CONTRIBUTORS: RAMAN ARORA, SANJEEV ARORA, JOAN BRUNA, NADAV COHEN, RONG GE, SURIYA GUNASEKAR, CHI JIN, JASON LEE, TENGYU MA, BEHNAM NEYSHABUR, ZHAO SONG,...

THEORY OF DEEP LEARNING

Contents

List of Figures

List of Tables

Introduction

This monograph discusses the emerging theory of deep learning. It is based upon 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.

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 1.1.3, 1.1.4 and 1.1.5 are about tail bounds for random scalar variables. Lemma 1.1.6 is about cdf of Gaussian distributions. Finally, Lemma 1.1.7 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*

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

Lemma 1.1.3 (Chernoff bound [**?**])**.** *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 1.1.4 (Hoeffding bound [**?**])**.** *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 1.1.5 (Bernstein inequality [**?**]). *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 1.1.6 (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 1.1.7 (Matrix Bernstein, Theorem 6.1.1 in [**?**]). *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.

≪Tengyu notes: Sanjeev notes:

Suggestion: when introducing usual abstractions like Lipschitz constt, Hessian norm etc. let's relate them concretely to what they mean in context of deep learning (noting that Lipschitz constt is wrt the vector of parameters). Be frank about what these numbers might be for deep learning or even how feasible it is to estimate them. (Maybe that discussion can go in the side bar.)

BTW it may be useful to give some numbers for the empirical liptschitz constt encountered in training.

One suspects that the optimization speed analysis is rather pessimistic.≫

≪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≫

≪Tengyu notes: should we use w or θ in this section?≫ ≪Suriya notes: I remembered that we agreed on w for parameters long time back - did we we go back to theta?≫

2.1 *Gradient descent*

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

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

The gradient descent (GD) algorithm is

$$w_0 = \text{initializaiton}$$
$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

where η is the step size or learning rate.

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

$$f(w) = f(w_t) + \underbrace{\langle \nabla f(w_t), w - w_t \rangle}_{\text{linear in } w} + \cdots$$

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

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

Then, the optimizer of the program above is equal to $w + \delta$ where

$$\delta = -\alpha \nabla f(w_t)$$

for some positive scalar α. ≪Tengyu notes: this fact can be an exercise≫ In other words, to locally minimize the first order approximation of $f(\cdot)$ around w_t, we should move towards the direction $-\nabla f(w_t)$.

2.1.1 *Formalizing the Taylor Expansion*

We will state a lemma that characterizes the descent of function values under GD. We make the assumption that the eigenvalues of $\nabla^2 f(w)$ is bounded between $[-L, L]$ for all w. We call functions satisfying it L-smooth functions. ≪Tengyu notes: missing definition of $\nabla^2 f$ but perhaps it should belong to somewhere else.≫ This allows us to approximate the function using Taylor expansion accurately in the following sense:

$$f(w) \leq f(w_t) + \langle \nabla f(w_t), w - w_t \rangle + \frac{L}{2}\|w - w_t\|_2^2 \tag{2.1}$$

≪Tengyu notes: another exercise≫

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.

Lemma 2.1.1 (Descent Lemma)**.** *Suppose f is L-smooth. Then, if $\eta < 1/(2L)$, we have*

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

The proof uses the Taylor expansion. The main idea is that even using the upper provided by equation (2.1) suffices.

Proof. We have that

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

where the second step follows from Eq. (2.1), and the last step follows from $\eta \leq L/2$. □

≪Tengyu notes: perhaps add a corollary saying that GD "converges" to stationary points≫

2.2 *Stochastic gradient descent*

Motivation: Computing the gradient of a loss function could be expensive. Recall that

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

Computing the gradient $\nabla\widehat{L}(h)$ scales linearly in n. Stochastic gradient descent (SGD) estimates the gradient by sampling a mini-batch of gradients. Especially when the gradients of examples are similar, the estimator can be reasonably accurate. (And even if the estimator is not accurate enough, as long as the learning rate is small enough, the noises averages out across iterations.)

The updates: We simplify the notations a bit for the 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 $i_1, \ldots, i_B$ 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 $\{i_1, \ldots, i_B\}$. The SGD algorithm updates the iterate by

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

2.3 *Accelerated Gradient Descent*

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

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

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 historical gradient and the update of w_t uses all past gradient. 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. It has the advantage of stronger worst case guarantees on convex functions. Both of the two algorithms can be used with stochastic gradient, but little is know about the theoretical guarantees about stochastic accelerate gradient descent.

2.4 *Local Runtime Analysis of GD*

When the iterate is 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 $A = \text{diag}(\lambda_1, \ldots, \lambda_d)$ 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 2.1.1 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 $O(\lambda_d/\lambda_1 \cdot \log(1/\epsilon))$ 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 more optimal if we can 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., one can use $\mathrm{diag}(\nabla f(w) \nabla f(w^\top))$ to approximate the Hessian, and then use the inverse of the diagonal matrix as the pre-conditioner.

≪Tengyu notes: more on adagrad?≫

3
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.

3.1 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 3.1.1. *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 3.1. Suppose node u is a weighted sum of the nodes $z_1, \ldots, z_m$ (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

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

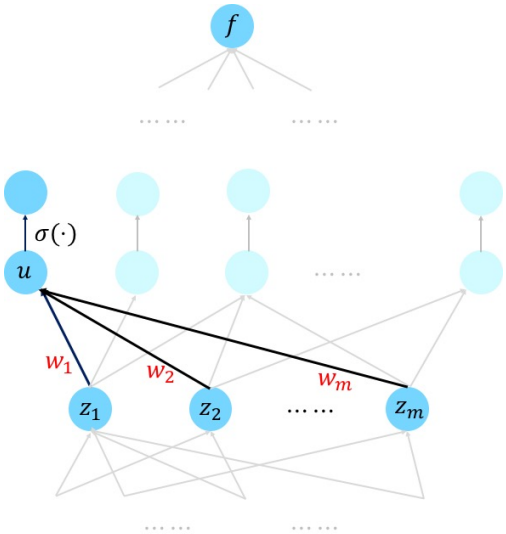


Figure 3.1: Why it suffices to compute derivatives with respect to nodes.

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. □

3.1.1 *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 3.2 we apply it to the same example as we used before but with a different focus and numbering of the nodes.

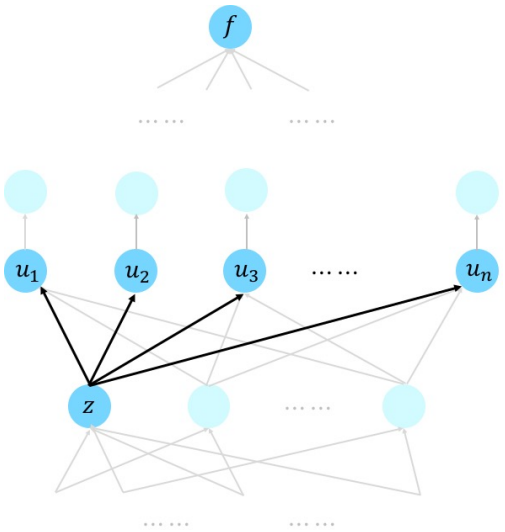


Figure 3.2: Multivariate chain rule: derivative with respect to node z can be computed using weighted sum of derivatives with respect to all nodes that z feeds into.

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.

3.1.2 *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 $O(VE)$.

3.2 *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 1 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 3.2.1. *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,

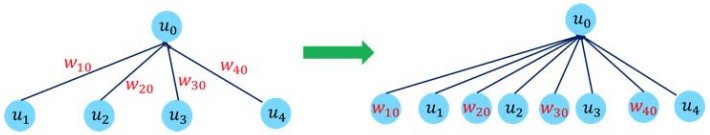
$$S = \sum_{i=1}^{m} \frac{\partial f}{\partial u_i} \frac{\partial u_i}{\partial z} = \frac{\partial f}{\partial z}.$$

This completes the induction and proves the Main Claim. □

3.3 *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 3.1.1 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.

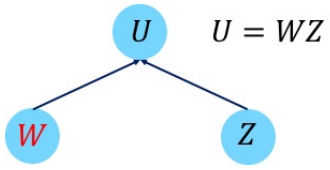


Figure 3.3: Vector version of above

3.4 *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.

3.4.1 Hessian-vector product in linear time: Pearlmutter's 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.

Claim 3.4.1. *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 $O(V + E)$ 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.

4
Basics of generalization theory

Generalization theory gives estimates of the number of training samples sufficient to guarantee that the the test loss of the trained net will be almost as good as the training loss. The classic ideas described in this chapter give very loose estimates. A later chapter in this book describes recent attempts to come up with tighter estimates of sample complexity. Generalization bounds are of interest in understanding mathematically *Why do learning algorithms generalize*?

Generalization theory takes inspiration from an old philosophical principle called *Occam's razor*: given a choice between a simpler theory 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 a 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"?

4.0.1 Occam's razor formalized for ML

In the language of Empirical Risk Minimization from Chapter 1 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	$\leftrightarrow$	Training dataset S
theory	$\leftrightarrow$	hypothesis h
All possible theories	$\leftrightarrow$	hypothesis class $\mathcal{H}$
Finding theory to fit observations	$\leftrightarrow$	Minimize training loss to find $h \in \mathcal{H}$
Theory is good	$\leftrightarrow$	h has low test loss
Simpler theory	$\leftrightarrow$	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 gives us $\widehat{h} = \arg\min \widehat{L}(h)$ where $\widehat{L}$ denotes the training loss. For this chapter we will use $\widehat{L_S}$ to emphasize the training set. Let $L_{\mathcal{D}}(h)$ denote the expected loss of h on the full data distribution $\mathcal{D}$. Then the *generalization error* is defined as $\Delta_S(h) = L_{\mathcal{D}}(h) - \widehat{L_S}(h)$. 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, so we say it *overfitted* to the sample S.

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

[1] This is the format of typical generalization bound!

$$\Delta_S(h) \leq \frac{\mathcal{C}(\mathcal{H}) + O(\log 1/\delta)}{m} + \textit{Sampling error term.} \tag{4.1}$$

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.

4.0.2 *Motivation for generalization theory*

If the experiment has already decided on the architecture, algorithm etc. to use then generalization theory is of very limited use. They can use a held out dataset which is never seen during training. At the end of training, evaluating average loss on this yields a valid estimate for $L_{\mathcal{D}}(h)$ for the trained hypothesis h.

Thus the hope in developing generalization theory is that it provides insights into suggesting architectures and algorithms that lead to good generalization.

4.1 *Some simple bounds on generalization error*

The first one we prove is trivial, but as we shall see is also at the heart of most other generalization bounds (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., the number of bits needed to describe a single hypothesis in this class) functions as a complexity measure.

