotone, this is also equivalent to minimizing the *log likelihood*, which is a sum over training samples and thus similar in form to the training objectives seen so far in the book:

$$\max_{\theta} \sum_{x^{(i)} \in S} \log p_{\theta}(x^{(i)}) \quad \textit{(log likelihood)} \tag{14.2}$$

Often one uses average log likelihood per datapoint, which means dividing (14.2) by $\|S\|$.

As in supervised learning, one has to keep track of training log-likelihood in addition to generalization, and choose among models that maximize it. In general such an optimization is computationally intractable for even fairly simple settings, and variants of gradient descent are used in practice.

Of course, the more important question is how well does the trained model learn the data distribution. Clearly, we need a notion of "goodness" for unsupervised learning that is analogous to *generalization* in supervised learning.

14.2.1 *Notion of goodness for distribution learning*

The most obvious notion of generalization follows from the log likelihood objective. The notion of generalization most analogous to the

[1] The maximum likelihood principle is a philosophical stance, not a consequence of some mathematical analysis.

one in supervised learning is to evaluate the log likelihood objective on *held-out* data: reserve some of the data for testing and compare the average log likelihood of the model on training data with that on test data.

Example 14.2.1. *The log likelihood objective makes sense for fitting any parametric model to the training data. For example, it is always possible to fit a simple Gaussian distribution $\mathcal{N}(\mu, \sigma^2 I)$ to the training data in $\Re^d$. The log-likehood objective is*

$$\sum_i \frac{|x_i - \mu|^2}{\sigma^2},$$

which is minimized by setting μ to $\frac{1}{m}\sum_i x_i$ and σ^2 to $\sum_i \frac{1}{n}|x_i - \mu|^2$.

Suppose we carry this out for the distribution of real-life images. What do we learn? The mean μ will be the vector of average pixel values, and σ^2 will correspond to the average variance per pixel. Thus a random sample from the learn distribution will look like some noisy version of the average pixel.

This example also shows that matching average log-likelihood for training and held-out data is insufficient for actually learning the distribution. The gaussian model only has $d+1$ parameters and simple ϵ-cover arguments as in Chapter 5 show under fairly general conditions (such as coordinates of x_i's being bounded) that if the number of training samples is moderately high then the log-likelihood on the average test sample is similar to that on the average training sample. However, the learned distribution may be nothing like the true distribution.

This is reminiscent of the situation in supervised learning whereby a nonsensical model —e.g., one that outputs random labels—has excellent generalization as well because it has similar loss on training as well as test data.

But how can we know that log likelihood objective is in principle capable of learning the distribution? The following theorem shows so.

Theorem 14.2.2. *Given enough training data, the θ maximizing (14.2) minimizes the KL divergence $KL(P||Q)$ where P is the true distribution and Q is the learnt distribution.*

Proof. This follows from

$$\begin{aligned} KL(P||Q) &= \mathop{\mathbb{E}}_{x\sim P}[\log \frac{P(x)}{Q(x)}] \\ &= \mathop{\mathbb{E}}_{x\sim P}[\log P(x)] - \mathop{\mathbb{E}}_{x\sim P}[\log Q(x)]. \end{aligned}$$

Notice that $\mathbb{E}_{x\sim P}[\log P(x)]$ is a constant that depends only upon the data distribution, and that computing log-likelihood using iid samples from P is like estimating the second term. We conclude that given enough samples, minimizing $KL(P||Q)$ amounts to maximising log likelihood up to an additive constant. □

Note that except for low-dimensional settings, the previous Theorem does not give any meaningful bounds on the number of training datapoints needed for proper learning.

14.3 Variational method

As sketched above, we are assuming a ground truth generative model $p(x,h)$ and we are assuming we have samples of x obtained by generating pairs of (x,h) according to the ground truth and discarding the h part. The *variational method* tries to learn $p(x)$ from such samples, where "variational"in the title refers to calculus of variations. It leverages *duality*, a widespread principle in math. The idea is to maintain a distribution $q(h|x)$ as an attempt to model $p(h|x)$ and improve a certain lower bound on $p(x)$. The key fact is the following. [2]

Lemma 14.3.1 (ELBO Bound). *For any distribution $q(h|x)$*

$$\log p(x) \geq \mathbb{E}_{q(h|x)}[\log(p(x,h))] + H[q(h|x)], \tag{14.3}$$

where H is the Shannon Entropy. (Note: equality is attained when $q(h|x) = p(h|x)$.)

Proof. Since

$$KL[q(h|x)\,||\,p(h|x)] = \mathbb{E}_{q(h|x)}\left[\log \frac{q(h|x)}{p(h|x)}\right] \tag{14.4}$$

and $p(x)p(h|x) = p(x,h)$ (Bayes' Rule) we have:

$$KL[q(h|x)|p(h|x)] = \mathbb{E}_{q(h|x)}[\log \frac{q(h|x)}{p(x,h)} \cdot p(x)] \tag{14.5}$$

$$= \underbrace{\mathbb{E}_{q(h|x)}[\log(q(h|x))]}_{-H(q(h|x))} - \mathbb{E}_{q(h|x)}[\log(p(x,h))] + \mathbb{E}_{q(h|x)}[\log p(x)] \tag{14.6}$$

But since KL divergence is always nonnegative, so we get:

$$\mathbb{E}_{q(h|x)}[\log(p(x))] - \mathbb{E}_{q(h|x)}[\log(p(x,h))] - H(q(h|x)) \geq 0 \tag{14.7}$$

which leads to the desired inequality since $\log(p(x))$ is constant over $q(h|x)$ and thus $\mathbb{E}_{q(h|x)}[\log(p(x))] = p(x)$.

□

[2] See the blog post on offconvex.org by Arora and Risteski on how algorithms try to use some form of gradient descent or local improvement to improve $q(h|x)$.

14.4 Autoencoders and Variational Autoencoder (VAEs)

Autoencoders find a compressed latent representation h of the datapoint x such that x can be approximately recovered from h. They can be defined in multiple ways by chaging the formalization of what "approximate recovery" means.

In this section we formalize them using latent variable generative models. A popular instantiation of this in deep learning is *Variational Autoencoder (VAE)* [3]. As its name suggests two core classical ideas rest behind the design of VAEs: autoencoders – the original data $x \in \mathbb{R}^n$ is mapped into a high-level descriptor $z \in \mathbb{R}^d$ on a low dimensional (hopefully) meaningful manifold; variational inference – the objective to maximize is a lower bound on log-likelihood instead of the log-likelihood itself.

Recall that in density estimation we are given a data sample $x_1, \dots, x_m$ and a parametric model $p_\theta(x)$, and our goal is to maximize the log-likelihood of the data: $\max_\theta \sum_{i=1}^m \log p_\theta(x_i)$. As a variational method, VAEs use the evidence lower bound (ELBO) as a training objective instead. For any distributions p on (x,z) and q on $z|x$, ELBO is derived from the fact that $KL(q(z|x)\,||\,p(z|x)) \geq 0$

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x,z)] - \mathbb{E}_{q(z|x)}[\log q(z|x)] = ELBO \qquad (14.8)$$

where equality holds if and only if $q(z|x) \equiv p(z|x)$. In the VAE setting, the distribution $q(z|x)$ acts as the encoder, mapping a given data point x to a distribution of high-level descriptors, while $p(x,z) = p(z)p(x|z)$ acts as the decoder, reconstructing a distribution on data x given a random seed $z \sim p(z)$. Deep learning comes in play for VAEs when constructing the aforementioned encoder q and decoder p. In particular,

$$q(z|x) = \mathcal{N}(z; \mu_x, \sigma_x^2 I_d), \quad \mu_x, \sigma_x = E_\phi(x) \qquad (14.9)$$

$$p(x|z) = \mathcal{N}(x; \mu_z, \sigma_z^2 I_n), \quad \mu_z, \sigma_z = D_\theta(z), \quad p(z) = \mathcal{N}(z; 0, I_d) \qquad (14.10)$$

where E_ϕ and D_θ are the encoder and decoder neural networks parameterized by ϕ and θ respectively, μ_x, μ_z are vectors of corresponding dimensions, and σ_x, σ_z are (nonnegative) scalars. The particular choice of Gaussians is not a necessity in itself for the model and can be replaced with any other relevant distribution. However, Gaussians provide, as is often the case, computational ease and intuitive backing. The intuitive argument behind the use of Gaussian distributions is that under mild regularity conditions every distribution can be approximated (in distribution) by a mixture of Gaussians. This follows from the fact that by approximating the CDF of a distribution by step functions one obtains an approximation in distribution by a mixture

[3]

of constants, i.e. mixture of Gaussians with ≈ 0 variance. The computational ease, on the other hand, is more clearly seen in the training process of VAEs.

14.4.1 Training VAEs

As previously mentioned, the training of variational autoencoders involves maximizing the RHS of (14.8), the ELBO, over the parameters ϕ, θ under the model described by (14.9), (14.10). Given that the parametric model is based on two neural networks E_ϕ, D_θ, the objective optimization is done via gradient-based methods. Since the objective involves expectation over $q(z|x)$, computing an exact estimate of it, and consequently its gradient, is intractable so we resort to (unbiased) gradient estimators and eventually use a stochastic gradient-based optimization method (e.g. SGD).

In this section, use the notation $\mu_\phi(x), \sigma_\phi(x) = E_\phi(x)$ and $\mu_\theta(z), \sigma_\theta(z) = D_\theta(z)$ to emphasize the dependence on the parameters ϕ, θ. Given training data $x_1, \ldots, x_m \in \mathbb{R}^n$, consider an arbitrary data point $x_i, i \in [m]$ and pass it through the encoder neural network E_ϕ to obtain $\mu_\phi(x_i), \sigma_\phi(x_i)$. Next, sample s points $z_{i1}, \ldots, z_{is}$, where s is the batch size, from the distribution $q(z|x = x_i) = \mathcal{N}(z; \mu_\phi(x_i), \sigma_\phi(x_i)^2 I_d)$ via the reparameterization trick [4] by sampling $\epsilon_1, \ldots, \epsilon_s \sim \mathcal{N}(0, I_d)$ from the standard Gaussian and using the transformation $z_{ij} = \mu_\phi(x_i) + \sigma_\phi(x_i) \cdot \epsilon_j$. The reason behind the reparameterization trick is that the gradient w.r.t. parameter ϕ of an unbiased estimate of expectation over a general distribution q_ϕ is not necessarily an unbiased estimate of the gradient of expectation. This is the case, however, when the distribution q_ϕ can separate the parameter ϕ from the randomness in the distribution, i.e. it's a deterministic transformation that depends on ϕ of a parameter-less distribution. With the s i.i.d. samples from $q(z|x = x_i)$ we obtain an unbiased estimate of the objective ELBO

$$\sum_{j=1}^{s} \log p(x_i, z_{ij}) - \sum_{j=1}^{s} \log q(z_{ij}|x_i) = \sum_{j=1}^{s} [\log p(x_i|z_{ij}) + \log p(z_{ij}) - \log q(z_{ij}|x_i)] \tag{14.11}$$

Here the batch size s indicates the fundamental tradeoff between computational efficiency and accuracy in estimation. Since each of the terms in the sum in (14.11) is a Gaussian distribution, we can write the ELBO estimate explicitly in terms of the parameter-dependent $\mu_\phi(x_i), \sigma_\phi(x_i), \mu_\theta(z_{ij}), \sigma_\theta(z_{ij})$ (while skipping some constants). A single term for $j \in [s]$ is given by

$$-\frac{1}{2}\left[\frac{||x_i - \mu_\theta(z_{ij})||^2}{\sigma_\theta(z_{ij})^2} + n \log \sigma_\theta(z_{ij})^2 + ||z_{ij}||^2 - \frac{||z_{ij} - \mu_\phi(x_i)||^2}{\sigma_\phi(x_i)^2} - d \log \sigma_\phi(x_i)^2\right] \tag{14.12}$$

[4]

Notice that (14.12) is differentiable with respect to all the components $\mu_\phi(x_i), \sigma_\phi(x_i), \mu_\theta(z_{ij}), \sigma_\theta(z_{ij})$ while each of these components, being an output of a neural network with parameters ϕ or θ, is differentiable with respect to the parameters ϕ or θ. Thus, the tractable gradient of the batch sum (14.11) w.r.t. ϕ (or θ) is, *due to the reparameterization trick*, an unbiased estimate of $\nabla_\phi ELBO$ (or $\nabla_\theta ELBO$) which can be used in any stochastic gradient-based optimization algorithm to maximize the objective ELBO and train the VAE.

14.5 Normalizing Flows

The limitation of VAE is that instead of direct log likelihood, it optimizes a lower bound to it. Ideally we would want to get around this limitation while staying with a deep model with sophisticated representation capability. (The simple Gaussian fit as described at the start of the chapter also optimizes log likelihood directly but it cannot represent complicated distributions.) *Normalizing flows* can do this,

The idea in *Normalizing Flows* (Rezende and Mohamed 2015) is to make the deep net *invertible*. Specifically, it computes a function $f_\theta \colon \Re^d \to \Re^d$ that is parametrized by trainable parameter vector θ and maps image x to its representation $h = f_\theta(x)$ (note: both have the same dimension). Importantly, f is an invertible map (i.e., one-to-one and onto) and differentiable (or almost everywhere differentiable). The advantage of such a transformation is that it gives a clear connection between the probability densities of x and h. In generative models h is assumed to have some prescribed probability density $\mu(h)$, usually uniform gaussian. Via the invertible map, this translates to a density $\rho(\cdot)$ on x given by

$$\rho(x) = \mu(f(x))|\det(J_f)| \tag{14.13}$$

where J_f is the Jacobian of f namely, whose (i,j) entry is $\partial f(x)_i/\partial x_j$ and $\det(\cdot)$ denotes determinant of the matrix. This exact expression for likelihood of the training datapoints allows usual gradient-based training.

Which raises the question: how does one constrain nets to be invertible? Note that it suffices to constrain individual layers to be invertible, because the overall Jacobian is the composition of layer Jacobians. [5] To make layers invertible one often uses a variant of the following trick from the models NICE [6] and Real NVP [7]. If z^l is the input to layer l and z^{l+1} its output, then identify a special set of coordinates A in z^l and z^{l+1} and impose the restriction (where z_A denotes

[5] Since $\det(AB) = \det(A)\mathbf{det}(B)$ the determinant of the net Jacobian is the product of the determinants of the layers.