Theorem 4.1.1 (Simple union bound)**.** *If the loss function takes values in* $[0, 1]$ *and hypothesis class* $\mathcal{H}$ *contains* N *distinct hypotheses then with*

probability at least $1-\delta$

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

Proof. For any fixed hypothesis g imagine drawing a training sample of size m. Then $\widehat{L_S}(g)$ is an average of iid 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. □

Of course, the union bound doesn't apply to deep nets *per se* because the set of hypotheses —even after we have fixed the architecture— consists of all vectors in $\mathbb{R}^k$, where k is the number of real-valued parameters. This is an uncountable set! However, we show it is possible to reason about the set of all nets as a finite set after suitable 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 $\text{Ball}(0,1)$. (Here $\text{Ball}(w,r)$ refers to set of all points in $\mathbb{R}^k$ within distance r of w.) We assume there is a $\rho > 0$ 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 $\gamma > 0$. [2] (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.)

[2] Another way to phrase this assumption in a somewhat stronger form is that the loss on any datapoint is a Lipschitz function of the parameter vector, with Lipschitz constant at most γ/ρ.

Definition 4.1.2 (ρ-cover)**.** *A set of points $w_1, w_2, \ldots \in \mathbb{R}^k$ is a ρ-*cover *in $\mathbb{R}^k$ if for every $w \in \text{Ball}(0,1)$ there is some w_i such that $w \in \text{Ball}(w_i, \rho)$.*

Lemma 4.1.3 (Existence of ρ-cover)**.** *There exists a ρ-cover of size at most $(2/\rho)^k$.*

Proof. The proof is simple but ingenious. 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

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

iterations, which is at most $(2/\rho)^k$ 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 4.1.4 (Generalization bound for normed spaces). [3] *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*

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

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] As you might imagine, this generalization bound via γ-cover is too loose, and gives very pessimistic estimates of what m needs to be.

4.2 Data dependent complexity measures

A *complexity measure* for hypothesis classes is a way to quantify their "complicatedness." It is defined to let us prove an upper bound on the number of training samples needed to keep down the generalization error. Above we implicitly defined two complexity measures: 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.

Theorists then realized that the above 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.

4.2.1 Rademacher Complexity

Rademacher complexity is a complexity measure that depends on data distribution. For simplicity we will assume 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* [4] 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.

$$\mathcal{R}_{m,D}(\mathcal{H}) = \mathop{\mathbb{E}}_{S_1,S_2}\left[\frac{1}{2m}\sup_{h\in\mathcal{H}}\left|\sum_{z\in S_1} l(z,h) - \sum_{z\in S_2} l(z,h)\right|\right], \qquad (4.2)$$

[4] Standard accounts of this often confuse students, or falsely impress them with a complicated proof of Thm 4.2.1. In the standard definition, loss terms are weighted by iid ± 1 random variables. Its value is within is $\pm O(1/\sqrt{m})$ of the one in our definition.

where the expectation is over S_1, S_2 are two iid samples (i.e., multisets) of size m each from the data distribution $\mathcal{D}$. The following theorem relates this to generalization error of the trained hypothesis.

Theorem 4.2.1. *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 $\mathcal{D}$ to get another (multi)set S_1 then with probability at least $1 - \delta$ the loss on these closely approximates $L_{\mathcal{D}}(h)$:

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

Now we notice that S_1, S_2 thus drawn are exactly like the sets drawn in the thought experiment [5] (4.2) and the maximizer h for this expression defined $\mathcal{R}_{m,D}$. So the right hand side is at most

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

□

[5] 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 discrepancy 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 (4.2).

Example: We can 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 $S = (x_1, x_2, x_3, ..x_m)$ (each $x_i \in \mathbb{R}^d$) is $\leq \max_{i \in [m]} \|x_i\|_2 / \sqrt{m}$.

4.2.2 *Alternative Interpretation: Ability to correlate with random labels*

Sometimes 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 (4.2) 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.

4.3 *PAC-Bayes bounds*

These bounds due to McAllester (1999) **[?]** 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 as well as their empirical observation that the sun has risen every day in their lifetime.

Coming back to ML, PAC-Bayes bounds assume that experimenter (i.e. machine learning 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[6] to the posterior Q, meaning now if asked to classify they would pick $h \sim Q$ and use that. Thus the expected training loss is

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

[6] To illustrate PAC-Bayes chain of thought for deep learning, P could be uniform distribution on all deep nets with a certain architecture, and the posterior is the distribution on deep nets obtained by random initialization followed by training on m randomly sampled datapoints using SGD.

Theorem 4.3.1 (PAC-Bayes bound)**.** *Consider a distribution $\mathcal{D}$ on the data. Let P be a prior distribution over hypothesis class $\mathcal{H}$ and $\delta > 0$. Then with probabilty $\geq 1-\delta$, on a i.i.d. sample S of size m from D, for all distributions Q over $\mathcal{H}$ (which could possibly depend on S), we have that*

$$\Delta_S(Q(\mathcal{H})) = \mathop{\mathbb{E}}_{h\sim Q}[L_D(h)] - \mathop{\mathbb{E}}_{h\sim Q}[L_S(h)] \leq \sqrt{\frac{D(Q||P) + \ln(m/\delta)}{2(m-1)}},$$

where $D(Q||P) = \mathbb{E}_{h\sim Q}[\ln \frac{Q(h)}{P(h)}]$ is the so-called KL-divergence[7].

[7] 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.

In other words, generalization error is upper bounded by the square root of the KL-divergence of the distributions (plus some terms that arise from concentration bounds). Thus, in order to minimize the error on the real distribution, we should try to simultaneously minimize the empirical error as well as the KL-divergence between the posterior and the prior. First, lets observe that for a fixed h, using a standard Hoeffdings inequality, we have that

$$\Pr_S[\Delta(h) > \epsilon] \leq e^{-2m\epsilon^2} \tag{4.3}$$

Roughly, this says that $\sqrt{m}\Delta_S(h)$ concentrates at least as strongly as a univariate gaussian. [8] By direct integration over gaussian distribution this also implies that

$$\mathop{\mathbb{E}}_{S}\left[e^{2(m-1)\Delta(h)^2}\right] \leq m$$

and therefore, with high probability over S,

$$e^{2(m-1)\Delta(h)^2} = O(m) \tag{4.4}$$

[8] Low generalization error alone does not imply that h is any good! For example h can have terrible loss on $\mathcal{D}$, which is faithfully captured in the training set!

Now consider the expression (derived by working backwards from statement of the claim)

$$2(m-1)\mathop{\mathbb{E}}_{h\sim Q}[\Delta(h)]^2 - D(Q||P) \leq 2(m-1)\mathop{\mathbb{E}}_{h\sim Q}[\Delta(h)^2] - D(Q||P)$$

where the inequality is by convexity of squares. This in turn is now

$$\begin{aligned}
2(m-1)\mathop{\mathbb{E}}_{h\sim Q}[\Delta(h)^2] - D(Q||P) &= \mathop{\mathbb{E}}_{h\sim Q}\left[2(m-1)\Delta(h)^2 - \ln\frac{Q(h)}{P(h)}\right] \\
&= \mathop{\mathbb{E}}_{h\sim Q}\left[\ln\left(e^{2(m-1)\Delta(h)^2}\frac{P(h)}{Q(h)}\right)\right] \\
&\leq \ln \mathop{\mathbb{E}}_{h\sim Q}\left[\left(e^{2(m-1)\Delta(h)^2}\frac{P(h)}{Q(h)}\right)\right]
\end{aligned}$$

where the last inequality uses Jensen's inequality [9] along with the concavity of ln. Also, since taking expectation over $h \sim Q$ is effectively like summing with a weighting by $Q(h)$, we have [10]

$$\ln \mathop{\mathbb{E}}_{h\sim Q}\left[\left(e^{2(m-1)\Delta(h)^2}\frac{P(h)}{Q(h)}\right)\right] = \ln \mathop{\mathbb{E}}_{h\sim P}\left[\left(e^{2(m-1)\Delta(h)^2}\right)\right]$$

Recapping, we thus have that

$$2(m-1)\mathop{\mathbb{E}}_{h\sim Q}[\Delta(h)]^2 - D(Q||P) \leq \ln\left(\mathop{\mathbb{E}}_{h\sim P}\left[e^{2(m-1)\Delta(h)^2}\right]\right) \tag{4.5}$$

Now using the fact that belief P was fixed before seeing S (i.e., is independent of S):

$$\mathop{\mathbb{E}}_{S}\left[\mathop{\mathbb{E}}_{h\sim P}\left[\, e^{2(m-1)\Delta(h)^2}\right]\right] = \mathop{\mathbb{E}}_{h\sim P}\left[\mathop{\mathbb{E}}_{S}\left[e^{2(m-1)\Delta(h)^2}\right]\right] \leq m.$$

Thus, (1) implies that with high probability over S,

$$\mathop{\mathbb{E}}_{h\sim P}\left[e^{2(m-1)\Delta(h)^2}\right] = O(m) \tag{4.6}$$

Thus, combining the above we get

$$2(m-1)\mathop{\mathbb{E}}_{h\sim Q}[\Delta(h)]^2 - D(Q||P) \leq O(\ln(m))$$

which implies

$$\mathop{\mathbb{E}}_{h\sim Q}[\Delta(h)]^2 \leq \frac{O(\ln(m)) + D(Q||P)}{2(m-1)}$$

Taking the square root on the both sides of the above Equation, then we get

$$\mathop{\mathbb{E}}_{h\sim Q}[\Delta(h)] \leq \sqrt{\frac{O(\ln(m)) + D(Q||P)}{2(m-1)}}$$

Thus, it completes our proof sketch.

[9] Jensen's Inequality: For a concave function f and random variable X, $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$

[10] 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!

5
Advanced Optimization notions

This chapter covers the basic 2nd order method (Newton's method) and then briefly discusses momentum, AdaGrad (as well as AdaDelta/RMSProp) and Adam. Some discussion of attempts to leverage Hessian-vector products and why they don't appear to help.

6
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. [**? ? ? ? ? ? ? ? ? ? ? ?**]) 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 measureâĂİ, 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 6.1.1), 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 6.3.2), 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, **?**] and **?**] 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 [**?** **?**].

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.

6.1 *Linear models in regression: squared loss*

We first demonstrate the algorithmic regularization in a simple 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.

$$L(w) = \sum_{i=1}^{n} \left(w^\top x^{(i)} - y^{(i)} \right)^2. \tag{6.1}$$

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 $\widehat{y}$ [**?**].