[6] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Nonlinear Independent Component Analysis. *Proc. ICLR*, 2015

[7] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Eestimation using Real NVP. *Proc. ICLR*, 2017

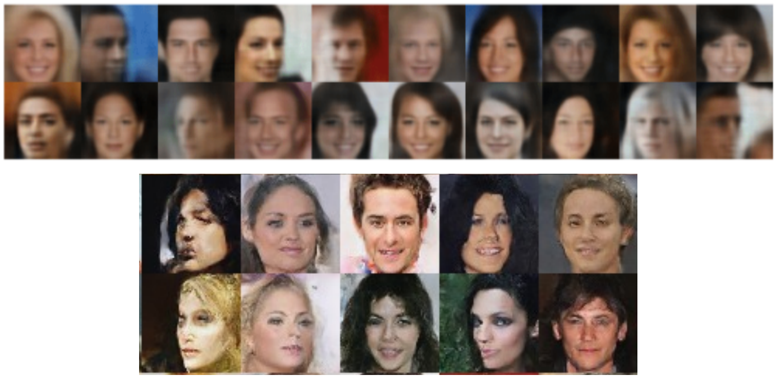


Figure 14.3: Faces in the top row were produced by a VAE based method and those in the second row by RealNVP using normalizing flows. VAE is known for producing blurry images. RealNVP's output is much better, but still has visible artifacts.

portion of z in the coordinates given by A, and B is shorthand for $\overline{A}$).

$$z_A^{l+1} = z_A^l \tag{14.14}$$
$$z_B^{l+1} = z_B^l \odot h_\theta(z_A^l) + s_\theta(z_A^l) \tag{14.15}$$

where $\odot$ denotes component-wise product and $h_\theta()$ is a function whose each output is nonnegative, with a convenient choice being to make it $\exp(r_\theta(z_A^l))$ for some other function $r_\theta()$.

This layer is invertible because given z^{l+1} one can recover z^l as follows:

$$z_A^l = z_A^{l+1} \tag{14.16}$$
$$z_B^l = (z_B^{l+1} - s_\theta(z_A^l)) \odot h_\theta(z_A^l) \tag{14.17}$$

Note that the choice of A, B can change from layer to layer, so all coordinates may get updated as they go through multiple layers. Furthermore, denoting by $z|_A$ the portion of the layer vector on coordinates A, the Jabobian for the layer mapping is lower triangular. Hence the determinant is the product of the diagonal entries.

$$\frac{\partial}{\partial z^l} z^{l+1} = \begin{pmatrix} I_{|A|\times|A|} & 0 \\ \frac{\partial}{z^l|_A} z^{l+1}|_B & \mathrm{diag}(h_\theta(z_A^l)) \end{pmatrix}$$

Normalizing flows can be extended to convolutional nets by restricting the convolutions are 1×1. Then convolutional filters just involve scalings of channel values, and the corresponding Jacobian is a diagonal nonzero matrix. Also the split of coordinates into A and B split can be done within channels as well. This is one of the ideas in GLOW model [8], which can generate better images than its predecessors.

More recent auto-regressive models such as PixelCNN are capable of producing very realistic-looking images from random seeds. However, they do not fit into the distribution learning paradigm described above so we do not discuss them here. They involve generating the

[8] Diederik P. Kingma and Prafulla Dhariwal. GLOW: Generative Flow with Invertible 1×1 convolutions. *Proc. Neurips*, 2019

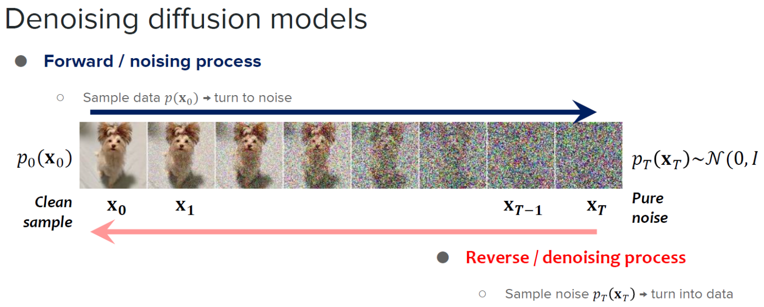
image pixel by pixel (roughly speaking) and thus do not parallelize well.

Problem 14.5.1. *Let* (z_1, z_2, z_3, z_4) *be distributed as a standard Gaussian* $\mathcal{N}(0, I)$ *in* $\mathbb{R}^4$. *Let* $f : \mathbb{R}^4 \to \mathbb{R}^4$ *be an invertible function which maps* (z_1, z_2, z_3, z_4) *to* $(z_1, z_2, e^{a_0} z_3 + a_1 z_1^2 + a_2 z_2^2, e^{b_0} z_4 + b_1 z_1^2 + b_2 z_2^2)$ *for some coefficients* $a_0, a_1, a_2, b_0, b_1, b_2 \in \mathbb{R}$. *Compute the probability density function of* $f(z_1, z_2, z_3, z_4)$.

14.6 Stable Diffusion

You may have seen AI models that generate artificial imagery given a text prompt such as “Pope Francis walking in a puffer jacket.” These are made by *diffusion models* [9], which we describe in this section, albeit without the text prompts.

Diffusion models are reminiscent of normalizing flows and autoencoders, in that they define a mapping f that transforms the set of all images to the set $\mathcal{N}(0, I)$, as well as an inverse mapping f^{-1} that maps gaussian vectors to images. The difference is that f is very trivial; just a series of noising steps. Furthermore, f^{-1} is just net custom-trained on denoising the output of f.



Denoising diffusion models

- Forward / noising process
  - Sample data $p(\mathbf{x}_0)$ → turn to noise

- Reverse / denoising process
  - Sample noise $p_T(\mathbf{x}_T)$ → turn into data

Figure 14.4: Example of noising an image and then denoising, using Diffusion Model. (Source: Binxu Wang)

The noising layers noise an an image x_0 to x_T in T steps as follows where $z_t \sim \mathcal{N}(0, I)$ and each $\alpha_t \in (0, 1)$

$$x_{t+1} = \sqrt{\alpha_t} x_t + \sqrt{1 - \alpha_t} z_t, \qquad t = 0, \ldots, T-1 \tag{14.18}$$

A simple induction shows this is equivalent to the following where $\overline{\alpha_t} = \prod_{i=1}^{t} \alpha_i$:

$$x_{t+1} = \sqrt{\overline{\alpha_t}} x_t + \sqrt{1 - \overline{\alpha_t}} z. \tag{14.19}$$

The above calculation is using the following.

Problem 14.6.1. *If* z_1, z_2 *are independent samples from* $\mathcal{N}(\mu_1, \sigma_1^2 I)$ *and* $\mathcal{N}(\mu_2, \sigma_2^2 I)$ *respectively then* $z_1 + z_2$ *is distributed as*

$$\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2 I).$$

[9]

15
Language Models (LMs)

Starting around 2020, language models (LMs) have suddenly become the most visible face of AI research. This chapter introduces the basic notions and results. The heart of it

A key concept underlying LMs is that text produced by humans is assumed to have be sampled from a probabilistic distribution, with $\Pr(w_1\ w_2\ \ldots w_i)$ the probability associated with a sequence of words $w_1\ w_2\ \ldots w_i$. Then Bayes' rule implies a factorization:

$$\Pr[w_1\ w_2\ \ldots w_{i+1}] = \Pr[w_1\ w_2\ \ldots w_i] \Pr[w_{i+1}\ |w_1 w_2 \ldots w_i] \tag{15.1}$$

This implies that to *generate* a sequence of words, we only need ability to compute $\Pr[w_1]$ and then a way to generate the $i+1$'th word given the previous i words. This, in a nutshell, is what a language model does.[1] The simplest models date back to 1950s, when computing pioneer Claude Shannon proposed various simple approximations to (15.1). The simplest, called *unigram* model, computes estimates of the probability $\Pr[w]$ by measuring empirical frequency of word w in a sufficiently large corpus, and uses the approximation $\Pr[w_{i+1}\ |w_1 w_2 \ldots w_i] = \Pr[w_{i+1}]$, which is equivalent to saying $\Pr[w_1\ w_2\ \ldots w_{i+1}] = \prod_i \Pr[w_i]$. The *bigram* model does something similar but assumes that $\Pr[w_{i+1}\ |w_1 w_2 \ldots w_i] = \Pr[w_{i+1} \mid w_i]$. That is to say, the probability of the next word depends upon the previous word, but not on any earlier words. Then one only needs to empirically estimate $\Pr[w \mid w']$ for all word pairs w, w' which can be done using a modest corpus size.

In recent decades as neural nets were applied to language modeling a key idea emerged: semantic vector of the context. The idea is that to predict the next word w_{i+1} using the sequence $w_1 w_2 \ldots w_i$ of preceding words, you compute an embedding $c_i \in \Re^D$ of the "meaning" of $w_1 w_2 \ldots w_i$. You also have a semantic embedding $v_w \in \Re^D$ of every word w. Then the distribution of the next word is described as:

$$\Pr[w_{i+1}\ |w_1 w_2 \ldots w_i] \propto \exp(v_{w_{i+1}} \cdot c_i). \tag{15.2}$$

[1] This exposition ignores other types of language models that are used to compute semantic embeddings for text-pieces. Famous ones include BERT, ERNIE, RoBERTA etc.

Since this idea has become ubiquitous in language modeling, let's understand something called *softmax*. [2] Describing the distribution via (15.2) amounts to saying that the distribution on the next word is softmax of the vector in $\Re^V$ whose $i+1$th coordinate is $v_{w_{i+1}} \cdot c_i$. Here V denotes the vocabulary size (i.e., number of distinct words). Semantic embeddings and softmax are used for defining the "next-word" distribution in today's LMs as well. The key change with modern LMs is the transformer architecture for computing the embeddings.

The goodness of a language model is computed via its *cross entropy loss*, which for a sequence of words $w_1 w_2 \dots w_t$ is:

$$\ell(M) = \sum_i \log \frac{1}{\Pr_M[w_{i+1} \mid w_1 w_2 \dots w_i]} \qquad \text{(Cross Entropy)} \qquad (15.3)$$

Models are trained by minimizing (via gradient descent on model parameters) this training loss on a text corpus, and their goodness is computed by their *test loss*—evaluating the same loss expression on a held-out text from the same corpus. Often the training corpus is so large that the model trains only once (or a few times) on each piece of text, and by the end, the test loss on held-out text is almost the same as the training loss. Thus generalization error is fairly small.

Since minimizing the above loss amounts to maximising

$$\prod_i \Pr_M[w_{i+1} \mid w_1 w_2 \dots w_i],$$

sometimes the goal of language modeling is described as trying to create a model that assigns the largest possible probability to the training corpus. This is actually a bit misleading to students. Section 15.2 explains that the real goal is to make the model learn the human distribution.

Language tasks: Decades of research in NLP has identified thousands of tasks. Some examples: *Implication:* given two pieces of text decide whether or not the first implies the other. *Sentiment* given a piece of text decide whether it has *positive, negative or neutral* sentiment. *Question-answering:* given a piece of text, answer questions related to it. *Translation:* Given a piece of text in one language, translate it into another.

For many years such tasks were difficult. Today as you know these as well as much more difficult tasks are routinely solved using large LMs. This chapter will not discuss the architecture and training details since good writeups exist on the internet. Instead it discusses the conceptual underpinnings of cross-entropy loss (Section 15.2), as well as how to train LMs to generate text that is useful or meaningful in human interactions (Sections 15.6, **??**. Then we focus on

[2] Recall that softmax is a mapping from $\Re^k$ to the simplex, namely, $\{(p_1, p_2, \dots, p_k) : p_i \geq 0, \sum_i p_i = 1\}$. For $(s_1, s_2, \dots, s_k) \in \Re^k$ define its softmax as the vector whose i'th coordinate is $\exp(s_i)/(\sum_{j=1}^k \exp(s_j))$.

15.1 Transformer Architecture

For about 15 years leading up to 2017, there was a lot of innovation in LMs with many neural architectures designed along the way. The goal of the architecture is to compute the word embeddings and context embeddings to allow prediction of the next word.

Starting 2017 these architectures have been effectively replaced in most settings with the transformer architecture, which is also used now for images, sound, genomes, and other types of data. We recommend reading an excellent introduction to transformers on Lilian Weng's blog. [3] The following question invites you to test your understanding.

Problem 15.1.1. *Consider a N-layer single-head transformer with input section length L and hidden state dimension d. For each layer l, let the input for the layer be X_l and the output be X_{l+1}, we have*

$$X_{l+1} = V_l \cdot \mathit{softmax}\left(\frac{Q_l^\top K_l}{\sqrt{d}}\right),$$

where $Q_l = W_l^q X_l$, $K_l = W_l^k X_l$, $V_l = W_l^v X_l$, and $X_l \in \mathbb{R}^{d\times L}$, $W_l^q, W_l^k, W_l^v \in \mathbb{R}^{d\times d}$. If the size of dataset is M (total number of tokens), find the asymptotic training time of each epoch in terms of M, N, L, d.
Note that here we consider the input data X_l as a collection of column vectors (each column is a data point), so the parameter matrices W's are multipled on the left of X's. In some other literatures such as the Lilian Weng's blog, X_l is a collection of row vectors (each row is a datapoint). The 2 definitions are sometimes used interchangeably.

15.2 Explanation of Cross-Entropy Loss

Now we try to understand the conceptual framework underlying cross-entropy loss (15.3). As mentioned, there is a ground-truth (i.e., humans') distribution for generating the next word, which assigns probability $p_i(w \mid w_1 w_2 \dots w_i)$ to the event that the $(i+1)$th word is w given that the previous words were $w_1 w_2 \dots w_i$. In interest of compact notation we shorten $p_i(w \mid w_1 w_2 \dots w_i)$ to $p_i(w)$, Thus the *entropy* of the $(i+1)$th word is

$$\sum_w p_i(w) \log \frac{1}{p_i(w)} \qquad \text{(ENTROPY)} \qquad (15.4)$$

This entropy is an inherent property of language, arising from many choices human writers make for the next word. Given sequence $w_1 w_2 \dots w_i$ the model has a probability distribution $q(w|w_1 w_2 \dots w_i)$ for the next word w. Extending our compact notation, we use $q_i(w)$

[3] *"Tranformer Family: Version 2.* `https://lilianweng.github.io/`. 2023

as a shorthand for this. The cross-entropy loss of the model on the $(i+1)$th word is $\log \frac{1}{q(w_{i+1})}$, which should be seen as an empirical estimate of

$$E_{w \sim p_i()}[\log \frac{1}{q(w)}] \quad \text{(EXPECTED C-E LOSS)} \tag{15.5}$$

KL divergence, also sometimes called *excess entropy*, is non-negative and defined as

$$KL(p_i||q_i) = E_{w \sim p_i()}[\log \frac{p_i(w)}{q_i(w)}] \quad \text{EXCESS ENTROPY} \tag{15.6}$$

Thus on a per-word basis we have:

$$\text{EXPECTED C-E LOSS} = \text{ENTROPY} + \text{EXCESS ENTROPY} \tag{15.7}$$

Summing over the entire held out corpus, one obtains a similar estimate for the entire corpus. One can make mild assumptions to the effect that the conditional probabilities $p_i(), q_i()$ only depend only on (say) the previous 10^3 words, whereas the held out corpus size M is much bigger, e.g., $M \gg 10^8$. So the corpus consists of a random walk of sorts, where every 10^4 words or so it switches to a different portion of the language distribution. Under such assumptions the above relationship, which holds in expectation at the word level, should hold fairly precisely at the corpus level.

To summarize, since ENTROPY of text is a constant, we can interpret ((15.7)) as follows.

Goal of Language Modeling: *The goal of language modeling is to minimize KL-divergence of the human's next-word distribution to the model's distribution.*

Note that this applies to vanilla modeling as in this chapter. In practice, training methods and even the loss function, deviate from the simple picture presented above.

15.3 Scaling Laws and Emergence

An old ambition in deep learning was to be able to simply scale up the network and train it with more data and continue to solve problems better. While this hope had worked out nicely prior to 2017, thereafter thanks to transformers it got turbocharged. The key discovery was so-called *scaling laws*.

These are empirically-derived expressions describe how test cross entropy loss on held-out data scales (in experiments) with number of model parameters (N) and size of the dataset (D) [4] [5], [6]. For Chinchilla models [7] the law is as follows:

$$L(N, D) = A + \frac{B}{N^{0.34}} + \frac{C}{D^{0.28}} \qquad A = 1.61 \quad B = 406.4 \quad C = 410.7. \tag{15.8}$$

[4]

[5]

[6] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *arXiv preprint arXiv:2102.06701*, 2021

[7] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022

Here the constants A, B, C in 15.8 hold only for the specific architecture and training strategy —even the constant A depends upon the tokenization. This description of macro behavior using two basic parameters —reminiscent of 2nd Law of Thermodynamics— will help us circumvent the need for mechanistic understanding of training. Our theory will only rely upon the general form of the equations, specifically, that the dependence is inverse polynomial in N, D. So it applies to other frameworks of training (e.g., overtrained models [8]) where scaling laws have also been found.

Emergence: Emergence refers to an interesting empirical phenomenon that as D, N are increased together then the model's performance (zero shot or few-shot) on a *broad range* of language tasks improves in a correlated way. The improvement can appear as a quick transition when D, N are plotted on a log scale (which is often the case) but it is now generally accepted that for most tasks the performance improves gradually when D, N are scaled up. Thus the term *slow emergence* is more correct. Furthermore, it is known that emergence happens at different rates for different tasks, and is often quickest for tasks where the text is plausibly close to text found in training data [9]. Plenty of tasks are known that stump current models, and they usually tend to be very different from what one would find in usual text corpora. See [WTB$^+$22**?** , SMK23] for experimental results on emergence rates of the broad range of language tasks. One might thus posit, with some justification from the above-mentioned studies, that the emergence of skills arises from training on related tasks that were implicitly solved while solving next-word prediction in the training dataset. This is indeed our starting point.

15.4 (Mis)understanding, Excess entropy, and Cloze Questions

Thinking about emergence and Scaling Laws, it is possible to get confused as follows: *"When we increase D from* 10^{11} *to* 10^{12} *then according to (15.8) this changes cross-entropy by a tiny amount. Why does it lead to big changes in macroscopic behavior?"* This section explains the flaw in this reasoning.

The flaw is that most of the loss captures merely the inherent entropy of language (the A term in (15.8)). We argue now that the model's mistakes on downstream tasks (i.e., its misunderstandings) are captured by the *excess* entropy, which as noted in Section 15.2 reduces by a constant factor each time the model is scaled up by an order of magnitude.

We illustrate using a classic example from [10], which later inspired the *Winograd Schema Challenge(WSC)* [11]:

[8] Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models, 2023

[9] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022

[10] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1971

[11] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012

`The city councilmen refused the demonstrators a permit because they feared violence.`

Here the pronoun `they` is ambiguous— grammar rules allow it to refer to either `demonstrators` or `city councilmen`. Winograd pointed out that disambiguating it (i.e., anaphora resolution) requires world knowledge that is unavailable in the text itself, namely that demonstrations can get violent, and city councilmen don't like violence.

A key idea in designing test-beds for language understanding such as WSC is the **Cloze Procedure** [12,13], popular also for testing language development in children. To test the model's understanding of `they` in this sentence, we can append a *prompt*: `Q. Who feared violence?`. This is followed by either a blank, or a choice of multiple answers: `A. city councilmen.  B. demonstrators.` For WSC examples, even though a human would be hundred percent sure of the answer, language models circa 2016 were roughly 50/50 confused between the two options.

In the above example, the human is 100% certain of the answer, which implies their entropy here is $\log 1$, namely 0. However if the model is split 50-50 between the two options this implies it has cross-entropy $\log 2$, all of which is *excess entropy*! Given the frequency of ambiguous pronouns in usual English, one concludes that a model that has not learned pronoun disambiguation will display huge excess entropy at many places in surrounding text. Thus reductions in excess entropy (which happen naturally due to scaling) will tend to squeeze out such errors. The ensuing analysis tries to make this intuition mathematically precise.

Of course, text corpora do not normally contain such artificial cloze questions. But one could imagine that the model's basic misunderstanding of the above type could, often, lead to prediction mistakes in neighboring text. Our theory in Section 15.8 will assume that cloze questions can closely capture the model's misunderstanding.

15.5 How to generate text from an LM

The title of this section may feel ridiculous, since the very definition of LM's in (15.1) involves ability to predict the next word given the preceding words. This appears to give an obvious way to generate goodtext: use the model distribution for the first word to sample a particular w_1, then sample from the distribution $\Pr[w_2|w_1]$ to generate the second word, and carry on like that. The procedure just described is called **random generation** or simply **sampling** and it actually does not produce good text[14].

The next natural idea is to **greedy:** Having generated $w_1 w_2 \ldots w_i$, generate w_{i+1} using the word that has the highest probability in the

[12]

[13] Cloze questions are multiple choice, which allows testing most language skills. They are not a good match for skills such as understanding of irony since one of the multiple choices already explains the joke.

[14] The quality of **sampling** gets better as models scale up and thus closer to the language distribution.

next position. At first sight this seems appealing since the training objective for LMs implicitly trains them to maximise the probability they assign to the training text given to them. So when generating text, why not try to generate pieces of text that are given the highest possible probability by the model? [15] The reason is that, as explained in Section 15.2, the true goal of language modeling is to minimize KL distance to the human distribution.

Greedy text looks flat and unexciting. For instance the greedy continuations of *Thanks for the dessert, it was* . . . is probably *great*, but there could be a variety of more interesting ones with lower probability, such as *exquisite*, *life-changing* etc. Human communication often veers into low-probability words. Indeed, the perplexity of text produced by the greedy method is far from that of human-generated text! A good discussion of this issue appears in [16], from where Figure 15.1 was taken.

Method	Perplexity
Human	12.38
Greedy	1.50
Beam, b=16	1.48
Stochastic Beam, b=16	19.20
Pure Sampling	22.73
Sampling, t=0.9	10.25
Top-k=40	6.88
Top-k=640	13.82
Top-k=40, t=0.7	3.48
Nucleus p=0.95	**13.13**

Figure 15.1: Perplexity of text from various generation methods. Random and Greedy are pretty bad. Nucleus Sampling with $p = 0.95$ gets closest to human. (We did not describe beam search, so please ignore those rows.)

The best methods actually *reshape* the distribution via some greedy-ish selections.

- **Top-k:** Identify the top k choices for first word, and restrict yourself to pick the first word among them (i.e., disallow picking any other word in this position). If their combined probability is p_k you do this by rescaling the probabilities of these k words by $1/p_k$ and zero-ing probabilities of all other words, and then pick from this distribution. Having picked the first word, continue similarly to pick the rest. You set k by trial and error to best match perplexity to human text.

- **Nucleus sampling (aka "top p"):** This is a softer variant of the above. Instead of making a hard decision about the number of possible choices for the first word (i.e., k), decide that you will allow k to vary but then impose hard constraint that the total probability of all the choices you will allow in the first position is p. Continue similarly for rest of the word positions, with the same

[15] Actually greedy doesn't quite maximise the probability but is an attempt in that direction.

[16] A Holtzman, J Buys, L Du, M Forbes, and Y Choi. The curious case of neural text degeneration. *ICLR*, 2020

"probability budget." You set p by trial and error to best match perplexity to human text.

15.6 Instruction tuning

As mentioned, LLMs may have a good idea about language but may not have a good understanding of participate in human conversation. For instance, if the human asks *Can you write a haiku that fits in a tweet?* The LLM may just response "yes" since it may not understand that it is expected to provide such haiku. Instruction-tuning consists of training on a dataset of (x, y) pairs where x is an instruction from a human and y is a model answer. The machine is fed such (x, y) pairs and is trained to minimize cross-entropy on the tokens in y.

At the end of this it is able to provide good answers to human questions. Let's call this distribution $\pi_{SFT}(y|x)$, where SFT stands for 'supervised fine tuning.'

15.7 Aligning LLMs with human preferences

Although instruction tuning gives LLMs the ability to answer questions, in settings where the prompts are problematic, its answers may not align with our notions of correctness, morality etc. For instance, when asked to help plot a crime, instruction-tuned LLMs will easily provide detailed instructions. Or when asked about an event that it has not seen a data, it may *hallucinate* facts about this event because a language model always has nonzero probability for all kinds of texts. Let $\mathcal{D}$ be a distribution on such problematic prompts. Think of this as giving a weighting to prompts according to how"tricky" they are.

The training dataset consists of human preference data consisting of pairs[17] $(x, y_1) \succ (x, y_2)$ where x is a problematic prompt drawn according to , y_1, y_2 are two different answers to it, and y_1 is prefered to y_2.

The famous Bradley-Terry model [18] for human preferences suggests that such pairwise rankings correspond to a probabilistic sampling from the following distribution:

$$p^*(y_1 \succ y_2 \mid x) = \sigma(r^*(y_1|x) - r^*(y_2|x)) \tag{15.9}$$

where sig is the *sigmoid* function [19] and r^* is a so-called *reward function* that maps a (prompt, response) pair (x, y) to $\Re$. The assumption is that humans have such a reward function in their heads. If you give a person two responses y_1, y_2 for prompt x then they prefer y_1 over y_2 according to the function above. [20] We express the reward $r^*(y_1 \mid x)$ using conditional notation because the reward function is only valid for comparing responses to the same prompt x. It can't be

[17] Humans tend to find it much easier to give a preference between two alternatives than to produce a ranking among a larger set of alternatives.

[18] Ralph A. Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of Paired Comparisons. *Biometrika*, 1952

[19] The sigmoid function $\sigma(t) = \frac{1}{1+e^{-t}}$ $= \frac{e^t}{1+e^t}$ maps $(-\infty, \infty)$ to $(0, 1)$.

[20] For instance, if the prompt is a question asking for help committing a horrendous crime, then most humans would strongly disapprove of a response $y|x$ that gives advice for committing the crime. This can be interpreted as as this response having extremely negative value of r^*.

used to compare rewards for a responses to two different prompts x and x'. How can we learn this (unknown) reward function $r*()$ in human heads? The trick is to

Problem 15.7.1 (Learning reward model)**.** *Suppose we have a dataset $\mathcal{D}$ consisting of (x, y_1, y_2) triples where x is a question, y_1, y_2 are two answers and human raters have indicated $(x, y_1) \succ (x, y_2)$. Show that the max-likelihood fit for a reward function r_ϕ is*

$$\mathcal{L}(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_1,y_2)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_1) - r_\phi(x, y_2))\right] \tag{15.10}$$

We want to train a parametric model $\pi_\theta(y|x)$ that acts according to the learnt reward function $r^*()$. We can approach this via the following thought experiment: Imagine that a prompt x is sampled from $\mathcal{D}$ and the model generates a response according to $\pi_\theta(y \mid x)$. Then the machine gets a *reward* $r^*(y \mid x)$. Thus a good model ought to be optimizing

$$\max \mathop{\mathbb{E}}_{x\sim\mathcal{D}, y\sim\pi(y|x)}[r^*(y \mid x)]. \tag{15.11}$$

However, this objective only involves the distribution $\mathcal{D}$ that favors "tricky" prompts. To continue to be a good language model, $\pi_\theta()$ must not wander away far from $\pi_{SFT}()$. Thus we must add a regularizer term to keep the distribution from deviating unnecessarily from $\pi_{SFT}()$.

$$\max \mathop{\mathbb{E}}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}[r^*(y \mid x)] - \beta \cdot KL(\pi_\theta(y \mid x)||\pi_{SFT}(y \mid x)||). \tag{15.12}$$

This is the essence of *Reinforcement Learning with Human Feedback* (RLHF)[21], which was for several years the prefered method to align language models with human values. We will not discuss in detail the exact optimization technique and instead move on to a better alternative below.

Problem 15.7.2. *Show that if the parameterization θ for the policy has unlimited size then the optimum solution satisfies*

$$\pi_\theta(y \mid x) = \frac{1}{Z(x)}\pi_{SFT}(y \mid x) \exp\left(\frac{1}{\beta}r^*(y \mid x)\right), \tag{15.13}$$

where $Z(x) = \sum_y \pi_{SFT}(y \mid x) \exp\left(\frac{1}{\beta}r^*(y \mid x)\right)$ is the partition function.

15.7.1 Direct Reward Optimization

The RLHF method above was often difficult to implement due to optimization issues. Just recently a method called *Direct Preference Optimization (DPO)* [22] has gained adherents because it leverages the same Bradley-Terry framework of preferences with a simpler optimization.

[21] P. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *NeurIPS*, 2017

[22] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arxiv*, 2023

The main idea is to skip the reward function and instead use (15.7.2) to express the reward function directly using the model $\pi_\theta(y \mid x)$, and directly optimizing the human preference pairs $(x, y_1) \succ (x, y_2)$. Specifically, (15.7.2) implies that the optimum model $\mathrm{p}^*(y \mid x)$ satisfies for all x

$$\Pr[(y_1 \mid x) \succ (y_2 \mid x)] = \frac{1}{1 + \exp\left(\beta \log \frac{\pi^*(y_2|x)}{\pi_{SFT}(y_2|x)} - \beta \log \frac{\pi^*(y_1|x)}{\pi_{SFT}(y_1|x)}\right)} \tag{15.14}$$

Problem 15.7.3. *(DPO) Use the above to show that the following objective captures the search for the best parametric model $\pi_\theta(y|x)$ given preferences (x, y_1, y_2)*

$$\max_\theta \mathbb{E}_{(x,y_1,y_2)} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_1 \mid x)}{\pi_{SFT}(y_1 \mid x)} - \beta \log \frac{\pi_\theta(y_2 \mid x)}{\pi_{SFT}(y_2 \mid x)}\right)\right]. \tag{15.15}$$

DPO actually attaches a KL penalty to this objective just like RLHF, and optimizes that. See the paper for implementation details and experiments.

15.8 Mathematical Framework for Skills and Emergence

We now give the main theory component of this chapter, which is a new mathematical framework for thinking about skills and how they might relate to language comprehension tasks, and also the emergence phenomenon. This is from a recent paper [23].

First, it is assumed that language comprehension involves a set of skills, though the theory will not need to know a precise list. (Scholars have discovered and named thousands of skills. Well-trained transformers have undoubtedly discovered many more that remain unnamed.) Next, the theory will assume scaling laws such as (15.8) and thus not need to reason about training and generalization. Instead, it can reason directly about the model's behavior on the test distribution, i.e., the distribution from which the training data was drawn. We assume this test distribution is structured as a long unordered list of text-pieces, each with an associated measure[24] Traditional cross-entropy loss is averaged using this associated measure.