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. (6.1) 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 6.1.1. *Consider gradient descent updates w_t for the loss in eq. (6.1) 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. (6.1) $\forall w,\ \nabla L(w) = \sum_i (w^\top x^{(i)} - y^{(i)}) x^{(i)} \in \text{span}(\{x^{(i)}\})$ - 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, **?**] 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;**?**], **?**] and **?**] study how the size of the mini-batches used in SGD influences generalization; and **?**] 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.

6.1.1 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.

$$w_{t+1} = \operatorname*{argmin}_{w} \langle w, \nabla L(w_t) \rangle + \frac{1}{2\eta} \|w - w_t\|_2^2. \tag{6.2}$$

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 ψ [**?** **?**] and steepest descent w.r.t. general norms [**?**].

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

$$\begin{aligned} w_{t+1} &= \arg\min_{w} \eta \, \langle w, \nabla L(w_t) \rangle + D_\psi(w, w_t), \\ \implies \nabla\psi(w_{t+1}) &= \nabla\psi(w_t) - \eta \nabla L(w_t) \end{aligned} \tag{6.3}$$

where $D_\psi(w, w') = \psi(w) - \psi(w') - \langle \nabla\psi(w'), w - w' \rangle$ is the *Bregman divergence* [**?**] w.r.t. ψ. This family captures updates where the geometry is specified by the Bregman divergence D_ψ. Examples of potentials ψ for mirror descent include the squared ℓ_2 norm $\psi(w) = 1/2\|w\|_2^2$, which leads to gradient descent; the entropy potential $\psi(w) = \sum_i w[i] \log w[i] - w[i]$; the spectral entropy for matrix valued w, where $\psi(\mathrm{w})$ is the entropy potential on the singular values of w; general quadratic potentials $\psi(w) = 1/2\|w\|_D^2 = 1/2\, w^\top D w$ for any positive definite matrix D; and the squared ℓ_p norms for $p \in (1, 2]$.

From eq. (6.3), we see that rather than w_t (called primal iterates), it is the $\nabla\psi(w_t)$ (called dual iterates) that are constrained to the low dimensional data manifold $\nabla\psi(w_0) + \mathrm{span}(\{x^{(i)}\})$. The arguments for gradient descent can now be generalized to get the following result.

Theorem 6.1.2. *For any realizable dataset* $\{x^{(i)}, y^{(i)}\}_{n=1}^N$*, and any strongly convex potential* ψ*, consider the mirror descent iterates* w_t *from eq.* (6.3) *for minimizing the empirical loss* $L(w)$ *in eq.* (6.1)*. For all initializations* w_0*, if the step-size schedule minimzes* $L(w)$*, i.e.,* $L(w_t) \to 0$*, then the asymptotic solution of the algorithm is given by*

$$w_t \to \operatorname*{arg\,min}_{w : \forall i, w^\top x^{(i)} = y^{(i)}} D_\psi(w, w_0). \tag{6.4}$$

In particular, if we start at $w_0 = \arg\min_{\mathrm{w}} \psi(w)$ (so that $\nabla\psi(w_0) = 0$), then we get to $\arg\min_{w \in \mathcal{G}} \psi(w)$. [1]

[1] The analysis of Theorem 6.1.2 and Proposition 6.1.1 also hold when instancewise stochastic gradients are used in place of $\nabla L(w_t)$.

Steepest descent w.r.t. general norms Gradient descent is also a special case of steepest descent (SD) w.r.t a generic norm $\|.\|$ [**?**] with

updates given by,

$$w_{t+1} = w_t + \eta_t \Delta w_t, \text{ where } \Delta w_t = \arg\min_v \langle \nabla L(w_t), v \rangle + \frac{1}{2}\|v\|^2. \quad (6.5)$$

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. (6.5) is not uniquely defined and there could be multiple direction Δw_t that minimize eq. (6.5). In such cases, any minimizer of eq. (6.5) 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. 6.5 is equivalent to mirror descent with $\psi(w) = 1/2\|w\|_D^2$. 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, **?**] 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 [**?**]. 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) [**?**]. 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 6.1 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

$\arg\min_{w \in \mathcal{G}} \|w - w_0\|$.

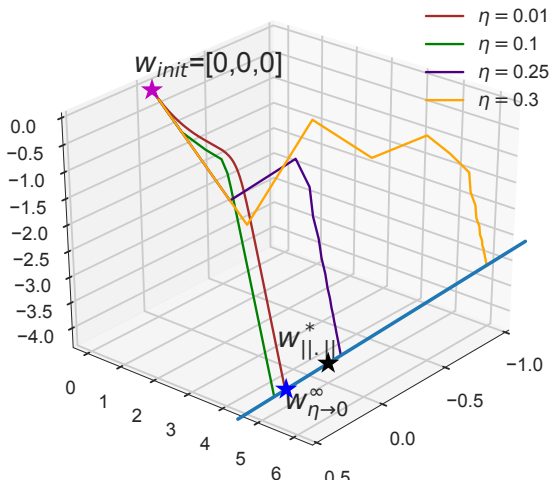


Figure 6.1: **Steepest descent w.r.t $\|.\|_{4/3}$:** the global minimum to which steepest descent converges to depends on η. Here $w_0 = [0, 0, 0]$, $w^*_{\|.\|} = \arg\min_{\psi \in G} \|w\|_{4/3}$ denotes the minimum norm global minimum, and $w^{\infty}_{\eta \to 0}$ denotes the solution of infinitesimal SD with $\eta \to 0$. Note that even as $\eta \to 0$, the expected characterization does not hold, i.e., $w^{\infty}_{\eta \to 0} \neq w^*_{\|.\|}$.

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 6.3.2, we study logistic like strictly monotonic losses used in classification, where we *can* get a characterization for steepest descent.

6.1.2 *Geometry induced by parameterization of model class*

In many learning problems, the same model class can be parameterized in multiple ways. 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 $u, v \in \mathbb{R}^d$ with $f_{u,v}(x) = (u \cdot v)^\top x$ or $f_{u,v}(x) = (u^2 - v^2)^\top x$. All such equivalent parameterizations lead to equivalent training objectives, however, in overparemterized models, using gradient descent on different parameterizations lead to different induced biases in the function space. For example, **? ?**] 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.

6.2 *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?≫

6.3 *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.

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. (6.1), 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 $\bar{w}_\infty = \lim_{t\to\infty} \frac{w_t}{\|w_t\|}$ 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 **?**] and **?**].

$$L(w) = \sum_{i=1}^{n} \exp(-y^{(i)} w^\top x^{(i)}). \tag{6.6}$$

6.3.1 *Gradient Descent*

?] 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 6.3.2. Below is a simpler statement and with a heuristic proof sketch intended to convey the intuition for such results.

Theorem 6.3.1. *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. (6.6), 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*

$$\widehat{w} = \operatorname*{argmin}_{w} \|w\|^2 \text{ s.t. } \forall i,\ w^\top x^{(i)} y^{(i)} \geq 1. \tag{6.7}$$

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:

$$\begin{aligned} -\nabla \mathcal{L}(w) &= \sum_{i=1}^{n} \exp\left(-w^\top x^{(i)}\right) x^{(i)} \\ &= \sum_{i=1}^{n} \exp\left(-g(t) \mathrm{w}_\infty^\top x^{(i)}\right) \exp\left(-\rho(t)^\top x^{(i)}\right) \mathbf{x}_n \,. \end{aligned} \tag{6.8}$$

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. 6.7). Making these intuitions rigorous constitutes the bulk of the proof in **?**], which uses a proof technique very different from that in the following section (Section 6.3.2).

6.3.2 *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. (6.5). The optimality condition of Δw_t in eq. (6.5) requires

$$\langle \Delta w_t, -\nabla L(w_t)\rangle = \|\Delta w_t\|^2 = \|\nabla L(w_t)\|_\star^2, \tag{6.9}$$

where $\|x\|_\star = \sup_{\|y\|\le 1} x^\top y$ 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. (6.5) is not uniquely defined and there could be multiple direction Δw_t that minimize eq. (6.5). In such cases, any minimizer of eq. (6.5) is a valid steepest descent update and satisfies eq. (6.9).

In the preliminary result in Theorem 6.3.1, 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 6.3.2. *For any separable dataset $\{x_i, y_i\}_{i=1}^n$ and any norm $\|\cdot\|$, consider the steepest descent updates from eq. (6.9) for minimizing $L(w)$ in eq. (6.6) with the exponential loss $\ell(u,y) = \exp(-uy)$. For all initializations w_0, and all bounded step-sizes satisfying $\eta_t \le \min\{\eta_+, \frac{1}{B^2 L(w_t)}\}$, where $B := \max_n \|x_n\|_\star$ and $\eta_+ < \infty$ is any finite number, the iterates w_t satisfy the following,*

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

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

A special case of Theorem 6.3.2 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 [**?**], 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 [**?**], and later on AdaBoost with small enough step-size was shown to converge in direction precisely to the maximum ℓ_1 margin solution [**? ? ?**], just as guaranteed by Theorem 6.3.2. In fact, **?**] 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 [**?**], hence the bounded step-sizes rule in Theorem 6.3.2.

Theorem 6.3.2 is a generalization of the result of **?** to steepest descent with respect to other norms, and our proof follows the same strategy as **?** . We first prove a generalization of the duality result of **?**]: 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 6.3.2. 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 6.3.3. *Gradient domination condition (Lemma 10 of [**?**])*
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 6.3.4. *(Lemma 11 and 12 of **?**]) Consider the steepest descent iterates w_t on (6.6) 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 6.3.2. **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,

$$
\begin{aligned}
&L(w_{t+1}) \leq \\
&L(w_t) + \eta_t \langle \nabla L(w_t), \Delta w_t \rangle + \sup_{\beta \in (0,1)} \frac{\eta_t^2}{2} \Delta w_t{}^\top \nabla^2 L\left(w_t + \beta \eta_t \Delta w_t\right) \Delta w_t \\
&\overset{(a)}{\leq} L(w_t) - \eta_t \|\nabla L(w_t)\|_\star^2 + \frac{\eta_t^2 B^2}{2} \sup_{\beta \in (0,1)} L\left(w_t + \beta \eta_t \Delta w_t\right) \|\Delta w_t\|^2 \\
&\overset{(b)}{\leq} L(w_t) - \eta_t \|\nabla L(w_t)\|_\star^2 + \frac{\eta_t^2 B^2}{2} L(w_t) \|\Delta w_t\|^2
\end{aligned}
\tag{6.10}
$$

where (a) uses $v^\top \nabla^2 L(w) v \leq L(w) B^2 \|v\|^2$ and (b) uses Proposition 6.3.4 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 . 6.10, using $\gamma_t = \|\nabla L(w_t)\|_\star = \|\Delta w_t\|$, we have that

$$
\begin{aligned}
L(w_{t+1}) &\leq L(w_t) - \eta \gamma_t^2 + \frac{\eta^2 B^2 L(w_t) \gamma_t^2}{2} \\
&= L(w_t) \left[1 - \frac{\eta \gamma_t^2}{L(w_t)} + \frac{\eta^2 B^2 \gamma_t^2}{2}\right] \\
&\overset{(a)}{\leq} L(w_t) \exp\left(-\frac{\eta \gamma_t^2}{L(w_t)} + \frac{\eta^2 B^2 \gamma_t^2}{2}\right) \\
&\overset{(b)}{\leq} L(w_0) \exp\left(-\sum_{u \leq t} \frac{\eta_u \gamma_u^2}{L(w_u)} + \sum_{u \leq t} \frac{\eta^2 B^2 \gamma_u^2}{2}\right),
\end{aligned}
\tag{6.11}
$$