Definition 15.8.1 (Text piece)**.** *The test corpus for the model is viewed as being divided into* text-pieces*, each consisting of C_{test} tokens. There is also a measure $\mu_2()$ on these text-pieces, with $\mu_2(t)$ denoting the measure of text-piece t. The usual cross-entropy loss is computed by weighting text-pieces with respect to this measure.*

Now we make some assumptions. We assume that the model's "comprehension" of a text piece is testable via suitable cloze ques-

[23] Sanjeev Arora and Anirudh Goyal. A theory for emergence of complex skills in language models. *arXiv preprint arXiv:2307.15936*, 2023

[24] Text-pieces should be thought of as having a size between a paragraph to a few pages, drawn from a longer corpus. To allow good prediction for the model, the text-piece could include ancillary text that preceded it the longer corpus. The model need not do predictions for the words in this ancillary text but can use it to make predictions on the text-piece.

tions analogous to the Winograd example in Section 15.4. Specifically, we assume that an (unknown) process CLOZE has been used to add such cloze questions to the text pieces at test time. These are clearly-marked multiple-choice questions in simple English that the model has to answer. Note that the training corpus did not contain such cloze questions, so this is a simple form of distribution shift at test time. The prediction loss on cloze questions does not require predicting the location or contents of the cloze question —it only requires selecting the correct answer to the multiple-choice cloze question.

We allow the process CLOZE to tailor the questions to the model being tested. Thus the next assumption is reasonable.

Assumption 15.8.2. *[Cloze Sufficiency Assumption:]* The pre-trained model's average (multiclass) prediction loss on Cloze questions — where the average is taken over the distribution of text pieces– closely tracks (within a small multiplicative factor like 1.1) the excess cross-entropy of the model on classical next-word prediction.

Note: As discussed in Section 15.4, if the cloze question is assumed to be perfectly answerable by a human then any incorrect answers by the model can be interpreted analogously excess cross entropy. Our assumption amounts to saying that mistakes on cloze questions closely capture the excess entropy of the model as defined in (15.3). The next theorem, shows that there *exists* a set of cloze questions (albeit fairly artificial) where the excess cross-entropy of the model's answer tracks the overall excess cross-entropy on next-word prediction.

Theorem 15.8.3. *If a model's excess entropy at the ith place in text is ϵ then there is a cloze question with binary answer such that the probability that the model answers it incorrectly is at most $\sqrt{2\epsilon}$.*

Proof. The proof involves Pinsker's Inequality (wikipedia version) which relates variation distance and KL divergence. As in Section 15.4 let $p_i()$ be the humans' probability distribution for the $i + 1$th word in the text piece and $q_i()$ be the model's distribution. The probability that the human and the model give different answers is the variation distance between the two distributions, which is the maximum (over all subsets A of words) of $\sum_{w \in A}(p_i(w) - q_i(w))$. Let A_{i+1} denote the subset for which the previous expression is maximised. The cloze question consists of replacing word w_{i+1} in the text with the question: *Is the next word among the words listed in option (a) or in option (b)*, where option (a) lists words in A_{i+1} and (b) lists words in $\overline{A_{i+1}}$. The theorem now follows from Pinsker's inequality. □

15.8.1 Skills: A Statistical View

Language is assumed to have an underlying set S of *skills*. Every text-piece t has an associated set of skills that are required for comprehending it. The theory allows this set of skills to be quite large —it only needs to be (a fair bit) smaller than the number of text-pieces in the distribution (an enormous number).

Definition 15.8.4 (skill graph)**.** *A* skill graph *is a bipartite graph (S, T, E) where nodes in S correspond to skills, nodes in T correspond to text-pieces, and (s, t) is in the edge set E if "comprehending" text-piece t (i.e., answering its associated cloze questions) requires using skill s. (See Figure* **??***)*

It is important to realize that we are interested in quantifying the model's *competence* on a skill. For example, while the above definition assumes there the distribution of text-pieces includes those whose comprehension requires the skill "anaphora resolution," a language model (or even human individuals!) will in general be unable to apply the skill correctly in all text pieces. Thus "competence on anaphora resolution" is not $0/1$ —instead it is quantified as the fraction of text-pieces associated with this skill whose cloze questions were correctly answered by the model. Quantifying the success rate of this (in other words, the model's capabilities) is the goal of the rest of the paper.

The final element of our theory is that the skill-graph has random edges, as made precise in Definition 15.8.5. To understand why this makes sense, we recall Winograd's example: *The city councilmen refused the demonstrators a permit because they feared violence*. Winograd implicitly assumes that the trickiest skill needed here is pronoun/anaphora resolution, but of course, applying that skill in this context requires other skills: understanding of causality (i.e., interpretation of "because") as well as world knowledge about "city councilmen," "permit," "demonstrators," etc. This example highlights the fact that if we were to look at random text-pieces that require pronoun disambiguation, we would encounter random real-world scenarios, whose comprehension requires very different set of skills. Moreover, the scenarios (and hence the relevant skills) could have different probabilities of occurring in the corpus.

For simplicity we assume that each text-piece requires exactly k skills for some k, and this set was drawn by iid sampling from an underlying measure on the set of skills. (Thinking of k as a random variable is natural but will not be considered here.) The next definition formalizes the above framework in form of a *skill cluster*.

Definition 15.8.5 (Degree-k skill cluster)**.** *This is a skill graph (S, T, E)*

where the collection of text pieces is generated by "nature" by applying the following process: pick a subset of k skills via iid sampling from an underlying measure μ_1 on skills, and then use a procedure GEN *to create a text-piece t whose comprehension requires these skills, as well as a measure $\mu_2(t)$ associated*[25] *with this text piece t. Then nature uses process* CLOZE *to add cloze prompts to test comprehension on t. The* prediction loss *on the text-piece is the cross-entropy loss on predicting the answers to the cloze questions in it. The average prediction loss over all text-pieces is computed with respect to the measure $\mu_2()$. We call the skill-graph thus produced a* degree-k skill cluster*.*

Now we formalize a simple model of what the full text corpus looks like. More complicated extensions of this framework (e.g., considering a hierarchy among corpora) are left for future work.

Definition 15.8.6. *(Text corpus) The text corpus consists of many skill clusters (e.g., math, newspapers, science, coding, etc.) $(S, T_1, E_1), (S, T_2, E_2), \ldots$ which share the same underlying set of skills S but have disjoint sets of text-pieces $T_1, T_2, \ldots$ that are generated as in Definition 15.8.5.*

Definition 15.8.5 allows us to define "competence on a skill" in the more familiar setting of statistical learning theory, specifically by letting us associate a statistical task with it. The task involves predicting answers to cloze questions in a sub-distribution of text pieces that contain that skill. Our emergence theory will apply to the family of tasks of the next definition.

Definition 15.8.7 (Competence on Skills)**.** *In the setting of Definition 15.8.5, for each skill cluster and each skill $s \in S$* statistical task τ_s corresponding to s and this cluster *is defined as follows. The learner is given a text-piece created by sampling $s_1, \ldots, s_{k-1}$ via iid sampling $(k-1)$ times from measure μ_1, and applying* GEN *and* CLOZE *to the skill-tuple $(s, s_1, \ldots, s_{k-1})$ to convert it into a text piece t with an associated measure $\mu_2(t)$ (but the measure is re-scaled so that the total measure of the inputs to this task τ_s is 1). The* error rate *of the model at the statistical tasks is the expected prediction loss on text-pieces drawn from the above distribution. Since error rate is between 0 and 1, the* competence *refers to $(1 - \text{error rate})$.*

For every k'-tuple of skills $(s_1, s_2, \ldots, s_{k'})$ (where $k' \leq k$) the statistical task $\tau_{s_1, s_2, \ldots, s_{k'}}$ corresponding to that k'-tuple is similarly defined. The inputs to the task are generated by completing the k'-tuple to a k-tuple $\vec{s}$ by iid sampling of $k - k'$ additional skills from μ_1 and then using GEN *and* CLOZE *to convert it into a text-piece.*

Competence on the k'-tuple is defined just as above. [26]

Note: The definition involves the k-tuple being picked by iid sampling from μ_1 which, in principle, allows a skill to be picked twice.

[25] Note that the measure on text-pieces has to have the correct marginals e.g., the μ_2-measure of all text-pieces containing a skill s is $\mu_1(s)$. There are many measures satisfying this weak condition, since the number of text pieces is way larger than the number of skills.

[26] Thus if a text-piece involves 5 skills, then that text-piece will appear in 5 statistical tasks corresponding to individual skills, $\binom{5}{2}$ tasks corresponding to pairs of skills, and so on. However, our method of measuring the loss incurred on these statistical tasks implicitly assumes that if the model incorrectly answered this cloze question (i.e., it assigned significant probability to the wrong answer), then that loss was incurred in *all* these statistical tasks. This is a conservative estimate of competence on skills.

However, the probability of picking the same skill twice scales as $O(1/|S|)$. Since the set of skills S is assumed to be large, the distribution is almost the same as sampling distinct k-tuples of skills. The small difference of $O(1/|S|)$ between the two methods will not affect any of the random graph theory calculations.

15.9 *Analysis of Emergence (uniform cluster)*

We have arrived at a core mathematical issue around emergence: *As the model's excess cross entropy goes down (due to scaling), this improves the model's performance on cloze tasks inserted in the test stream (Assumption 15.8.2). How does this improve competence on the skills as well as on tuples of skills –in other words, performance on the associated cloze questions?*

This section analyzes a simple setting where the test-stream consists of a single degree-k skill cluster, and the skills are uniformly distributed and so are the text-pieces—in other words, the distributions μ_1 and μ_2 in Definition 15.8.5 are uniform. Section **??** will extend the analysis to the general setting. The calculations below only require the total number of skills to be much less than the support size of the distribution of text—in other words, the set of skills can be extremely large.

Let's say the model *makes a mistake* on a text-piece if the total prediction loss on all the cloze-questions[27] of that text-piece is at least $1/2$. As the model is scaled up, there are two distinct phases.

Phase 1: In every text-piece, the error on its cloze questions exceeds $1/2$ *.*
Thus the model has not developed competence in any skill, since it is making mistakes in every text-piece.

Phase 2: On the average text-piece, the total error on all cloze questions is less than $1/2$.
Now one begins to see nontrivial competence on some skills. The analysis below will kick in.

If the average cross-entropy loss for the text-pieces is δ we conclude Y consists of at most 2δ fraction of text pieces. The following result guarantees that statistical tasks corresponding to most skills do not assign significant probability to text pieces in Y –in other words, the model has good performance on statistical tasks connected with these skills.

Theorem 15.9.1 (Basic)**.** *Let* $\alpha, \beta, \theta > 0, \beta > 1, \alpha\beta < 1, \theta < 1$ *satisfy*

$$H(\theta) + k\theta \left(H(\beta\alpha) - \beta\alpha \log \frac{1}{\alpha} - (1 - \beta\alpha) \log(\frac{1}{1 - \alpha}) \right) < 0 \qquad (15.16)$$

and the distribution on skills and text pieces be uniform in the skill-cluster. Then irrespective of the details of GEN *and* CLOZE *processes, the following*

[27] This is the amount of error incurred if the incorrect answer is chosen with noticeable probability on even a single cloze question).

property holds for every subset Y of text pieces that contains at least θ fraction of text pieces: at least $1-\alpha$ fraction of skills have at most $\beta\theta k|T|/|S|$ edges to Y (in other words, at most β times the number of edges a skill would be expected *to have to text-pieces in Y).*

Note: Since edges between a skill node s and set Y correspond to errors in the statistical task τ_s, Theorem 15.9.1 is giving a lower bound bound for competence on $(1-\alpha)$ fraction of skills: it is at least $1-\beta\theta$.

15.9.1 *Proof of Theorem 15.9.1.*

The proof uses the famous *Probabilistic Method* [28]. Here, one is trying to show that in a certain probability space, there are no *bad* outcomes. Denoting by W an integer random variable denoting the number of bad outcomes, if we show that $\mathbb{E}[W] \approx 0$, then it follows that $W = 0$ with probability at least $1 - \mathbb{E}[W]$. Concretely, in the proof, W will be the number of "bad" set pairs (Y, Z) of a certain size that violate the lemma.

Proof. For $Y \subseteq V_1, |Y| = \theta|T|$ and $Z \subseteq S, |Z| \leq \alpha|S|$ we say that (Y, Z) are *bad* if Z has at least $\alpha\beta\theta k|T|$ edges to Y. Let W denote the number of such Z's. The expectation is upper bounded by

$$|S||T|\binom{|S|}{\alpha|S|} \times \binom{|T|}{\theta|T|} \times \binom{k\theta|T|}{\beta\alpha k\theta|T|} \times \alpha^{\beta\alpha\theta k|T|} \times (1-\alpha)^{(1-\beta\alpha)\theta k|T|} \tag{15.17}$$

For (15.17) to be $\ll 1$ it suffices for its logarithm to be negative. By Stirling's approximation $\binom{N}{tN} \leq 2^{(H(t)+\epsilon_N)N}$ where $H(t) = -t\log t - (1-t)\log(1-t)$ is the binary entropy function and ϵ_N goes to zero rapidly as $N \to \infty$. Applying this to (15.17) and taking logarithms, and assuming $|S| \ll |T|$, we arrive at the condition (15.16) for large $|T|$. □

15.9.2 *Competence on skills and tuples of skills: Performance Curves*

Definition 15.9.2 (performance curve)**.** *The contour plot (i.e., the boundary) of the region of α, β combinations satisfying Theorem 15.9.1 is called a* performance curve *and denoted $C_{(k,\theta)}$. A performance curve C is* better *than another curve C' if for every α, β on C there is a corresponding point (α, β') on C' for $\beta' > \beta$.*

We draw performance curves by ploting $(1-\alpha)$ on the horizontal axis and the vertical axis plots $\beta\theta$, so point $(0.8, 0.16)$ on a curve means at least 0.8 fraction of skills have at most 0.16 fraction of their edges in the "error set" Y (hence 0.84 fraction of their edges are outside the error set). The emergence curves shift down noticeably (i.e.,

[28] N Alon and J Spencer. *The Probabilistic Method (4th Ed)*. Wiley, 2016

imply emergence of more skills) as we increase k. The next lemma shows this trend always holds.

Lemma 15.9.3 (Monotonicity)**.** *If $\theta' < \theta$ then the performance curve for θ', k lies below that for θ, k.*

If $k' > k$ then the performance curve of θ, k' lies below that for k, θ.