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. (6.11), we have,

$$
\begin{aligned}
\max_{n \in [N]} \exp(-\langle w_{t+1}, x_n \rangle) &\leq L(w_{(t+1)}) \\
&\leq L(w_0) \exp\left(-\sum_{u \leq t} \frac{\eta \gamma_u^2}{L(w_u)} + \sum_{u \leq t} \frac{\eta^2 B^2 \gamma_u^2}{2}\right)
\end{aligned}
\tag{6.12}
$$

By applying $-\log$,

$$\min_{n\in[N]} \langle w_{t+1}, x_n \rangle \geq \sum_{u\leq t} \frac{\eta\gamma_u^2}{L(w_u)} - \sum_{u\leq t} \frac{\eta^2 B^2 \gamma_u^2}{2} - \log L(w_0). \tag{6.13}$$

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

$$\|w_{t+1}\| \leq \|w_0\| + \sum_{u\leq t} \eta \|\Delta w_u\| \leq \|w_0\| + \sum_{u\leq t} \eta\gamma_u. \tag{6.14}$$

To complete the proof, we simply combine Equations (6.13) and (6.14) to lower bound the normalized margin.

$$\begin{aligned} \frac{\langle w_{t+1}, x_n \rangle}{\|w_{t+1}\|} \geq \frac{\sum_{u\leq t} \frac{\eta\gamma_u^2}{L(w_u)}}{\sum_{u\leq t} \eta\gamma_u + \|w_0\|} &- \left(\frac{\sum_{u\leq t} \frac{\eta^2 B^2 \gamma_u^2}{2} + \log L(w_0)}{\|w_{t+1}\|} \right). \\ := (I) &\quad +(II). \end{aligned} \tag{6.15}$$

For term (I), from Proposition 6.3.3, 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

$$\frac{\sum_{u\leq t} \frac{\eta\gamma_u^2}{L(w_u)}}{\sum_{u\leq t} \eta\gamma_u + \|w_0\|} \geq \gamma \frac{\sum_{u\leq t} \eta\gamma_u}{\sum_{u\leq t} \eta\gamma_u + \|w_0\|} \to \gamma, \tag{6.16}$$

using $\sum_{u\leq t} \eta\gamma_u \to \infty$ and $\|w_0\| < \infty$ from Proposition 6.3.4.

For term (II), $\log L(w_0) < \infty$ and $\sum_{u\leq t} \frac{\eta^2 B^2 \gamma_u^2}{2} < \infty$ using Proposition 6.3.3. Thus $(II) \to 0$.

Using the above in Equation (6.15), we obtain

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

□

6.4 *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

$$L(w) = \sum_{i=1}^n \exp(-y_i f_i(w)), \tag{6.17}$$

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 **??**, 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 **??**, we expect that gradient descent on Equation (6.17) 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 6.4.1 (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 [2].

[2]

Assumption 6.4.2 (Smoothness)**.** *We assume* $f_i(w)$ *is a* C^2 *function.*

Assumption 6.4.3 (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,*

$$\ell_{nt} = h_t a_n + h_t \epsilon_{nt} \tag{6.18}$$

$$w_t = g_t \bar{w} + g_t \delta_t, \tag{6.19}$$

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 6.4.4 (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 6.4.1 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 6.4.5. *Define $\bar{w} = \lim_{t\to\infty} \frac{w_t}{\|w_t\|}$. Under Assumptions 6.4.2, 6.4.3, and 6.4.4, $\bar{w} \in \mathcal{W}$ 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 6.4.6. *Under the setting of Theorem 6.4.5,*

$$\nabla f_i(w_t) = \nabla f_i(g_t\bar{w}) + O(Bg_t^{\alpha-1}\|\delta_t\|). \tag{6.20}$$

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 6.4.7. *Under the conditions of Theorem 6.4.5, $a_i = 0$ 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 6.4.7) ,

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

Via Assumption 6.4.4, term $I = \Omega(g_t^{\alpha-1}h_t)$ and from Lemma 6.4.6 , $II = o(I)$. Using these, the first term I is the largest and so after normalizing,

$$\frac{\dot{w}_t}{\|\dot{w}_t\|} = \sum_{i\in S} a_i \nabla f_i(g_t\bar{w}) + o(1). \tag{6.21}$$

Since $\lim_t \frac{w_t}{\|w_t\|} = \lim_t \frac{\tilde{w}_t}{\|w_t\|}$ [**?**], then

$$\lim_{t\to\infty} \frac{w_t}{\|w_t\|} = \sum_{i\in S} \nabla f_i(g_t \bar{w}). \tag{6.22}$$

Thus we have shown w satisfies the first-order optimality condition of Definition 6.4.1. □

6.5 *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?≫

7
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

$$\forall \alpha \in [0,1], w, w', f(\alpha w + (1-\alpha)w') \leq \alpha f(w) + (1-\alpha) f(w'). \quad (7.1)$$

$$\forall w, w', f(w') \geq f(w) + \langle \nabla f(w), w' - w \rangle. \quad (7.2)$$

These equations characterize important geometric properties of the objective function $f(w)$. In particular, Equation (7.1) 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 (7.2) 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 (7.1)) or connects local properties with global optimality (such as Equation (7.2)).

7.1 Preliminaries and challenges in nonconvex landscapes

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

Definition 7.1.1 (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 $f(w^*) \le f(w)$. A point w is a* local minimum/maximum *if there exists a radius $\epsilon > 0$ such that for every $\|w' - w\|_2 \le \epsilon$, we have $f(w) \le f(w')$ ($f(w) \ge f(w')$ 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.

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

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

Definition 7.1.2 (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 7.1.3 (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 whether a point is a saddle point, local minimum or local maximum based on its Hessian.

Claim 7.1.4. *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].

[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.

Flat regions Even if a function does not have any spurious local minima or saddle point, it can still be nonconvex, see Figure 7.1. 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 7.2.

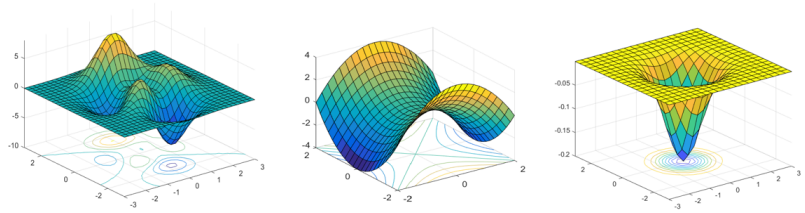


Figure 7.1: Obstacles for non-convex optimization. From left to right: local minimum, saddle point and flat region.

7.2 *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.

The main idea here is to identify properties of the objective and also a *potential function*, such that the potential function keeps de-

creasing as we run simple optimization algorithms such as gradient descent. Many properties were used in previous literature, including

Definition 7.2.1. *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 7.2.2. *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].*

[3] Polyak-Lojasiewicz and RSI requires standard smoothness definition as in Equation (2.1), weakly-quasi-convex requires a special smoothness property detailed in [**?**].

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

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.

7.2.1 Generalized linear model

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

$$y = \sigma(w_*^\top x) + \epsilon. \tag{7.3}$$

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

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

$$L(w) = \frac{1}{2} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[(y - \sigma(w^\top x)^2 \right]. \tag{7.4}$$

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 (7.3)), 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.

7.2.2 *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:

$$\nabla g(w) = \mathop{\mathbb{E}}_{x,y}\left[(\sigma(w^\top x) - y)x\right] = \mathop{\mathbb{E}}_{x}\left[(\sigma(w^\top x) - \sigma(w_*^\top x))x\right]. \quad (7.5)$$

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 (7.5). 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]

7.3 *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*.

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

Definition 7.3.1 (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 o gradient). The optimization algorithms can be used to find an approximate second order stationary point:

Definition 7.3.2 (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 7.3.3 (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 [**? ? ?**], matrix completion [**? ?**], dictionary learning [**?**], phase retrieval [**?**], tensor decomposition [**?**], synchronization problems [**?**] and certain objective for two-layer neural network [**?**] are known to be locally optimizable.

7.4 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].

[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.

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:

$$\min_{x\in\mathbb{R}^d} f(x) := \frac{1}{4}\|M - xx^\top\|_F^2. \tag{7.6}$$

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.

7.4.1 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 (7.6), 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

perturbation):

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

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]

$$\nabla f(x) = (xx^\top-M)x, \quad \nabla^2 f(x) = \|x\|_2^2 I+2xx^\top-M. \tag{7.7}$$

[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 $\delta^\top[\nabla^2 f(x)]\delta$.

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).

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 7.4.1 (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.

7.4.2 *Finding directions of improvements*

The approach in Section 7.4.1 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 7.4.1 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 7.4.2 (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 (7.6) becomes

$$\min_x f(x) = \frac{1}{4}\|zz^\top - xx^\top\|_F^2. \tag{7.8}$$

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 $M = xx^\top$.

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

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 7.4.3. *For objective function (7.8), 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 7.4.4. *For objective function (7.8), 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 (7.7), except that M is now $zz^\top$. First, since x is not a direction of improvement, we must have

$$\langle \nabla f(x), x\rangle = 0 \implies \|x\|_2^4 = \langle x, z\rangle^2. \tag{7.9}$$

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 (7.9) 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 (7.9) 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 7.4.3 is very similar to Lemma 7.4.4, except we need to allow slacks in every equation and inequality we use. The additional benefit of having the more robust Lemma 7.4.3 is that the

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 7.4.4 and Lemma 7.4.3 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.

8
Ultra-wide Neural Networks and Neural Tangent Kernel

Training a neural network is a non-convex optimization problem, and in the worst case, it is NP-hard [**?**]. 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 even the original labels are replaced with random labels [**?**].

A widely believed explanation for this surprising phenomenon is that the neural network is over-parameterized. For example, Wide ResNet uses 100x parameters than the number of training data. Thus there must exist one such neural network of this architecture that can fit all training data. However, theoretically the existence does not imply that the network found by a *randomly initialized* gradient method can fit all the data.