Proof. Follows from the fact that $H(\theta)/\theta$ is a decreasing function in the interval $(0, 1)$. □

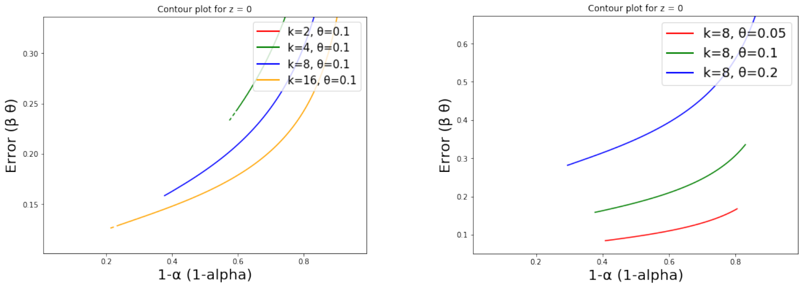


Figure 15.2: Performance Curves: Left plot has $\theta = 0.1$ and varies $k = 2, 4, 8, 16$. Higher values of k greatly improve performance (for $k = 2$ valid α, β did not exist). The right plot has $k = 8$ and $\theta = 0.05, 0.1, 0.2$. Section **??** clarifies that it also describes the model's performance curve for t-tuples of skills for for $\theta = 0.05$ and $t = 1, 2, 4$ respectively (e.g., blue curve for 4-tuples).

15.9.3 *The tensorization argument*

While the above method yields performance curves, better curves can be derived via a tensorization argument. Consider the following k'*-wise recombination* operation on the test stream. First randomly partition the test stream into subsets of size k', and then concatenate the k' text pieces within each subset to create a larger piece of text that we refer to as a "k'-piece," and whose measure is the sum of the measures of the component test-pieces. All cloze questions for the old test-pieces are retained and no new cloze questions are inserted. Clearly, if the error of the model per average text-piece was δ, then the error per average b-piece is $k'\delta$. However, each k'-piece is now using a random $k'k$-tuple of skills. Importantly, this set of $k'k$ skills consists of iid draws from the skill distribution, which can also be seen as a k-tuple of k'-tuples. Thus viewing k'-tuples of skills as 'complex skills' we can use these complex skills as the skill set in the setting of Theorem 15.9.1, which gives us an easy corollary quantifying the performance on tasks corresponding to k'-tuples of skills.

Lemma 15.9.4 (Emergence for k'-tuples of skills)**.** *Consider the skill-graph (S', T', E) where S' consists of all k'-tuples of skills, T' consists of k'-pieces, and E consists of (s', t') where s' is a k'-tuple of skills and t' is a k'-piece where this tuple of skills is used. Let Y consist of θ fraction of k'-pieces. Then for any $\alpha, \beta > 0, \beta > 1, \alpha\beta < 1$ satisfying (15.17) there are at least $1 - \alpha$ fraction of k'-tuples of skills that have at most $\alpha\beta\theta\theta N_1$ $\beta\theta$ fraction of their edges connected to Y.*

The next problem asks you to derive a somewhat surprising general principle that's also hinted at in caption of Figure 15.2. Assume (for simplicity) a Chinchilla-like scaling law that 10x up-scaling leads to factor 2 reduction in excess entropy. If a model is considered to have reasonable performance on individual skills at current scaling, then after further up-scaling of 10x one would see similar reasonable performance on skill-pairs, and scaling up by yet another 10x after that will yield similar reasonable performance on 4-tuples of skills, etc. Note that these are *provable lower bounds* on performance gains—actual gains could be higher. Figure 15.2 illustrates the phenomenon.

Problem 15.9.5. *Show that when a model M_1 with loss δ is scaled up (e.g., as per equation (15.8)) so that the new model M_2 has loss δ/k', then the performance curve inferred by our method for k'-tuples of skills using M_2 is identical to the curve inferred for individual skills on model M_1.*

16
Generative Adversarial Nets

Chapter 14 described some classical approaches to generative models, which are often trained using a log-likelihood approach. We also saw that they often do not suffice for high-fidelity learning of complicated distributions such as the distribution of real-life images. *Generative Adversarial Nets (GANs)* is an approach that generates more realistic samples. It relies upon power of deep nets at discriminative tasks. For convenience throughout this chapter we assume the data of interest are images, which we think of as points in $\Re^d$ for some d. The model is trying to generate realistic images.

Before developing the theory of GANs we survey various notions of how to test similarity of two distributions, because that discussion feeds directly into the setup used in GANs. This is relevant because Example 14.2.1 illustrated how the the notion of *generalization* from supervised learning can lead us astray when it comes to reasoning about correctness of distribution learning.

16.1 Distance between Distributions

How can we measure how different two distributions P and Q are? If we have access to a formula for computing the density of each distribution, then one can compute an f-divergence for any suitable $f\colon \Re \rightarrow \Re$ that is convex.

$$D_f(P||Q) = \int f(\frac{P(x)}{Q(x)})Q(x)dx \tag{16.1}$$

Problem 16.1.1. *(i) Show that f-divergence is nonnegative. (ii) Show that the f-divergence for $f(t) = t\log t$ coincides with $KL(P||Q)$. (iii) Show that the f-divergence for $f = \frac{1}{2}|t-1|$ coincides with total variation (or ℓ_1) distance: $|P - Q|_1 = \int |P(x) - Q(x)|dx$.*

However, in practice one doesn't have a formula for the probability density function, and must estimate distance using samples from P

and Q. A natural idea is to compare the expectation of a class of *test* functions on the two distributions.

Example 16.1.2. *The expectation $\mathbb{E}_P[x]$ is the mean of the distribution P. Similarly expectation of monomials of form $x_{i1}x_{i2}\cdots x_{ik}$ constitutes the kth moment. Moments can be estimated from samples (under fairly general conditions) and the difference of moments of distributions P, Q can be seen as some measure of their difference.*

Unfortunately, accurate estimation of all higher moments for multivariate distributions gets expensive with respect to both computation time and sample complexity. This motivates a notion of distance that arises in *transportation metrics*. [1] If $\mathcal{F}$ is a class of functions, then define the distance between P and Q using the highest different in expectation achievable over functions in $\mathcal{F}$.

$$d(P,Q) = \sup_{f\in\mathcal{F}} |\underset{x\sim \mathrm{P}}{\mathbb{E}}[f(x)] - \underset{x\sim Q}{\mathbb{E}}[f(x)]| \qquad (16.2)$$

For example $\mathcal{F}$ could be polynomials of degree at most k.

Problem 16.1.3. *Show that the distance defined above satisfies triangle inequality.*

16.2 Introducing GANs

Generative Adversarial Nets (Goodfellow et al.[2]) is a framework to learn generative models via the definition in (16.2). Specifically, one tries to train a generative model G, which (as in Chapter 14) produces an image $G(u)$ using random vector u. This yields a distribution on images, and we check the quality of this distribution by using a class $\mathcal{F}$ of deep nets (with a fixed size and architecture) and estimate the distance in (16.2) by trying to find a net that is a maximiser of the expression. Now we spell out the main components of GANs.

Idea 1: Since deep nets are good at recognizing images —e.g., distinguishing pictures of people from pictures of cats—why not let a deep net be the judge of the outputs of $G()$?

More concretely, suppose images are represented as vectors in $\Re^d$ and let P_{real} be the distribution over real images. The generator G has learned to generate synthetic images from a new distribution P_{synth} (i.e., the distribution of $G(h)$ when h is a random seed). We could try to train a discriminator deep net D that maps images to numbers in $[0,1]$ and tries to discriminate between these distributions in the following sense. Its expected output $E_x[D(x)]$ as high as possible when x is drawn from P_{real} and as low as possible when x is drawn from P_{synth}. The discriminator can be trained

[1] Please look up Wasserstein metrics and Earth-Mover distance on the internet.

[2] I Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, A Courville, and Y Bengio. Generative Adversarial Networks. *NeurIPS*, 2014

with standard supervised learning (e.g, regression) with two labels. If $P_{synth} = P_{real}$ then of course no classifier can achieve a gap in this expected output, and so the training will fail. If, on the other hand, we are able to train a good discriminator deep net —one whose average output is noticeably different between real and synthetic samples— then this is proof positive that the two distributions are different. [3]

Idea 2: If a good discriminator net has been trained, use it to provide "gradient feedback"that improves the generative model.

The natural goal for the generator is to make $E_h[D(G(h))]$ as high as possible, because that means it is better able to fool the discriminator D. Given a fixed D the natural way to improve G is to pick a few random seeds h, and slightly adjust the trainable parameters of G to increase this objective. Note that this gradient computation involves back-propagation through the composed net $D(G(\cdot))$. [4]

Idea 3: Turn the training of the generative model and the accompanying discriminator net into a game of many moves (i.e., rounds of parameter updates).

Each move for the discriminator consists of taking a few samples from P_{real} and P_{synth} and improving its ability to discriminate between them. Each move for the generator consists of producing a few samples from P_{synth} and updating its parameters so that $E_u[D(G(h))]$ goes up a bit.

Notice, the discriminator always uses the generator as a black box —i.e., never examines its internal parameters —whereas the generator needs the discriminator's parameters to compute its gradient direction. Specifically, the gradient for G is computed by backpropagating through D. Also, the generator does not ever use real images from P_{real} for its computation. (Though of course it does rely on the real images indirectly since the discriminator is trained using them.)

One can fill in the above framework in multiple ways. The most obvious is that the generator could try to maximize $E_u[f(D(G(h)))]$ where f is some increasing function. (We call this the *measuring function.*) Concretely, if D, G are deep nets with specified architecture and whose number of parameters is fixed in advance by the algorithm designer, then the training objective is:

$$\min_G \max_D \quad E_{x \sim P_{real}}[f(D(x))] + E_h[f(1 - D(G(h)))]. \tag{16.3}$$

Problem 16.2.1. *(1) Write an expression for updates for G and D for the loss in (16.3) when f is the identity map (i.e. $f(x) = x$). (2) Write an expression where G and D anticipate the effect of the other's immediate response to their update. (This can be done in more than one way.)*

[3] There is an in-between case, whereby the distributions are different but the discriminator net doesn't detect a difference. This case is going to be important in the story very soon.

[4] These updates to G assume D is fixed and vice versa. Many papers have suggested alternative update methods that anticipate D's response to this update, and show evidence that this makes training more stable in practice. See Problem 16.2.1.

Effect of f: The measuring function has the effect of giving different importance to different samples. Goodfellow et al. originally used $f(x) = \log(x)$, which, since the derivative of $\log x$ is $1/x$, implicitly gives much more importance to synthetic data $G(u)$ where the discriminator outputs very low values $D(G(h))$. In other words, using $f(x) = \log x$ makes the training more sensitive to instances which the discriminator finds terrible than to instances which the discriminator finds so-so. By contrast, $f(x) = x$ gives the same importance to all samples and results in *Wasserstein GAN*.

Problem 16.2.2. *Show that if the discriminator has unbounded capacity (i.e., able to compute any function) then for $f(x) = \log x$ the optimum value of the expression in (16.3) is the following quantity (called* Jensen-Shannon Divergence*) where $\mu = P_{real}, \nu = P_{synth}$ and KL was defined in Section 5.6:* [5]

$$KL(\mu \| \frac{\mu+\nu}{2}) + KL(\nu \| \frac{\mu+\nu}{2}).$$

16.2.1 Game-theoretic interpretation and implications for training

A serious practical difficulty in implementing training as above is that it can be oscillatory, meaning the above objective can go up and down. This is unlike usual deep net training, where training (at least in cases where it works) steadily improves the objective. The reason is that implicitly the discriminator and generator are playing a 2-person zero sum game [6] where their "moves " are the two circuits D, G and the payoff from generator (minimizer) to discriminator (maximizer) is the loss. Thus generator is picking moves to minimize the following payoff

$$\max_D \quad E_{x \sim P_{real}}[f(D(x))] + E_h[f(1 - D(G(h)))]$$

whereas discriminator is maximising

$$\min_G \quad E_{x \sim P_{real}}[f(D(x))] + E_h[f(1 - D(G(h)))].$$

Such games do not always have an *equilibrium,* i.e., a point where both players are reacting optimally to the other and thus lack an incentive to change. (It is akin to a saddle point in optimization.)

Although equilibrium may not exist when viewed as a two-person game, of course during training both players are under the control of the training algorithm. Thus suitable modifications to gradient-based training could conceivably allow convergence to some solution even though it is not an equilibrium. (For example, even if D is not optimal response to the current G, gradients may not allow a way to improve on the current D.) An extensive list of papers present such ideas; some examples are: NEED SOME REFERENCES HERE

[5] Hint: The optimum D will be unrealistically powerful: given input x its output depends on the probabilities $P_{real}(x), P_{synth}(x)$.

[6] Please read up about zero sum games online, including the famous Min-Max Theorem about equilibria.

16.3 "Generalization" for GANs vs Mode Collapse

Section 14.2.1 discussed complications arising when we try to learn distributions from finite samples. In the GAN setting the training objective (16.3) was described using the full distribution but of course in practice the discriminator D is discriminating between finite samples of the two distributions.

Usually in machine learning good generalization means that the average loss function on test dataset is similar to that on the training dataset. However, we saw that for the usual loss functions like log-likelihood, this notion of generalization does not imply that the distribution has been learned well. After GANs were introduced there was extensive study of whether GAN approach could bypass these issues, and in this effort a large number of training objectives and algorithms were tried. It was noted that they were learning the distribution fairly imperfectly[7] but it was unclear whether this would go away with bigger training datasets or different objectives.

Example 16.3.1. *Since the objective (16.3) allows maximization over all neural nets D of the allowed architecture, even two different samples from the same distribution can look very different to a neural net. For example if we take two finite samples from the d-dimensional Gaussian $\mathcal{N}(0, I)$ then even if the samples have size say d^3, they are distinguishable by a deep net that is somewhat larger than d^3. The reason is that the samples are two discrete sets in which all pairs of points are almost orthogonal (with high probaility). While this is an artificial example, it is the case that in real-life GAN training, the objective does not usually drop to zero —the net is indeed able to somewhat distinguish the samples from the two distributions.*

We now describe theoretical analysis from [8] showing that the quality of the learnt distribution is inherently limited by the *representational capacity* of the discriminator *regardless of how large we make the the training dataset*.

As usual when discussing generalization, let us assume the net has some finite size, say N, and the sample size of the distributions is large enough for nets of size N to generalize. The following two problems are drawn from

Problem 16.3.2. *Suppose the loss is C-Lipschitz and the parameter vector is in $\Re^d$ and has ℓ_2-norm at most L. Suppose this discriminator trained on a training set of size N achieved training loss at most ϵ_1. Show that if $N > \Omega(\frac{L}{\epsilon_2^2} \log(C/\epsilon_2))$ then it has test loss at most $\epsilon_1 + \epsilon_2$ on the full distribution.* [9]

Does this imply that GANs actually learn the distribution? No, just as in Example 14.2.1: generalization only implies training and test loss are close, not that the distributions are close.

[7] A representative problem is *mode collapse*: the learnt distribution does not appear to have the same amount of diversity as the

[8] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and Equilibrium in Generative Adversarial Nets (GANs). *Proc. ICML*, 2017

[9] Hint: Use the results of Chapter 5, eg Theorem 5.2.7.