Over-parameterization also brings new theoretical challenge in generalization. Traditional generalization bounds often require the number of parameters is much smaller than the number of data points where in the over-parameterized regime, these bounds become vacuous.

This chapter relates a neural network trained by *randomly initialized gradient descent* with the kernel method with particular kernel function: *Neural Tangent Kernel* (NTK) which as first proposed by Jacot, Gabriel and Hongler [**?**]. In the following, we will describe how does NTK arise and use NTK to explain the optimization and generalization behavior of over-parameterized neural networks. At last, we will also discuss some practical usages of NTK.

8.1 Evolving Equation on Predictions

NTK arises from the dynamics of the predictions on training data points. 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 dynamics can be described by an ordinary differential equation (ODE):

$$\frac{dw(t)}{dt} = -\nabla \ell(w(t)).$$

Note this the dynamics on the parameters. The following lemma describes the dynamics of the predictions on training data points.

Lemma 8.1.1. *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 $\left\langle \frac{\partial f(w(t),x_i)}{\partial w}, \frac{\partial f(w(t),x_j)}{\partial w} \right\rangle$:*

$$\frac{du(t)}{dt} = -H(t) \cdot (u(t) - y). \tag{8.1}$$

Proof of Lemma 8.1.1. The parameters w evolve according to the differential equation

$$\frac{dw(t)}{dt} = -\nabla \ell(w(t)) = -\sum_{i=1}^{n} \left(f(w(t), x_i) - y_i\right) \frac{\partial f(w(t), x_i)}{\partial w}, \tag{8.2}$$

where $t \geq 0$ is a continuous time index. Under Equation (8.2), the evolution of the network output $f(w(t), x_i)$ can be written as

$$\frac{df(w(t), x_i)}{dt} = -\sum_{j=1}^{n} (f(w(t), x_j) - y_j) \left\langle \frac{\partial f(w(t), x_i)}{\partial w}, \frac{\partial f(w(t), x_j)}{\partial w} \right\rangle. \tag{8.3}$$

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 (8.3) can be written more compactly as

$$\frac{du(t)}{dt} = -H(t) \cdot (u(t) - y), \tag{8.4}$$

where $H(t) \in \mathbb{R}^{n\times n}$ is a kernel matrix defined as $[H(t)]_{i,j} = \left\langle \frac{\partial f(w(t),x_i)}{\partial w}, \frac{\partial f(w(t),x_j)}{\partial w} \right\rangle$ $(\forall i, j \in [n])$. □

The statement of Lemma 8.1.1 involves a matrix $H(t)$. Below we define a neural network architecture whose width is allowed to go to infinity, while fixing the training data as above. In the limit, it can be shown that the matrix $H(t)$ remains *constant* during training i.e.,

equal to $H(0)$. Moreover, under a random initialization of parameters, the random matrix $H(0)$ converges in probability to a certain deterministic kernel matrix H^* as the width goes to infinity, which is the *Neural Tangent Kernel* $k(\cdot,\cdot)$ evaluated on the training data. If $H(t) = H^*$ for all t, then Equation (8.1) becomes

$$\frac{du(t)}{dt} = -H^* \cdot (u(t) - y). \tag{8.5}$$

Note that the above dynamics is identical to the dynamics of *kernel regression* under gradient flow, for which at time $t \to \infty$ the final prediction function is (assuming $u(0) = 0$)

$$f^*(x) = (k(x, x_1), \ldots, k(x, x_n)) \cdot (H^*)^{-1} y. \tag{8.6}$$

8.2 Coupling Ultra-wide Neural Networks and NTK

In this section, we consider a simple two-layer neural network of the following form:

$$f(a, W, x) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma\left(w_r^\top x\right) \tag{8.7}$$

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. [1] We also assume all any input x has Euclidean norm 1, $\|x\|_2 = 1$. The scaling $1/\sqrt{m}$ will play an important role in proving $H(t)$ is close to the fixed H^* kernel. Throughout the section, to measure the closeness of two matrices A and B, we use the operator norm $\|\cdot\|_2$.

[1] Note rectified linear unit (ReLU) activation function does not satisfy this assumption. However, one can use a specialized analysis of ReLU to show $H(t) \approx H^*$ [**?**].

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., $W = [w_1, \ldots, w_m]$. 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 $\frac{\partial f(a,W,x_i)}{\partial w_r} = \frac{1}{\sqrt{m}} a_r x_i \dot{\sigma}\left(w_r^\top x_i\right)$. Therefore, each entry of $H(0)$ admits the formula

$$\begin{aligned}
[H(0)]_{ij} &= \sum_{r=1}^{m} \left\langle \frac{\partial f(a, W(0), x_i)}{\partial w_r(0)}, \frac{\partial f(a, W(0), x_j)}{\partial w_r(0)} \right\rangle \\
&= \sum_{r=1}^{m} \left\langle \frac{1}{\sqrt{m}} a_r x_i \dot{\sigma}\left(w_r(0)^\top x_i\right), \frac{1}{\sqrt{m}} a_r x_j \dot{\sigma}\left(w_r(0)^\top x_i\right) \right\rangle \\
&= x_i^\top x_j \cdot \frac{\sum_{r=1}^{m} \dot{\sigma}\left(w_r(0)^\top x_i\right) \dot{\sigma}\left(w_r(0)^\top x_j\right)}{m}
\end{aligned}$$

Here the last step we used $a_r^2 = 1$ for all $r = 1, \ldots, m$ because we initialize $a_r \sim \text{Unif}[\{-1, 1\}]$. Recall every $w_r(0)$ is i.i.d. sampled from 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:

$$H_{ij}^* \triangleq x_i^\top x_j \cdot \mathbb{E}_{w\sim N(0,I)}\left[\dot{\sigma}\left(w^\top x_i\right)\dot{\sigma}\left(w^\top x_j\right)\right]$$

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 8.2.1 (Perturbation on the Initialization)**.** *Fix some $\epsilon > 0$. If $m = \Omega\left(\frac{n^4\log(n/\delta)}{\epsilon^2}\right)$, then with probability at least $1-\delta$ over $w_1(0),\ldots,w_m(0)$, we have*

$$\|H(0)-H^*\|_2 \le \epsilon.$$

Proof of Lemma 8.2.1. We first fixed an entry (i,j). Note

$$\left|x_i^\top x_j \dot{\sigma}\left(w_t(0)^\top x_i\right)\dot{\sigma}\left(w_r(0)^\top x_j\right)\right| \le 1.$$

Applying Hoeffding inequality, we have with probability $1-\frac{\delta}{n^2}$,

$$\left|[H(0)]_{ij} - H_{ij}^*\right| \le \frac{\epsilon}{n^2}.$$

Next, applying the union bound over all pairs $(i,j)\in[n]\times[n]$, we have for all (i,j), $\left|[H(0)]_{ij}-H_{ij}^*\right| \le \frac{\epsilon}{n^2}$. To establish the operator norm bound, we simply use the following chain of inequalities

$$\|H(0)-H^*\|_2 \le \|H(0)-H^*\|_F \le \sum_{ij}\left|[H(0)]_{ij}-H_{ij}^*\right| \le n^2\cdot\frac{\epsilon}{n^2} = \epsilon.$$

□

Now we proceed to show during training, $H(t)$ is close to $H(0)$. Formally, we prove the following lemma.

Lemma 8.2.2. *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 $m=\Omega\left(\frac{n^6t^2}{\epsilon^2}\right)$, we have*

$$\|H(t)-H(0)\|_2 \le \epsilon.$$

Proof of Lemma 8.2.2. 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.

$$\begin{aligned}
\|w_r(t) - w_r(0)\|_2 =& \left\| \int_0^t \frac{dw_r(\tau)}{d\tau} d\tau \right\|_2 \\
=& \left\| \int_0^t \frac{1}{\sqrt{m}} \sum_{i=1}^n (u_i(\tau) - y_i) a_r x_i \dot{\sigma} \left(w_r(\tau)^\top x_i \right) d\tau \right\|_2 \\
\leq& \frac{1}{\sqrt{m}} \int \left\| \sum_{i=1}^n (u_i(\tau) - y_i) a_r x_i \dot{\sigma} \left(w_r(\tau)^\top x_i \right) \right\|_2 d\tau \\
\leq& \frac{1}{\sqrt{m}} \sum_{i=1}^n \int_0^t \left\| u_i(\tau) - y_i a_r x_i \dot{\sigma} \left(w_r(\tau)^\top x_i \right) \right\|_2 d\tau \\
\leq& \frac{1}{\sqrt{m}} \sum_{i=1}^n \int_0^t O(1) d\tau \\
=& O\left(\frac{tn}{\sqrt{m}} \right).
\end{aligned}$$

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 8.2.1, 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 [**?**].

Remark 3: One can generalize the proof for multi-layer neural network. See [**?**] 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 [**?**].

8.3 Explaining Optimization and Generalization of Ultra-wide Neural Networks via NTK

Now we have established the following approximation

$$\frac{du(t)}{dt} \approx -H^* \cdot (u(t) - y) \tag{8.8}$$

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

$$\begin{aligned}\frac{dv_i^\top u(t)}{dt} =& -v_i^\top H^* \cdot (u(t) - y) \\ =& -\lambda_i \left(v_i^\top (u(t) - y)\right).\end{aligned}$$

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

$$v_i^\top (u(t) - y) = \exp(-\lambda_i t) \left(v_i^\top (u(0) - y)\right). \tag{8.9}$$

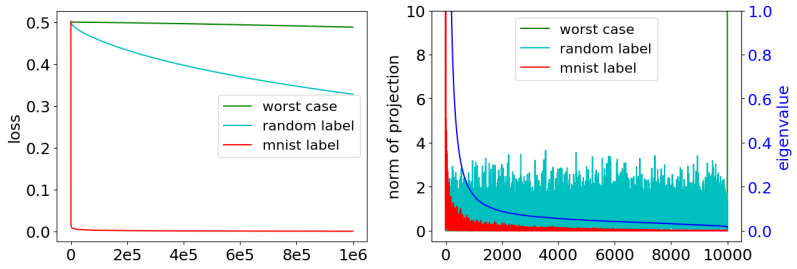


Figure 8.1: Convergence rate vs. projections onto eigenvectors of the kernel matrix.

Now we use Equation (8.9) 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 [**? ?**].

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 (8.9) implies that each component of this difference, i.e., $v_i^\top (u(t) - y)$ 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 $(u(t) - y) = \sum_{i=1}^n v_i^\top (u(t) - y)$. Since we know each $v_i^\top (u_i(t) - y) \to 0$, we can conclude that $(u(t) - y) \to 0$ as well.

Equation (8.9) 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 of y onto the top eigenvectors to be larger.[2] 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 8.1, 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 (8.7) 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

[2] Here we ignore the effect of $u(0)$ for simplicity. See [**?**] on how to mitigate the effect on $u(0)$.

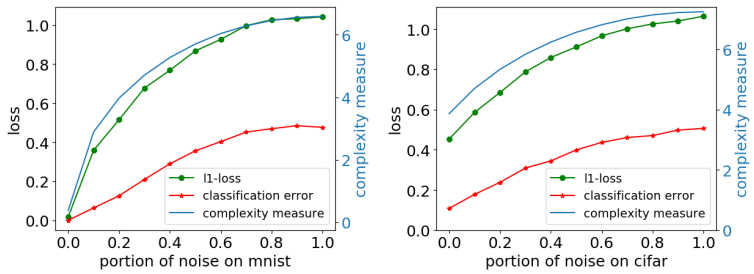


Figure 8.2: Generalization error vs. complexity measure.

that original labels have much better alignment with top eigenvectors, thus enjoying faster convergence.

Understanding Generalization of Ultra-wide Neural Networks The approximation in Equation (8.8) implies the final prediction function of ultra-wide neural network is approximately the kernel prediction function defined in Equation (8.6). 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 (8.6), 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):

$$\frac{\sqrt{2y^\top (H^*)^{-1} y \cdot \mathrm{tr}(H^*)}}{n}. \tag{8.10}$$

This is a *data-dependent* complexity measure that upper bounds the generalization error.

We can check this complexity measure empirically. In Figure 8.2, 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 (8.7) 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.

8.4 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 $h = 1, 2, \ldots, L$:

$$f^{(h)}(x) = W^{(h)} g^{(h-1)}(x) \in \mathbb{R}^{d_h}, g^{(h)}(x) = \sqrt{\frac{c_\sigma}{d_h}} \sigma\left(f^{(h)}(x)\right) \in \mathbb{R}^{d_h} \tag{8.11}$$

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, 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 (8.11) ensures that the norm of $g^{(h)}(x)$ for each $h \in [L]$ is approximately preserved at initialization (see [**?**]). 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 8.1.1 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

$$b^{(h)}(x) = \begin{cases} 1 \in \mathbb{R}, & h = L+1, \\ \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x) \left(W^{(h+1)}\right)^\top b^{(h+1)}(x) \in \mathbb{R}^{d_h}, & h = 1, \ldots, L, \end{cases} \tag{8.12}$$

$$D^{(h)}(x) = \operatorname{diag}\left(\dot{\sigma}\left(f^{(h)}(x)\right)\right) \in \mathbb{R}^{d_h \times d_h}, \qquad h = 1, \ldots, L. \tag{8.13}$$

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 $h \in [L]$,

$$\begin{aligned}
&\Sigma^{(0)}(x, x') = x^\top x', \\
&\Lambda^{(h)}(x, x') = \begin{pmatrix} \Sigma^{(h-1)}(x, x) & \Sigma^{(h-1)}(x, x') \\ \Sigma^{(h-1)}(x', x) & \Sigma^{(h-1)}(x', x') \end{pmatrix} \in \mathbb{R}^{2\times 2}, \\
&\Sigma^{(h)}(x, x') = c_\sigma \mathbb{E}_{(u,v)\sim\mathcal{N}\left(0,\Lambda^{(h)}\right)} \left[\sigma(u)\sigma(v)\right].
\end{aligned} \tag{8.14}$$

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

$$\begin{aligned}
&\mathbb{E}\left[\left[f^{(h+1)}(x)\right]_i \cdot \left[f^{(h+1)}(x')\right]_i \middle| f^{(h)}\right] \\
&= \left\langle g^{(h)}(x), g^{(h)}(x') \right\rangle \\
&= \frac{c_\sigma}{d_h} \sum_{j=1}^{d_h} \sigma\left(\left[f^{(h)}(x)\right]_j\right) \sigma\left(\left[f^{(h)}(x')\right]_j\right),
\end{aligned} \tag{8.15}$$

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 (8.14).

Next we deal with the second term $\left\langle b^{(h)}(x), b^{(h)}(x') \right\rangle$. From Equation (8.12) we get

$$\begin{aligned}
&\left\langle b^{(h)}(x), b^{(h)}(x') \right\rangle \\
&= \left\langle \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x) \left(W^{(h+1)}\right)^\top b^{(h+1)}(x), \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x') \left(W^{(h+1)}\right)^\top b^{(h+1)}(x') \right\rangle.
\end{aligned} \tag{8.16}$$

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.)

$$\begin{aligned}
&\left\langle \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x) \left(W^{(h+1)}\right)^\top b^{(h+1)}(x), \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x') \left(W^{(h+1)}\right)^\top b^{(h+1)}(x') \right\rangle \\
\approx &\left\langle \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x) \left(\widetilde{W}^{(h+1)}\right)^\top b^{(h+1)}(x), \sqrt{\frac{c_\sigma}{d_h}} D^{(h)}(x') \left(\widetilde{W}^{(h+1)}\right)^\top b^{(h+1)}(x') \right\rangle \\
\to &\frac{c_\sigma}{d_h} \mathrm{tr} D^{(h)}(x) D^{(h)}(x') \left\langle b^{(h+1)}(x), b^{(h+1)}(x') \right\rangle \\
\to &\dot{\Sigma}^{(h)}(x, x') \left\langle b^{(h+1)}(x), b^{(h+1)}(x') \right\rangle.
\end{aligned}$$

Applying this approximation inductively in Equation (8.16), we get

$$\left\langle b^{(h)}(x), b^{(h)}(x') \right\rangle \to \prod_{h'=h}^{L} \dot{\Sigma}^{(h')}(x, x').$$

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:

$$\Theta^{(L)}(x, x') = \sum_{h=1}^{L+1} \left(\Sigma^{(h-1)}(x, x') \cdot \prod_{h'=h}^{L+1} \dot{\Sigma}^{(h')}(x, x') \right).$$

8.5 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. [**?**] tested the NTK classifier on 90 small to medium scale datasets from UCI database. [3] 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.

[3] https://archive.ics.uci.edu/ml/datasets.php

For every neural network architecture, one can derive a corresponding kernel function. Du et al. [**?**] 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 [**?**]. However, for large scale data, Arora 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.

8.6 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 (8.10)

9
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 [**?**].

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 [**?**], batch normalization [**?**], dropout [**?**], 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.

[1] We refer the reader to [**?**] for an excellent exposition of over 50 of such proposals.

In this chapter, we investigate regularization due to Dropout, an algorithmic heurisitic recently proposed by [**?**]. 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].

[2] The parameter p is treated as a hyper-parameter which we typically tune for based on a validation set.

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 in-

sights from the success of the sexual reproduction model in the evolution of advanced organisms. Another motivation that was cited by [**?**] 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 9.4), matrix sensing (Section 9.1.1), matrix completion (Section 9.1.2), and deep learning (Section 9.2).

[3]

9.1 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 $\mathrm{U} \in \mathbb{R}^{d_2 \times d_1}, \mathrm{V} \in \mathbb{R}^{d_0 \times d_1}$:

$$\min_{\mathrm{U} \in \mathbb{R}^{d_2 \times d_1}, \mathrm{V} \in \mathbb{R}^{d_0 \times d_1}} \widehat{L}(\mathrm{U}, \mathrm{V}) := \frac{1}{n} \sum_{i=1}^{n} (y_i - \langle \mathrm{U}\mathrm{V}^\top, \mathrm{A}^{(i)} \rangle)^2. \tag{9.1}$$

When $d_1 \gg r_*$, there exist many "bad" empirical minimizers, i.e., those with a large true risk. However, recently, [**?**] 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 [**?**].

We propose solving the ERM problem (9.1) 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:

$$\widehat{L}_{\text{drop}}(\mathrm{U},\mathrm{V}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\mathrm{B}}(y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)} \rangle)^2, \tag{9.2}$$

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 $p \in [0,1)$:

$$\widehat{L}_{\text{drop}}(\mathrm{U},\mathrm{V}) = \widehat{L}(\mathrm{U},\mathrm{V}) + \frac{p}{1-p}\widehat{R}(\mathrm{U},\mathrm{V}), \tag{9.3}$$

where

$$\widehat{R}(\mathrm{U},\mathrm{V}) = \sum_{j=1}^{d_1} \frac{1}{n}\sum_{i=1}^{n} (\mathrm{u}_j^\top \mathrm{A}^{(i)} \mathrm{v}_j)^2 \tag{9.4}$$

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 9.2. Then, we can write

$$\begin{aligned} \mathbb{E}_{\mathrm{B}}[(y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)} \rangle)^2] \;\; &= \;\; \Big(\mathbb{E}_{\mathrm{B}}[y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)} \rangle]\Big)^2 \\ &\quad + \mathrm{Var}(y_i - \langle \mathrm{UBV}^\top, \mathrm{A}^{(i)} \rangle) \end{aligned} \tag{9.5}$$

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 (9.2), 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:

[4] Under mild assumptions, we can formally show that the dropout regularizer is well concentrated around its mean

$$\Theta(\mathrm{M}) := \min_{\mathrm{UV}^\top = \mathrm{M}} R(\mathrm{U}, \mathrm{V}), \quad \text{where} \quad R(\mathrm{U}, \mathrm{V}) := \mathbb{E}_{\mathrm{A}}[\widehat{R}(\mathrm{U}, \mathrm{V})].$$

Next, we consider two two important examples of random sensing matrices.

9.1.1 Gaussian Sensing Matrices

We assume that the entries of the sensing matrices are independently and identically distributed as standard Gaussian, i.e., $\mathrm{A}_{k\ell}^{(i)} \sim \mathcal{N}(0, 1)$. For Gaussian sensing matrices, we show that the induced regularizer due to Dropout provides nuclear-norm regularization. Formally, we show that

$$\Theta(\mathrm{M}) = \frac{1}{d_1} \|\mathrm{M}\|_*^2. \tag{9.6}$$

Proof. We recall the general form for the dropout regularizer for the matrix sensing problem in Equation 9.4, 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.

$$\begin{aligned}
R(\mathrm{U}, \mathrm{V}) &= \sum_{j=1}^{d_1} \mathbb{E}(\mathrm{u}_j^\top \mathrm{A} \mathrm{v}_j)^2 \\
&= \sum_{j=1}^{d_1} \mathbb{E}\Big(\sum_{k=1}^{d_2} \sum_{\ell=1}^{d_0} \mathrm{U}_{kj} \mathrm{A}_{k\ell} \mathrm{V}_{\ell j}\Big)^2 \\
&= \sum_{j=1}^{d_1} \sum_{k,k'=1}^{d_2} \sum_{\ell,\ell'=1}^{d_0} \mathrm{U}_{kj} \mathrm{U}_{k'j} \mathrm{V}_{\ell j} \mathrm{V}_{\ell' j} \, \mathbb{E}[\mathrm{A}_{k\ell} \mathrm{A}_{k'\ell'}] \\
&= \sum_{j=1}^{d_1} \sum_{k=1}^{d_2} \sum_{\ell=1}^{d_0} \mathrm{U}_{kj}^2 \mathrm{V}_{\ell j}^2 \, \mathbb{E}[\mathrm{A}_{k\ell}^2] \\
&= \sum_{j=1}^{d_1} \sum_{k=1}^{d_2} \sum_{\ell=1}^{d_0} \mathrm{U}_{kj}^2 \mathrm{V}_{\ell j}^2 \\
&= \sum_{j=1}^{d_1} \|\mathrm{u}_j\|^2 \|\mathrm{v}_j\|^2
\end{aligned}$$

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 [**?**].