Problem 16.3.3. *Under the same conditions as Problem 16.3.2 show that if the learned distribution P_{synth} has finite support and is a uniform distribution on $\Omega(\frac{L}{\epsilon_2^2}\log(C/\epsilon_2))$ random iid samples from P_{real} then no discriminator can achieve test loss more than $\epsilon_1 + \epsilon_2$ when comparing the distributions P_{synth} and P_{real}.*

In the previous problem, let's think of P_{real} as the distribution on all real-life images. Then P_{synth} is quite different from P_{real}: it is the uniform distribution on some small set of real-life images. Nevertheless no discriminator (whose size, norm and Lipschitz constant are suitably upper bounded) can distinguish between the two distributions. As we will see, such a P_{synth} is indeed observed in practice. Furthermore, changes to GANs architecture (e.g. Encoder-Decoder GANs) do not affect this basic result [10].

The following exercise gives a (probably very loose) upper bound on the size of a generator that can produce such a P_{synth}.

Problem 16.3.4. *Show that the uniform distribution on M images where each image is in $\Re^d$ an be generated using a net with $O(Md^2)$ parameters.* [11]

Putting together the previous three problems we conclude that in the following scenario the GAN objective is insufficient to prevent the verifier from learning a distribution supported on a small finite set of images ("mode collapse"): The generator has somewhat larger "capacity" than the discriminator. [12]

16.3.1 *Experimental verification of Mode Collapse: Birthday Paradox Test*

The theory above suggests that GANs trained using discriminators of a certain "capacity"(in the sense of generalization theory) have solutions with low training and test error where the synthetic distribution P_{synth} is supported on a small set of images and thus is quite different from P_{real}). This phenomenon of the GAN ending up with a synthetic distribution consisting of a small set of images is called *mode collapse* and was earlier believed to be a result of either failed training or using a training set of real images that is too small. The results above suggested that mode collapse is not a surprising outcome with generators and discriminators of low-ish capacity (as opposed to capacity that scales with the number of distinct modes in P_{real}.)

This raised a question whether we can detect such model collapse in real-life GANs. In other words, estimate how many "distinct"images it can produce. At first glance, such an estimation seems very difficult. After all, automated/heuristic measures of image similarity can be easily fooled, and we humans surely don't have enough

[10] Do GANs learn the Distribution? Some Theory and Empirics

[11] More precisely the distribution will be a mixture of gaussians of tiny variance centered at the M images.

[12] It is an open question whether mode collapse can be avoided when the discriminator is much larger, although in that case usually the training loss is hard to reduce, for reasons explored in Example 16.3.1.

time to go through millions or billions of images, right?

Luckily, a crude estimate is possible using the simple birthday paradox, a staple of undergrad discrete math.

Problem 16.3.5 (Birthday paradox). [13] *Consider a uniform distribution on a set of size N. Show that a random sample of size* $2\sqrt{N}$ *contains a duplicate probability at least* $1 - 1/e$. *(The name for this paradox comes from its implication that if you put* 23 *random people in a room, then the odds are good that two of them have the same birthday is significant.)*

Let's realize the implications for GANs. Imagine for argument's sake that P_{real} is the distribution on pictures of faces. What is the number of modes in this distribution? At a minimum it is the number of distinct human faces? This feels like a rather large set, because all of us know tens of thousands of faces (including those encountered in the news) and don't see any unrelated *doppelgangers*; only identical twins. More precisely, the birthday paradox says that if the number of distinct human faces is N then we would expect to have seen doppelgangers after having seen $\sqrt{N}$ faces.

In the GAN setting, the distribution is continuous, not discrete. Thus our proposed birthday paradox test for GANs [14] is as follows.

(a) Pick a sample of size s from the generated distribution. (b) Use an automated measure of image similarity to flag the 20 (say) most similar pairs in the sample. (c) Visually inspect the flagged pairs and check for images that a human would consider near-duplicates. (d) Repeat.

If this test reveals that samples of size s have duplicate images with good probability, then suspect that the distribution has support size about s^2.

16.3.2 Other notes on GANs and mode collapse

While recent GANs (such as Progressive GAN) use very large discriminators and generators to produce images of better visual quality (as judged by humans) they still suffer from mode collapse, in line with the above theory.

On the other hand, Florian et al [15] argue that the above analysis takes assumes static near-equilibrium in training, whereas real-life training never arrives at an equilibrium, and the training dynamics resulting from non-equilibrium can act as a power shaper of the GAN's behavior.

A recent paper [16] shows that even though GANs suffer from mode collapse, they can be used to predict generalization. In other words, given a dataset S and a discriminative model trained on it, use the following predicter of generalization error: train a conditional GAN

[13] Do GANs learn the Distribution? Some Theory and Empirics

[14] Sanjeev Arora and Yi Zhang. Theoretical Limitations of Encoder-Decoder GANs Architectures. *Proc. ICLR*, 2018

[15] F Schaefer, H Zheng, and A Anandkumar. Implicit competitive regularization in GANs. *ICML*, 2020

[16] Y Zhang, A Gupta, N Saunshi, and S Arora. On predicting generalization using gans. *ICLR*, 2022

using S and use random samples from the trained GAN in lieu of held-out data to predict generalization. This is shown to work better than other predictors of generalization error.

The basic idea of GANs has been extended for other settings, most notably to learn good image to image maps (e.g., changing a photograph to a painting, or virtually trying on an article of clothing on the image of a person). This works very well and there is no analog of the mode collapse result.

17
Self-supervised Learning

Semantic representations (aka *semantic embeddings*) of complicated data types (e.g. images, text, video) have become central in machine learning, and also crop up in machine translation, language models, GANs, domain transfer, etc. These involve learning a representation function f such that $f(x)$ is a compact and "high level" representation of datapoint x —meaning it retains semantic information while discarding low level details — e.g., the colors of individual pixels in an image. The test of a good representation is that it should greatly simplify solving new classification tasks, by allowing them to be solved via linear classifiers (or other low-complexity classifiers) using small amounts of labeled data.

Researchers are most interested in unsupervised representation learning using unlabeled data. A popular early example is *word embeddings* which used simple linear algebra [1] and proved useful in information retrieval for several decades. More recent word embeddings such as word2vec[2] became the inspiration for semantic embeddings of diverse data types such as molecules, social networks, images, text etc.

In this chapter we encounter *self-supervised* learning, a family of methods for learning good representations. Working with unlabeled data, the learner defines a learning objective for finding a good representation function. An important difference from learning paradigms studied elsewhere in the book is that the training and test tasks are different, and hence the notion of generalization does not capture the final goal of learning. This is an example of *training on task A to later do well on task B*, which one imagines is actually an important aspect of intelligent behavior.

[1] LSI paper

[2] word2vec paper; see wikipedia page of word2vec for links to other similar algorithms

18
Adversarial Examples and efforts to combat them

While modern deep nets exhibit superhuman accuracy at solving classification tasks on images, they have a surprising Achilles heel that was first reported in [1]: for most correctly classified images x there is a small perturbation vector δ such that $x + \delta$ is mis-classified by the deep net, and yet to humans $x + \delta$ looks pretty similar to x. These are called *adversarial examples*; note that δ is specially constructed given x using an optimization technique, so $x + \delta$ is not from the usual input distribution that the classifier was trained on. Still it is striking that $x + \delta$ is misclassifier despite looking like a normal image to us humans.

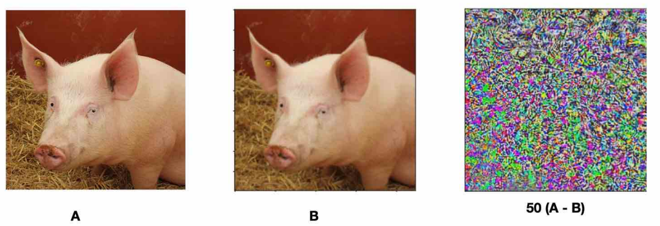


Figure 18.1: *Flying pigs?* (A) is image of a pig, and (B) is a slightly perturbed version of it. A normally trained ResNet50 classifier labels (B) as "airliner." The difference between the two images is tiny; in (C) you see an image that is 50 times the pixel-wise difference between (A) and (B). Without the $50x$ scaling (C) would consist of pixels with values close to 0 (i.e., blank image). *Source: Kolter-Madry Tutorial.*

Adversarial examples have been extensively documented in a variety of datasets and neural architectures. Powerful methods (based upon optimization) have been discovered to find such examples. The concepts and definitions of the current chapter closely track those of Kolter and Madry's online tutorial [2]. Because the phenomenon hints at fragility of current ML-based systems, myriad attempts have been made to mitigate this issue by changing training protocols, though progress has been slow.

18.1 Basic Definitions

The classifier $f : \mathbb{R}^d \to \mathcal{Y}$ maps inputs to labels in a finite set $\mathcal{Y}$. There is an allowed set of *perturbations* $\Delta \subseteq \Re^d$. In this chapter we assume Δ is the set of vectors of ℓ_p norm at most ϵ for some p, where

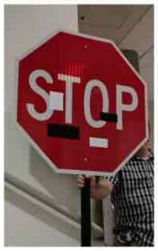


Figure 18.2: An adversarial 3D object! This stop sign with a few stickers on it reliably fooled image recognition classifiers to classify it as a 45mph speed limit.

[1] C Szegedy, W Zaremba, I Sutskever, J Bruna, D Erhan, I Goodfellow, and R Fergus. Intriguing properties of neural networks. In *ICLR*, 2014

[2] Kolter Z and Madry M. Adversarial robustness –theory and practice

p is one of $1, 2, \infty$. The *adversary* has to find a $\delta \in \Delta$ such that $f(x + \delta) \neq f(x)$.

The *targeted adversary* is given a specific target label $y' \neq f(x)$ and has to find a $\delta \in \Delta$ such that $f(x + \delta) = y'$. *Black box attacks* involve an adversary that does not know the internal parameter vector of f; the adversary can only provide inputs to f and see the answer. *White box attacks* allow the adversary access to the internal parameter vector.[3] We will focus on white box attacks. We assume there is a natural loss function $\ell(w, x, y)$ giving the loss of classifier w on input x and label y.

Now we describe the basic attack and defense methods.

18.1.1 Attack method: PGD

A representative (and popular) attack method uses *Projected Gradient Descent (PGD)* [4] where $\text{Proj}_{x_0+\Delta}(x)$ is the closest point of type $x_0 + \delta$ to x, where ℓ_p norm of the perturbation δ is no more than Δ. Furthermore, although x has label y, the label assigned by the classifer to $x_0 + \delta$ is y'.

Note that if norm is ℓ_2, then $\text{Proj}_{x_0+\Delta}(x)$ is simply $x_0 + \Delta\frac{(x-x_0)}{|x-x_0|_2}$.

Problem 18.1.1. *Give methods to compute* $Proj_{x_0+\Delta}(x)$ *for norms* ℓ_1 *and* ℓ_∞.

Now we can describe the most popular method to find adversarial examples.

PGD method: *Given input* x_0*, do the following iteration k times:* $x \leftarrow x + \eta \nabla_x \ell(w, x, y)$*, followed by* $x \leftarrow Proj_{x_0+\Delta}(x)$.

Note that the gradient is with respect to the input x, and not the parameter vector w! It can be computed by a simple modification of backpropagation. [5]

For targeted attacks one can use $\nabla_x \ell(w, x, y) - \nabla_x \ell(w, x, y')$.

Deep nets trained in the usual way are very susceptible to such attacks. For most inputs x, algorithms such as the one above can find a close-by point $x + \delta$ the classifier outputs a different label.

18.1.2 Adversarial Defense

To make the net somewhat resistant to the above attack, one has to train it differently. Specifically it is trained using adversarial examples and taught to classify them correctly. Using a batch of inputs, the adversary (such as the one above)is used to generate adversarial examples. The parameters are now adjusted to make the classifier output the correct label on these examples. Then the adversary is used to generate a new set of adversarial examples for the newly adjusted parameters. This back and forth is repeated some number of

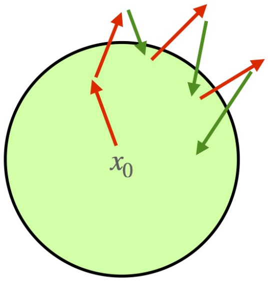


Figure 18.3: PGD attack on input x_0. The red arrows correspond to gradient-based updates. When they produce a point outside $\text{Ball}(x_0, r)$ a projection operation (denoted by the green arrows) finds the closest point in the ball.

[3] While black box attacks may appear hopeless, in practice they do exist. Adversaries generate adversarial images using white box attacks on their own deep nets trained on the same dataset. These images are able to fool other nets with unknown architecture and parameters. Success of black box attacks hints that different architectures are fairly similar in their classification behavior.

[4] A Madry, A Makelov, L Schmidt, D Tsipras, and A Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *ICLR*, 2018

[5] Often attacks use so-called *sign gradient*, which rounds all positive coordinates to $+1$ and negative coordinates to -1.

times. Essentially the classifier is being trained to move the decision boundary away from the datapoints; see Figure 18.4.

The above prototocol has many variants, but the end result is a utility/robustness trade off: the ability of the adversary to find adversarial examples is blunted a fair bit, so that instead of being able to flip the answer for almost all inputs x, it can only do so for, say, 40% of the inputs. The downside is that in the process the classifier's overall accuracy on the original dataset (i.e., the usual inputs) drops a fair bit as well (say, from 95% to 80%). Thus the robust classifier has significantly lower utility for the non-adversarial setting.

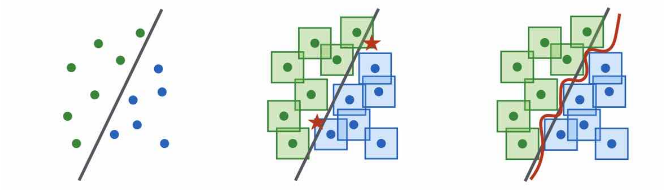


Figure 18.4: Conceptual illustration of adversarial examples for ℓ_∞-bounded perturbations. In the vanilla classifier most datapoints are close to the decision boundary, as measured by ℓ_∞ distance. The red stars are adversarial examples. After adversarial training, a small ℓ_∞-ball around the datapoint no longer intersects the decision boundary. Credit: Madry et al 2018.

Realize moreover that at the end of training the classifier only has the ability to evade adversarial examples generated by the particular adversary (i.e., attack algorithm) it trained with. Often using a *different* attack algorithm new adversarial examples can be generated. This has been shown many times, and is a very frustrating state of affairs indeed.

18.1.3 *Other defense ideas*

A lot of energy was spent on trying to avert adversarial attacks by building in some form of obfuscation inside the classifier, usually by performing a non-differentiable transformation inside the classifier net. The motivation is to make it difficult to implement the gradient-based attacks mentioned above. However, most such defenses were broken with some ingenuity; for an introduction see [6].

18.2 *Provable defense via randomized smoothing*

Since many defenses were actually broken, often within months, it seems advisable to have a more principled defense with some mathematical proof of security. We now describe *randomized smoothing*, the best class of such defenses known – albeit it leads to significant lowering of classification accuracy. [7] For simplicity we describe the idea for ℓ_2-attacks.

Definition 18.2.1. *If $\mathcal{D}$ is a distribution on (input, label) pairs, where inputs are in $\Re^d$, then classifier g is* γ-robust at (x, y) *if $f(x') = y$ for all x' such that $\|x' - x_0\|_2 \leq \gamma$.*

[6]

[7] We do not survey another approach to provable defense via theorem-proving based upon mixed-integer programming, which has seen extensive work as well but does not so far match the guarantees provided by randomized smoothing.

The idea in randomized smoothing is to try to produce a robust classifier by taking a local spatial average of the outputs of another classifier. Below, we will for simplicity equate *ℓ_2-ball of radius $\beta\sqrt{d}$ around x* with the gaussian distribution $\mathcal{N}(x, \beta^2 I)$, since the two are very close.

Definition 18.2.2. *If $\mathcal{D}$ is a distribution on (input, label) pairs and g is a classifier, then* the β-smoothing of g, *denoted g^β, is the classifier that, given x, outputs the probabilistic answer $g(x+\delta)$ where δ is a random vector from $\mathcal{N}(0, \beta^2 I)$. The classifier g^β_{smooth} is a classifier that on input x gives the* plurality label, *namely, the label that is given highest probability by g^β. (It breaks ties among labels arbitrarily.)*

While definition of g^β involves an average over a continuous distribution, the probability that $g^\beta(x) = y$ for a particular label y can be estimated with arbitrary additive accuracy by sampling. The output of the classifier g^β_{smooth} can be similarly determined via sampling with probability close to 1.[8] The goal will be to show that g^β_{smooth} is γ-robust for some small γ.

Theorem 18.2.3. *If $g^\beta(x)$ outputs label y with probability p_a then $g^\beta(x')$ outputs y with probability at least*

$$\Phi(\Phi^{-1}(p_a) - \frac{1}{\beta}|x - x'|_2)$$

where $\Phi()$ is the cumulative distribution function[9] of the univariate Gaussian $\mathcal{N}(0,1)$.

The theorem is proved a few paragraphs later. First we note the following simple corollary.

Corollary 18.2.4. *If $p_a = \Pr[g^\beta(x) = y]$ and all other labels are given probability at most p_0, then y is the label given highest probability by $g^\beta(x+\delta)$ provided $|\delta|_2 \leq \frac{\beta}{2}(\Phi^{-1}(p_a) - \Phi^{-1}(p_0))$.*

Let's understand how the corollary can be used to train a classifier g that is robust to ℓ_2 perturbations. Usually x is an input in the training set with label y, then in normal training you do gradient updates to the deep net towards reducing the sum of the losses associated with assigning label y to x. To ensure robustness to ℓ_2 perturbations, you change training to also assign the same label y to a random sample of noised inputs $x + \delta$ where $\delta \sim \mathcal{N}(0, \beta^2 I)$.

When training ends, using held-out data estimate the fraction ρ of held out images x where (a) the plurality label of g^β is correct and (b) there is a noticeable gap between its probability p_a and the probability p_0 of the next most likely label. Then Corollary 18.2.4 implies that for such points x the classifier g^β_{smooth} outputs the same label y for all x' where $|x - x'|_2 \leq \frac{\beta}{2}(\Phi^{-1}(p_a) - \Phi^{-1}(p_0))$.

[8] In practice this requires a slight gap between the probability of most popular label and that of the second most popular label.

[9] This means $\Phi(t) = \Pr_{z\sim\mathcal{N}(0,I)}[z \leq t]$.

Proof. (Theorem 18.2.3) Letting x' be any point in the neighborhood of x we try to upper bound the difference between $\Pr[g^\beta(x) = y]$ and $\Pr[g^\beta(x') = y]$. Then sampling a random u from $\mathcal{N}(z, \beta^2 I_{d\times d})$ can be alternatively viewed as first picking the projection of this vector along $x' - x$ according to the univariate gaussian $\mathcal{N}(z, \beta^2)$ and then picking the rest of the vector perpendicular to $x - x'$ according to the the $d - 1$ dimensional gaussian $\mathcal{N}(z, \beta^2 I_{d-1\times d-1})$. Say z is the projection on the infinite line passing through x, x'. Define $E(z) = \int_u 1_{g(u)=y} du$ where $\int_u$ integrates over the $d - 1$ dimensional distribution $\mathcal{N}(0, \beta^2 I)$ in such an hyperplane at z. Then we have

$$\Pr[g^\beta(x) = y] = \int_z E(z) dz \tag{18.1}$$

where $\int_z$ integrates over the univariate Gaussian density $\mathcal{N}(x, \beta^2)$. A corresponding expression holds for $\Pr[g^\beta(x') = y]$, with $\int_z$ integrating over univariate Gaussian density $\mathcal{N}(x', \beta^2)$ centered at x'. What is the largest difference between the two, assuming x is to the left of x' on this line?

Intuitively it seems clear that $\Pr[g^\beta(x) = y] - \Pr[g^\beta(x') = y]$ is maximized when $E(z)$ takes its highest possible value, 1 on points close to x, and then begins to switch to lower values closer to x'. Assuming this intuition is correct[10] let's see how low it can get at x'. the worst case must be when $E(z)$ is 1 for $z = x$ to $z < x + \beta\Phi^{-1}(p_a)$, and zero to the right of that. Then since $g^\beta(x') = g^\beta(x + x' - x)$, the same $E(z)$'s enter the average at x' with somewhat different weighting, corresponding to a shift by $|x - x'|/\beta$ standard deviations in the standard normal distribution. Thus in this worst-case configuration $\Pr[g^\beta(x') = y]$ equals $\Phi(\Phi^{-1}(p_a) - |x - x'|/\beta)$ and in general is at least that. □

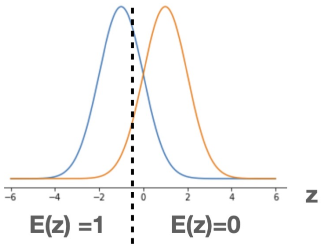


Figure 18.5: Univariate gaussians centered at x (blue one)and x' (the red one), and the point where $E(z)$ switches from 1 to 0.

The analysis above can be tightened a bit; see [11]. While the above argument relies upon the close relationship between the Gaussian distribution and ℓ_2 distance, it can be extended to ℓ_p bounded attacks using suitable analogs for ℓ_p distance.

Problem 18.2.5. *If g is a function mapping data points in $\Re^d$ to $\Re$ and $g(x) \leq 1$ for all x, then show that there is a constant C (independent of d) such that the gradient of g^1_{smooth} has norm at most C.*

[10] Correctness of this intuition is implied by the *Neyman Pearson lemma* of statistics.

[11] H Salman, G Yang, J Li, P Zhang, H Zhang, I Razenshteyn, and S Bubeck. Provably robust deep learning via adversarially trained smoothed classifiers. *NeurIPS*, 2019

19
Examples of Theorems, Proofs, Algorithms, Tables, Figures

In this chapter, Zhao provide examples of many things, like Theorems, Lemmas, Algorithms, Tables, and Figures. If anyone has question, feel free to contact Zhao directly.

19.1 Example of Theorems and Lemmas

We provide some examples

Theorem 19.1.1 (d-dimension sparse Fourier transform)**.** *There is an algorithm (procedure* FourierSparseRecovery *in Algorithm 5) that runs in ??? times and outputs ??? such that ???.*

Note that, usually, if we provide the algorithm of the Theorem/Lemma. Theorem should try to ref the corresponding Algorithm.

For the name of Theorem/Lemma/Corollary ..., let us only capitalize the first word,

Lemma 19.1.2 (Upper bound on the gradient)**.** *Blah blah.*

Problem 19.1.3. *This is how you put in a problem. It inherits chapter and section numbers.*

Theorem 19.1.4 (Main result)**.**

19.2 Example of Long Equation Proofs

We can rewrite $\|Ax' - b\|_2^2$ in the following sense,

$$\begin{aligned}\|Ax' - b\|_2^2 &= \|Ax' - Ax^* + AA^\dagger b - b\|_2^2 \\ &= \|Ax^* - Ax'\|_2^2 + \|Ax^* - b\|_2^2 \\ &= \|Ax^* - Ax'\|_2^2 + \mathrm{OPT}^2\end{aligned}$$

where the first step follows from $x^* = A^\dagger b$, the second step follows from Pythagorean Theorem, and the last step follows from $\mathrm{OPT} := \|Ax^* - b\|_2$.

19.3 *Example of Algorithms*

Here is an example of algorithm. Usually the algorithm should ref some Theorem/Lemma, and also the corresponding Theorem/Lemma should ref back. This will be easier to verify the correctness.

Algorithm 5 Fourier Sparse Recovery Algorithm

```
procedure FourierSparseRecovery($x, n, k, \mu, R^*$)    ▷ Theorem 19.1.1
    Require that $\mu = \frac{1}{\sqrt{k}} \|\widehat{x}_{-k}\|_2$ and $R^* \geq \|\widehat{x}\|_\infty / \mu$
    $H \leftarrow 5, \nu \leftarrow \mu R^*/2,\ y \leftarrow \vec{0}$
    Let $\mathcal{T} = \{\mathcal{T}^{(1)}, \cdots, \mathcal{T}^{(H)}\}$ where each $\mathcal{T}^{(h)}$ is a list of i.i.d. uniform samples in $[p]^d$
    while true do
        $\nu' \leftarrow 2^{1-H}\nu$
        $z \leftarrow$ LinfinityReduce($\{x_t\}_{t \in \mathcal{T}}$)
        if $\nu' \leq \mu$ then return $y + z$    ▷ We found the solution
        $y' \leftarrow \vec{0}$
        for $f \in \mathrm{supp}(y + z)$ do
            $y'_f \leftarrow \Pi_{0.6\nu}(y_f + z_f)$    ▷ We want $\|\widehat{x} - y'\|_\infty \leq \nu$ and the dependence between $y'$ and $\mathcal{T}$ is under control
        end for
        $y \leftarrow y', \nu \leftarrow \nu/2$
    end while
end procedure
```

19.4 Example of Figures

We should make sure all the pictures are plotted by the same software. Currently, everyone feel free to include their own picture. Zhao will re-plot the picture by tikz finally.

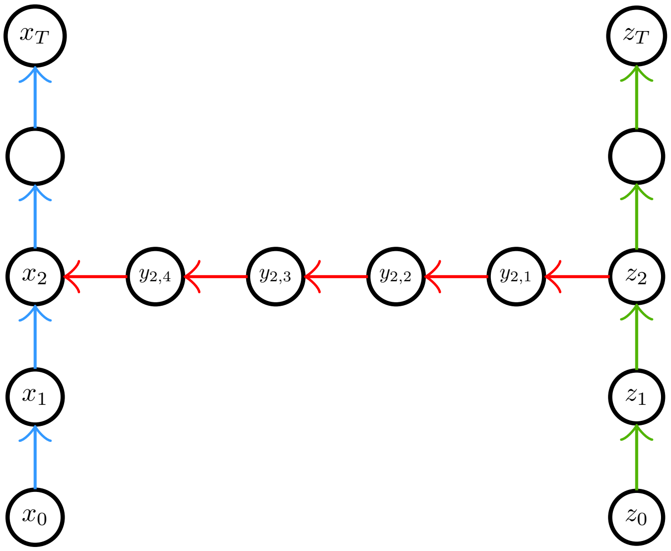


Figure 19.1: A chasing sequence

19.5 *Example of Tables*

Reference	**Samples**	**Time**	**Filter**	**RIP**
[GMS05]	$k \log^{O(d)} n$	$k \log^{O(d)} n$	Yes	No
[CT06]	$k \log^6 n$	$\mathrm{poly}(n)$	No	Yes
[RV08]	$k \log^2 k \log(k \log n) \log n$	$\widetilde{O}(n)$	No	Yes
[HIKP12]	$k \log^d n \log(n/k)$	$k \log^d n \log(n/k)$	Yes	No
[CGV13]	$k \log^3 k \log n$	$\widetilde{O}(n)$	No	Yes
[IK14]	$2^{d \log d} k \log n$	$\widetilde{O}(n)$	Yes	No
[Bou14]	$k \log k \log^2 n$	$\widetilde{O}(n)$	No	Yes
[HR16]	$k \log^2 k \log n$	$\widetilde{O}(n)$	No	Yes
[Kap16]	$2^{d^2} k \log n$	$2^{d^2} k \log^{d+O(1)} n$	Yes	No
[KVZ19]	$k^3 \log^2 k \log^2 n$	$k^3 \log^2 k \log^2 n$	Yes	Yes
[NSW19]	$k \log k \log n$	$\widetilde{O}(n)$	No	No

Table 19.1: We ignore the O for simplicity. The ℓ_∞/ℓ_2 is the strongest possible guarantee, with ℓ_2/ℓ_2 coming second, ℓ_2/ℓ_1 third and exactly k-sparse being the weaker. We also note that all [RV08, CGV13, Bou14, HR16] obtain improved analyses of the Restricted Isometry property; the algorithm is suggested and analyzed (modulo the RIP property) in [BD08]. The work in [HIKP12] does not explicitly state the extension to the d-dimensional case, but can easily be inferred from the arguments. [HIKP12, IK14, Kap16, KVZ19] work when the universe size in each dimension are powers of 2.

19.6 *Exercise*

This section provides several examples of exercises.

Exercises

Exercise 19.6-1: Solve the following equation for $x \in C$, with C the set of complex numbers:

$$5x^2 - 3x = 5 \tag{19.1}$$

Exercise 19.6-2: Solve the following equation for $x \in C$, with C the set of complex numbers:

$$7x^3 - 2x = 1 \tag{19.2}$$

Bibliography

[ADG$^+$16] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, 2016.

[ADH$^+$19a] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332, 2019.

[ADH$^+$19b] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

[ADL$^+$19] Sanjeev Arora, Simon S Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. *arXiv preprint arXiv:1910.01663*, 2019.

[AG23] Sanjeev Arora and Anirudh Goyal. A theory for emergence of complex skills in language models. *arXiv preprint arXiv:2307.15936*, 2023.

[AGL$^+$17] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and Equilibrium in Generative Adversarial Nets (GANs). *Proc. ICML*, 2017.

[AGNZ18] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *Proc. ICML 2018*, pages 254–263, 2018.

[ALL19] S Arora, Z Li, and K Lyu. Theoretical analysis of auto rate-tuning by batch normalization. *ICLR*, 2019.

[aro] Do GANs learn the Distribution? Some Theory and Empirics.

[AS16] N Alon and J Spencer. *The Probabilistic Method (4th Ed)*. Wiley, 2016.

[AZ18] Sanjeev Arora and Yi Zhang. Theoretical Limitations of Encoder-Decoder GANs Architectures. *Proc. ICLR*, 2018.