Theorem 9.1.1. 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 9.1.1 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 9.1.1.

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 $\|\widetilde{u}_i\|\|\widetilde{v}_i\| = \frac{1}{d_1}\|M\|_*$.

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.

[5] since $(Q^\top G_U Q)_{jj} = \|\widetilde{u}_j\|^2$

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 $W_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & w_{22} & w_{23} & \cdots & w_{2d} \end{bmatrix} \in \mathbb{R}^{d\times d}$ 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 $\operatorname{Tr}(\widetilde{U}\widetilde{U}^\top) = \operatorname{Tr}(Q^\top G_U Q) = \operatorname{Tr}(\Sigma) = \operatorname{Tr}(Q^\top G_V Q) = \operatorname{Tr}(\widetilde{V}^\top \widetilde{V})$.

□

9.1.2 *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 [**?**] and [**?**]. Formally, we show that

$$\Theta(\mathrm{M}) = \frac{1}{d_1}\|\mathrm{diag}(\sqrt{p})\mathrm{U}\mathrm{V}^\top \mathrm{diag}(\sqrt{q})\|_*^2. \tag{9.7}$$

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 9.1.1 on $(\sqrt{\mathrm{diag}(\mathrm{p})}\mathrm{U}, \sqrt{\mathrm{diag}(\mathrm{q})}\mathrm{V})$, there exists

a rotation matrix Q such that

$$\|\sqrt{\mathrm{diag}(p)}Uq_j\|\|\sqrt{\mathrm{diag}(q)}Vq_j\| = \frac{1}{d_1}\|\sqrt{\mathrm{diag}(p)}UV^\top\sqrt{\mathrm{diag}(q)}\|_*.$$

We evaluate the expected dropout regularizer at UQ, VQ:

$$\begin{aligned}
R(UQ, VQ) &= \sum_{j=1}^{d_1} \|\sqrt{\mathrm{diag}(p)}Uq_j\|^2\|\sqrt{\mathrm{diag}(q)}Vq_j\|^2 \\
&= \sum_{j=1}^{d_1} \frac{1}{d_1^2}\|\sqrt{\mathrm{diag}(p)}UV^\top\sqrt{\mathrm{diag}(q)}\|_*^2 \\
&= \frac{1}{d_1}\|\sqrt{\mathrm{diag}(p)}UV^\top\sqrt{\mathrm{diag}(q)}\|_*^2 \\
&\leq \Theta(UV^\top)
\end{aligned}$$

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 9.1.2. Without loss of generality, assume that $d_2 \geq d_0$ and $\|M_*\| \leq 1$. 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 (9.2), and assume that $\max_i \|U(i,:)\|^2 \leq \gamma$, $\max_i \|V(i,:)\|^2 \leq \gamma$. Let α be such that $\widehat{R}(U, V) \leq \alpha/d_1$. 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:

$$L(U, V) \leq \widehat{L}(U, V) + C(1+\gamma)\sqrt{\frac{\alpha d_2 \log(d_2)}{n}} + C'(1+\gamma^2)\sqrt{\frac{\log(2/\delta)}{2n}}$$

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 9.1.2 follows from standard generalization bounds for ℓ_2 loss [**?**] based on the Rademacher complexity [**?**] of the class of functions with weighted trace-norm bounded by $\sqrt{\alpha}$, i.e. $\mathcal{M}_\alpha := \{M : \|\mathrm{diag}(\sqrt{p})M\mathrm{diag}(\sqrt{q})\|_*^2 \leq \alpha\}$. A bound on the Rademacher complexity of this class was established by [**?**]. 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 [**?**].

We note that for large enough sample size, $\widehat{R}(U,V) \approx R(U,V) \approx \Theta(UV^\top) = \frac{1}{d_1}\|\mathrm{diag}(\sqrt{p})UV^\top \mathrm{diag}(\sqrt{q})\|_*^2$, 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 $\widehat{R}(U,V) \le \alpha/d_1$ in the statement of the corollary.

In practice, for models that are trained with dropout, the training error $\widehat{L}(U,V)$ 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 9.1.2.

The assumption $\max_i \|U(i,:)\|^2 \le \gamma$, $\max_i \|V(i,:)\|^2 \le \gamma$ 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 [**?**]. 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 $\widehat{L}(U,V)$ 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.

9.2 *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 $\{(x_i, y_i)\}_{i=1}^n \sim \mathcal{D}^n$ 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 $f_w : \mathcal{X} \to \mathcal{Y}$, parameterized by w, that has a small *population risk* $L(w) := \mathbb{E}_{\mathcal{D}}[\ell(f_w(x), y)]$.

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 [**?**]. 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*:

$$\widehat{L}_{\mathrm{drop}}(\mathrm{w}) := \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\mathrm{B}}\,\|\mathrm{y}_i - \mathrm{W}_k\mathrm{B}\sigma(\mathrm{W}_{k-1}\sigma(\cdots \mathrm{W}_2\sigma(\mathrm{W}_1\mathrm{x}_i)\cdots))\|^2 \tag{9.8}$$

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 9.8) 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 9.2.1 (Dropout regularizer in deep regression)**.**

$$\widehat{L}_{\mathrm{drop}}(\mathrm{w}) = \widehat{L}(\mathrm{w}) + \widehat{R}(\mathrm{w}), \quad \text{where} \quad \widehat{R}(\mathrm{w}) = \lambda\sum_{j=1}^{d_{k-1}}\|\mathrm{W}_k(:,j)\|^2\widehat{a}_j^2.$$

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

$$\mathbb{E}_{\mathrm{B}}[\|\mathrm{y} - \mathrm{W}_k \mathrm{B} \mathrm{a}_{k-1}(\mathrm{x})\|^2] = \sum_{i=1}^{d_k} \mathbb{E}_{\mathrm{B}}(y_i - \mathrm{W}_k(i,:)^\top \mathrm{B} \mathrm{a}_{k-1}(\mathrm{x}))^2. \tag{9.9}$$

The following holds for each of the summands above:

$$\begin{aligned}\mathbb{E}_{\mathrm{B}}(y_i - \mathrm{W}_k(i,:)^\top \mathrm{B} \mathrm{a}_{k-1}(\mathrm{x}))^2 &= \Big(\mathbb{E}_{\mathrm{B}}[y_i - \mathrm{W}_k(i,:)^\top \mathrm{B} \mathrm{a}_{k-1}(\mathrm{x})]\Big)^2 \\ &\quad + \mathrm{Var}(y_i - \mathrm{W}_k(i,:)^\top \mathrm{B} \mathrm{a}_{k-1}(\mathrm{x})).\end{aligned}$$

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 (9.9)

$$\begin{aligned}\mathbb{E}_{\mathrm{B}}[\|\mathrm{y} - \mathrm{W}_k \mathrm{B} \mathrm{a}_{k-1}(\mathrm{x})\|^2] &= \|\mathrm{y} - \mathrm{W}_k \mathrm{a}_{k-1}(\mathrm{x})\|^2 \\ &\quad + \frac{p}{1-p} \sum_{j=1}^{d_{k-1}} \|\mathrm{W}_k(:,j)\|^2 a_{j,k-1}(\mathrm{x})^2\end{aligned}$$

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 [**?**]. Such a connection has been previously established for deep linear networks [**? ?**]; here we extend that result to single hidden layer ReLU networks.

Proposition 9.2.2. 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

$$R(\mathrm{w}) := \mathbb{E}[\widehat{R}(\mathrm{w})] = \frac{\lambda}{2} \sum_{i_0, i_1, i_2 = 1}^{d_0, d_1, d_2} \mathrm{W}_2(i_2, i_1)^2 \mathrm{W}_1(i_1, i_0)^2, \tag{9.10}$$

Proof of Proposition 9.2.2. Using Proposition 9.2.1, 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 $\mathbb{E}_z[\sigma(z)^2] = \frac{1}{2}\mathbb{E}_z[z^2]$.

$$\begin{aligned}
\mathop{\mathbb{E}}_{z}[\sigma(z)^2] &= \int_{-\infty}^{\infty} \sigma(z)^2 d\mu(z) \\
&= \int_{0}^{\infty} \sigma(z)^2 d\mu(z) = \int_{0}^{\infty} z^2 d\mu(z) \\
&= \frac{1}{2} \int_{-\infty}^{\infty} z^2 d\mu(z) = \frac{1}{2} \mathop{\mathbb{E}}_{z}[z^2].
\end{aligned}$$

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. □

9.3 *Landscape of the Optimization Problem*

While the focus in Section 9.2 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 9.11) 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 [**? ?**]. 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. $U = V$. We assume that the input distribution is isotropic, i.e. $C_x = I$. In this case, the population risk reduces to

$$\begin{aligned}\mathbb{E}[\|y - UU^\top x\|^2] &= \mathrm{Tr}\,(C_y) - 2\langle C_{yx}, UU^\top\rangle + \|UU^\top\|_F^2 \\ &= \|M - UU^\top\|_F^2 + \mathrm{Tr}\,(C_y) - \|C_{yx}\|_F^2\end{aligned}$$

where $M = \frac{C_{yx}+C_{xy}}{2}$. Ignoring the terms that are independent of the weights matrix U, the goal is to minimize $L(U) = \|M - UU^\top\|_F^2$. Using Dropout amounts to solving the following problem:

$$\min_{U\in\mathbb{R}^{d_0\times d_1}} L_\theta(U) := \|M - UU^\top\|_F^2 + \lambda \underbrace{\sum_{i=1}^{d_1} \|u_i\|^4}_{R(U)} \tag{9.11}$$

We can characterize the global optima of the problem above as follows.