[BBV16] Afonso S Bandeira, Nicolas Boumal, and Vladislav Voroninski. On the low-rank approach for semidefinite programs arising in synchronization and community detection. In *Conference on learning theory*, pages 361–382, 2016.

[BD08] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14(5-6):629–654, 2008.

[BDK$^+$21] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *arXiv preprint arXiv:2102.06701*, 2021.

[Ber24] Sergei Bernstein. On a modification of chebyshev's inequality and of the error formula of laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math*, 1(4):38–49, 1924.

[BH89] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[BKH16] J Ba, J R Kiros, and G E Hinton. Layer normalization. *NeurIPS*, 2016.

[BM02] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[BM03] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 2003.

[BNS16a] Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Global optimality of local search for low rank matrix recovery. In *Advances in Neural Information Processing Systems*, pages 3873–3881, 2016.

[BNS16b] Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Global optimality of local search for low rank matrix recovery. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3873–3881, 2016.

[Bou14] Jean Bourgain. An improved estimate in the restricted isometry problem. In *Geometric Aspects of Functional Analysis*, pages 65–70. Springer, 2014.

[BR89] Avrim Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Advances in neural information processing systems*, pages 494–501, 1989.

[Bre67] L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 1967.

[BT52] Ralph A. Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of Paired Comparisons. *Biometrika*, 1952.

[BT03] A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 2003.

[BV04] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[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. *arXiv preprint arXiv:1611.01838*, 2016.

[CGV13] Mahdi Cheraghchi, Venkatesan Guruswami, and Ameya Velingker. Restricted isometry of Fourier matrices and list decodability of random linear codes. *SIAM Journal on Computing*, 42(5):1888–1914, 2013.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

[CKL$^+$21] Jeremy Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. *ICLR*, 2021.

[CLB$^+$17] P. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *NeurIPS*, 2017.

[CLG01] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems (NIPS)*, pages 402–408, 2001.

[CT06] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

[CW82] R D Cook and S Weisberg. Residuals and influence in regression. 1982.

[DHS11] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[DHS$^+$19] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems*, pages 5724–5734, 2019.

[DKB15] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Nonlinear Independent Component Analysis. *Proc. ICLR*, 2015.

[DLL$^+$18] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018.

[DP94] X Deng and C Papadimitriou. On the complexity of cooperative solution concepts. *Math of Operations Research*, 1994.

[DPBB17] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, 2017.

[DSDB17] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Eestimation using Real NVP. *Proc. ICLR*, 2017.

[DZPS18] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes overparameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

[DZPS19] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes overparameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2019.

[EHJT04] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of statistics*, 2004.

[Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.

[FSSS11] Rina Foygel, Ohad Shamir, Nati Srebro, and Ruslan R Salakhutdinov. Learning with the weighted trace-norm under arbitrary sampling distributions. In *Advances in Neural Information Processing Systems*, pages 2133–2141, 2011.

[GDG$^+$17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[GHJY15a] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015.

[GHJY15b] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conf. Learning Theory (COLT)*, 2015.

[GJZ17a] Rong Ge, Chi Jin, and Yi Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1233–1242. JMLR. org, 2017.

[GJZ17b] Rong Ge, Chi Jin, and Yi Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. *arXiv preprint arXiv:1704.00708*, 2017.

[GLM16a] Rong Ge, Jason D Lee, and Tengyu Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems*, pages 2973–2981, 2016.

[GLM16b] Rong Ge, Jason D Lee, and Tengyu Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[GLM18] Rong Ge, Jason D Lee, and Tengyu Ma. Learning one-hidden-layer neural networks with landscape design. In *ICLR*. arXiv preprint arXiv:1711.00501, 2018.

[GLSS18a] Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. *arXiv preprint arXiv:1802.08246*, 2018.

[GLSS18b] Suriya Gunasekar, Jason D Lee, Daniel Soudry, and Nati Srebro. Implicit bias of gradient descent on linear convolutional networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[GMS05] Anna C Gilbert, S Muthukrishnan, and Martin Strauss. Improved time bounds for near-optimal sparse Fourier representations. In *Optics & Photonics 2005*, pages 59141A–59141A. International Society for Optics and Photonics, 2005.

[GPAM+14] I Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, A Courville, and Y Bengio. Generative Adversarial Networks. *NeurIPS*, 2014.

[GWB+17] Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6151–6159, 2017.

[HBD+20] A Holtzman, J Buys, L Du, M Forbes, and Y Choi. The curious case of neural text degeneration. *ICLR*, 2020.

[HBM+22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[HHS17] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, 2017.

[HIKP12] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse Fourier transform. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 563–578. ACM, 2012.

[HLLL19] Wenqing Hu, Chris Junchi Li, Lei Li, and Jian-Guo Liu. On the diffusion approximation of nonconvex stochastic gradient descent. *Annals of Mathematical Sciences and Applications*, 4(1), 2019.

[HMR18] Moritz Hardt, Tengyu Ma, and Benjamin Recht. Gradient descent learns linear dynamical systems. In *JLMR*. arXiv preprint arXiv:1609.05191, 2018.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HR16] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled Fourier matrices. In *SODA*, pages 288–297. arXiv preprint arXiv:1507.01768, 2016.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 1997.

[IK14] Piotr Indyk and Michael Kapralov. Sample-optimal Fourier sampling in any constant dimension. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 514–523. IEEE, 2014.

[IPE$^+$22] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022.

[IS15a] S Ioffe and C Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

[IS15b] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[JGH18] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[JGN$^+$17] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. *arXiv preprint arXiv:1703.00887*, 2017.

[JKA$^+$18] S Jastrzębski, Z Kenton, D Arpit, N Ballas, A Fischer, Y Bengio, and A Storkey. Three Factors Influencing Minima in SGD. *ICANN*, 2018.

[JNG$^+$19] Chi Jin, Praneeth Netrapalli, Rong Ge, Sham M Kakade, and Michael I Jordan. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. *arXiv preprint arXiv:1902.04811*, 2019.

[Kap16] Michael Kapralov. Sparse Fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time. In *Symposium on Theory of Computing Conference, STOC'16, Cambridge, MA, USA, June 19-21, 2016*, 2016.

[Kaw16] Kenji Kawaguchi. Deep learning without poor local minima. In *Adv in Neural Information Proc. Systems (NIPS)*, 2016.

[KD19] Diederik P. Kingma and Prafulla Dhariwal. GLOW: Generative Flow with Invertible 1×1 convolutions. *Proc. Neurips*, 2019.

[KGC17] Jan Kukavcka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[KKSK11] Sham M Kakade, Varun Kanade, Ohad Shamir, and Adam Kalai. Efficient learning of generalized linear and single index models with isotonic regression. In *Advances in Neural Information Processing Systems*, pages 927–935, 2011.

[KL17] P W Koh and P Liang. Understanding black-box predictions via influence functions. In *Proc. ICML*, 2017.

[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. In *International Conference on Learning Representations*, 2016.

[KS09] Adam Tauman Kalai and Ravi Sastry. The isotron algorithm: High-dimensional isotonic regression. In *COLT*. Citeseer, 2009.

[KST09] Sham M Kakade, Karthik Sridharan, and Ambuj Tewari. On the complexity of linear prediction: Risk bounds, margin bounds, and regularization. In *Advances in neural information processing systems*, 2009.

[KVZ19] Michael Kapralov, Ameya Velingker, and Amir Zandieh. Dimension-independent sparse Fourier transform. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2709–2728. SIAM, 2019.

[LA19] Z Li and S Arora. An exponential learning rate schedule for deep learning. *ICLR*, 2019.

[Lan02] John Langford. *Quantitatively tight sample complexity bounds*. PhD Thesis CMU, 2002.

[LBZ+22] Z Li, S Bhojanapalli, M Zaheer, Reddi S, and Kumar S. Robust training of neural networks using scale invariant architectures. *arxiv*, 2022.

[LDM12] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

[LLA20] Zhiyuan Li, Kaifeng Lyu, and Sanjeev Arora. Reconciling modern deep learning with traditional optimization analyses: The intrinsic learning rate. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[LMA21] Zhiyuan Li, Sadhika Malladi, and Sanjeev Arora. On the validity of modeling sgd with stochastic differential equations (sdes). *Advances in Neural Information Processing Systems*, 34, 2021.

[LMAPH19] Stéphane Lathuilière, Pablo Mesejo, Xavier Alameda-Pineda, and Radu Horaud. A comprehensive analysis of deep regression. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[LMZ18] Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *Conference On Learning Theory*, pages 2–47, 2018.

[LSJR16] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent converges to minimizers. *arXiv preprint arXiv:1602.04915*, 2016.

[LTW19] Qianxiao Li, Cheng Tai, and E Weinan. Stochastic modified equations and dynamics of stochastic gradient algorithms i: Mathematical foundations. *J. Mach. Learn. Res.*, 20:40–1, 2019.

[LWLA22] Zhiyuan Li, Tianhao Wang, Jason D Lee, and Sanjeev Arora. Implicit bias of gradient descent on reparametrized models: On equivalence to mirror descent. *Advances in Neural Information Processing Systems*, 35:34626–34640, 2022.

[MA19] Poorya Mianjy and Raman Arora. On dropout and nuclear norm regularization. In *International Conference on Machine Learning*, 2019.

[MAV18] Poorya Mianjy, Raman Arora, and Rene Vidal. On the implicit bias of dropout. In *International Conference on Machine Learning*, pages 3537–3545, 2018.

[McA99] David A McAllester. Some pac-bayesian theorems. *Machine Learning*, 37(3):355–363, 1999.

[MHB17] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017.

[MMS$^+$18] A Madry, A Makelov, L Schmidt, D Tsipras, and A Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *ICLR*, 2018.

[MRB$^+$23] Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models, 2023.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[NBE17] Agarwal N, Bullins B, and Hazan E. Second-order stochastic optimization for machine learning in linear time. 2017.

[NBMS18] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *ICLR*, 2018.

[Nes98] Yurii Nesterov. *Introductory Lectures on Convex Programming Volume I: Basic Course*. Springer, 1998.

[Nes00] Yurii Nesterov. Squared functional systems and optimization problems. In *High performance optimization*, pages 405–440. Springer, 2000.

[Ney17] Behnam Neyshabur. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*, 2017.

[NH92] Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473–493, 1992.

[NK19] V Nagarajan and Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. *NeurIPS*, 2019.

[NP06] Yurii Nesterov and Boris T Polyak. Cubic regularization of Newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[NS23] Mark Braverman Sanjeev Aror Nikunj Saunshi, Arushi Gupta. Understanding influence functions and data models via harmonic analysis. *ICLR*, 2023.

[NSS15] Behnam Neyshabur, Ruslan R Salakhutdinov, and Nati Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2015.

[NSW19] Vasileios Nakos, Zhao Song, and Zhengyu Wang. (Nearly) Sample-optimal sparse Fourier transform in any dimension; RIPless and Filterless. In *FOCS*. `https://arxiv.org/pdf/1909.11123.pdf`, 2019.

[NTS15a] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *International Conference on Learning Representations*, 2015.

[NTS15b] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015.

[NY83] A. Nemirovskii and D. Yudin. *Problem complexity and method efficiency in optimization*. Wiley, 1983.

[Pea94] Barak Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 1994.

[PKCS17] Dohyung Park, Anastasios Kyrillidis, Constantine Caramanis, and Sujay Sanghavi. Non-square matrix sensing without spurious local minima via the burer-monteiro approach. In *AISTATS*. arXiv preprint arXiv:1609.03240, 2017.

[RDS04] Cynthia Rudin, Ingrid Daubechies, and Robert E Schapire. The dynamics of adaboost: Cyclic behavior and convergence of margins. *Journal of Machine Learning Research*, 5(Dec):1557–1595, 2004.

[RSM+23] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arxiv*, 2023.

[RV08] Mark Rudelson and Roman Vershynin. On sparse reconstruction from Fourier and Gaussian measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 61(8):1025–1045, 2008.

[SF12] Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

[Sha] Lloyd Shapley. *"Notes on the n-Person Game – II: The Value of an n-Person Game"*. RAND Corporation.

[SHK+14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1), 2014.

[SHS17] Daniel Soudry, Elad Hoffer, and Nathan Srebro. The implicit bias of gradient descent on separable data. *arXiv preprint arXiv:1710.10345*, 2017.

[Smi18] Le Smith, Kindermans. Don't Decay the Learning Rate, Increase the Batch Size. In *ICLR*, 2018.

[SMK23] Ryan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *ArXiv e-prints*, April 2023.

[SQW16a] Ju Sun, Qing Qu, and John Wright. Complete dictionary recovery over the sphere i: Overview and the geometric picture. *IEEE Transactions on Information Theory*, 63(2):853–884, 2016.

[SQW16b] Ju Sun, Qing Qu, and John Wright. A geometric analysis of phase retrieval. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2379–2383, 2016.

[SQW18] Ju Sun, Qing Qu, and John Wright. A geometric analysis of phase retrieval. *Foundations of Computational Mathematics*, 18(5):1131–1198, 2018.

[SS10] Nathan Srebro and Ruslan R Salakhutdinov. Collaborative filtering in a non-uniform world: Learning with the weighted trace norm. In *Advances in Neural Information Processing Systems*, pages 2056–2064, 2010.

[SSJ20] Bin Shi, Weijie J Su, and Michael I Jordan. On learning rates and schrödinger operators. *arXiv preprint arXiv:2004.06977*, 2020.

[SSS10] Shai Shalev-Shwartz and Yoram Singer. On the equivalence of weak learnability and linear separability: New relaxations and efficient boosting algorithms. *Machine learning*, 2010.

[SY19] Zhao Song and Xin Yang. Quadratic suffices for over-parametrization via matrix chernoff bound. *arXiv preprint arXiv:1906.03593*, 2019.

[SYL$^+$19] H Salman, G Yang, J Li, P Zhang, H Zhang, I Razenshteyn, and S Bubeck. Provably robust deep learning via adversarially trained smoothed classifiers. *NeurIPS*, 2019.

[SZA20] F Schaefer, H Zheng, and A Anandkumar. Implicit competitive regularization in GANs. *ICML*, 2020.

[SZS+14] C Szegedy, W Zaremba, I Sutskever, J Bruna, D Erhan, I Goodfellow, and R Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[Tel13] Matus Telgarsky. Margins, shrinkage, and boosting. *arXiv preprint arXiv:1303.4172*, 2013.

[Tro15] Joel A Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

[Wer88] P. J. Werbos. Backpropagation: Past and future. In *IEEE InternationalConference on Neural Networks*, page 343–353, 1988.

[WGL+20] Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In *Conference on Learning Theory*, pages 3635–3673, 2020.

[WH17] Y Wu and K He. Group Normalization. *ECCV*, 2017.

[Win71] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1971.

[WRS+17] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 2017.

[WTB+22] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[ZBH+16a] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[ZBH+16b] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[ZGSA22] Y Zhang, A Gupta, N Saunshi, and S Arora. On predicting generalization using gans. *ICLR*, 2022.

[ZM] Kolter Z and Madry M. Adversarial robustness –theory and practice.

[ZY$^+$05] Tong Zhang, Bin Yu, et al. Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, 2005.