Theorem 9.3.1. For any $j \in [r]$, let $\kappa_j := \frac{1}{j}\sum_{i=1}^{j}\lambda_i(C_{yx})$. Furthermore, define $\rho := \max\{j \in [r] : \lambda_j(C_{yx}) > \frac{\lambda j \kappa_j}{r+\lambda j}\}$. Then, if U_* is a global optimum of Problem 9.11, it satisfies that $U_*U_*^\top = \mathcal{S}_{\frac{\lambda\rho\kappa_\rho}{r+\lambda\rho}}(C_{yx})$.

Next, it is easy to see that the gradient of the objective of Problem 9.11 is given by

$$\nabla L_\theta(U) = 4(UU^\top - M)U + 4\lambda U\mathrm{diag}(U^\top U).$$

We also make the following important observation about the critical points of Problem 9.11. Lemma 9.3.2 allows us to bound different norms of the critical points, as will be seen later in the proofs.

Lemma 9.3.2. If U is a critical point of Problem 9.11, then it holds that $UU^\top \preceq M$.

Proof of Lemma 9.3.2. Since $\nabla L_\theta(U) = 0$, we have that

$$(M - UU^\top)U = \lambda U\mathrm{diag}(U^\top U)$$

multiply both sides from right by $U^\top$ and rearrange to get

$$MUU^\top = UU^\top UU^\top + \lambda U\mathrm{diag}(U^\top U)U^\top \tag{9.12}$$

Note that the right hand side is symmetric, which implies that the left hand side must be symmetric as well, i.e.

$$MUU^\top = (MUU^\top)^\top = UU^\top M,$$

so that M and $UU^\top$ commute. Note that in Equation (9.12), $U\mathrm{diag}(U^\top U)U^\top \succeq 0$. Thus, $MUU^\top \succeq UU^\top UU^\top$. Let $UU^\top = W\Gamma W^\top$ be a compact eigen-decomposition of $UU^\top$. We get

$$MUU^\top = MW\Gamma W^\top \succeq UU^\top UU^\top = W\Gamma^2 W^\top.$$

Multiplying from right and left by $W\Gamma^{-1}$ and $W^\top$ respectively, we get that $W^\top MW \succeq \Gamma$ which completes the proof. □

We show in Section 9.3.1 that (a) local minima of Problem 9.11 inherit the same implicit bias as the global optima, i.e. all local minima are equalized. Then, in Section 9.3.2, 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 9.3.4).

9.3.1 *Implicit bias in local optima*

Recall that the population risk $L(U)$ is rotation invariant, i.e. $L(UQ) = L(U)$ for any rotation matrix Q. Now, if the weight matrix U were not equalized, then there exist indices $i, j \in [r]$ such that $\|u_i\| > \|u_j\|$. 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 9.3.3. All local minima of Problem 9.11 are equalized, i.e. if U is a local optimum, then $\|u_i\| = \|u_j\|$ $\forall i, j \in [r]$.

Lemma 9.3.3 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 9.1.

Proof of Lemma 9.3.3. We show that if U is not equalized, then any ϵ-neighborhood of U contains a point with dropout objective strictly smaller than $L_\theta(U)$. More formally, for any $\epsilon > 0$, we exhibit a rotation Q_ϵ such that $\|U - UQ_\epsilon\|_F \leq \epsilon$ and $L_\theta(UQ_\epsilon) < L_\theta(U)$. Let U be a critical point of Problem 9.11 that is not equalized, i.e. there exists two columns of U with different norms. Without loss of generality, let $\|u_1\| > \|u_2\|$. 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 regularizer,

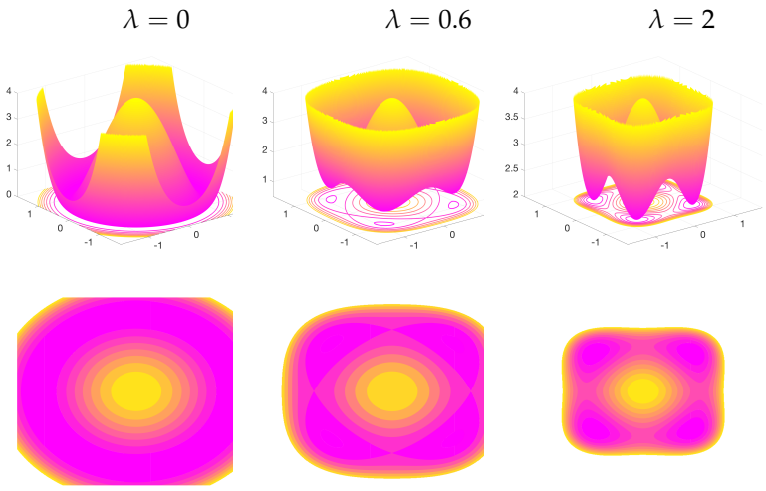


Figure 9.1: Optimization landscape (top) and contour plot (bottom) for a single hidden-layer linear autoencoder network with one dimensional input and output and a hidden layer of width $r = 2$ with dropout, for different values of the regularization parameter λ. Left: for $\lambda = 0$ the problem reduces 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.

while preserving the value of the loss. To that end, define

$$Q_\delta := \begin{bmatrix} \sqrt{1-\delta^2} & -\delta & 0 \\ \delta & \sqrt{1-\delta^2} & 0 \\ 0 & 0 & I_{r-2} \end{bmatrix}$$

and let $\widehat{U} := UQ_\delta$. It is easy to verify that Q_ϵ is indeed a rotation. First, we show that for any ϵ, as long as $\delta^2 \leq \frac{\epsilon^2}{2\,\mathrm{Tr}(M)}$, we have $\widehat{U} \in \mathcal{B}_\epsilon(U)$:

$$\begin{aligned} \|U - \widehat{U}\|_F^2 &= \sum_{i=1}^{r} \|u_i - \widehat{u}_i\|^2 \\ &= \|u_1 - \sqrt{1-\delta^2}u_1 - \delta u_2\|^2 + \|u_2 - \sqrt{1-\delta^2}u_2 + \delta u_1\|^2 \\ &= 2(1-\sqrt{1-\delta^2})(\|u_1\|^2 + \|u_2\|^2) \\ &\leq 2\delta^2\,\mathrm{Tr}(M) \leq \epsilon^2 \end{aligned}$$

where the second to last inequality follows from Lemma 9.3.2, because $\|u_1\|^2 + \|u_2\|^2 \leq \|U\|_F^2 = \mathrm{Tr}(UU^\top) \leq \mathrm{Tr}(M)$, 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 $\widehat{U}$ is strictly smaller than that of U. Observe that

$$\begin{aligned} \|\widehat{u}_1\|^2 &= (1-\delta^2)\|u_1\|^2 + \delta^2\|u_2\|^2 + 2\delta\sqrt{1-\delta^2}u_1^\top u_2 \\ \|\widehat{u}_2\|^2 &= (1-\delta^2)\|u_2\|^2 + \delta^2\|u_1\|^2 - 2\delta\sqrt{1-\delta^2}u_1^\top u_2 \end{aligned}$$

and the remaining columns will not change, i.e. for $i = 3, \cdots, r$, $\widehat{u}_i = u_i$. Together with the fact that Q_δ preserves the norms, i.e. $\|U\|_F = \|UQ_\delta\|_F$, we get

$$\|\widehat{u}_1\|^2 + \|\widehat{u}_2\|^2 = \|u_1\|^2 + \|u_2\|^2. \tag{9.13}$$

Let $\delta = -c \cdot \mathrm{sgn}(u_1^\top u_2)$ for a small enough $c > 0$ such that $\|u_2\| < \|\widehat{u}_2\| \leq \|\widehat{u}_1\| < \|u_1\|$. Using Equation (9.13), This implies that $\|\widehat{u}_1\|^4 + \|\widehat{u}_2\|^4 < \|u_1\|^4 + \|u_2\|^4$, which in turn gives us $R(\widehat{U}) < R(U)$ and hence $L_\theta(\widehat{U}) < L_\theta(U)$. Therefore, a non-equalized critical point cannot be local minimum, hence the first claim of the lemma. □

9.3.2 *Landscape properties*

Next, we characterize the solutions to which dropout converges. We do so by understanding the optimization landscape of Problem 9.11. Central to our analysis, is the following notion of *strict saddle property*.

Definition 9.3.4 (Strict saddle point/property)**.** Let $f : \mathcal{U} \to \mathbb{R}$ be a twice differentiable function and let $U \in \mathcal{U}$ 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. $\lambda_{\min}(\nabla^2 f(U)) < 0$. 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 [**?** **?**]. Following this idea, there has been a flurry of works on studying the landscape of different machine learning problems, including low rank matrix recovery [**?**], generalized phase retrieval problem [**?**], matrix completion [**?**], deep linear networks [**?**], matrix sensing and robust PCA [**?**] and tensor decomposition [**?**], 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 9.11 reduces to standard squared loss minimization which has been shown to have no spurious local minima and satisfy strict saddle property (see, e.g. [**?** **?**]). 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 9.3.5. Let $r := \mathrm{Rank}(M)$. Assume that $d_1 \leq d_0$ and that the regularization parameter satisfies $\lambda < \frac{r\lambda_r(M)}{(\sum_{i=1}^r \lambda_i(M)) - r\lambda_r(M)}$. Then it holds for Problem 9.11 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 $UU^\top \in \mathbb{R}^{d_0 \times d_0}$ 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 9.3.5 guarantees that any critical point U that is not a global optimum is a strict saddle point, i.e. $\nabla^2 L(U, U)$ 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 9.3.5 hold when the regularization parameter λ is sufficiently small. Assumptions of this kind are common in the literature (see, for example [**?**]). While this is a *sufficient* condition for the result in Theorem 9.3.5, it is not clear if it is *necessary*.

Proof of Theorem 9.3.5. Here we outline the main steps in the proof of Theorem 9.3.5.

1. In Lemma 9.3.3, we show that the set of non-equalized critical points does not include any local optima. Furthermore, Lemma 9.3.6 shows that all such points are strict saddles.

2. In Lemma 9.3.7, 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 9.3.6. *All critical points of Problem 9.11 that are not equalized, are strict saddle points.*

Proof of Lemma 9.3.6. By Lemma 9.3.3, 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 $\|u_1\| > \|u_2\|$ and consider the curve

$$\Delta(t) := [(\sqrt{1-t^2}-1)u_1 + tu_2, (\sqrt{1-t^2}-1)u_2 - tu_1, o_{d,r-2}]$$

It is easy to check that for any $t \in \mathbb{R}$, $L(U + \Delta(t)) = L(U)$ since $U + \Delta(t)$ 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

$$\begin{aligned}
g'(t) = &-4t(\|\mathrm{u}_1\|^4 + \|\mathrm{u}_2\|^4) + 16t(\mathrm{u}_1\mathrm{u}_2)^2 + 8t\|\mathrm{u}_1\|^2\|\mathrm{u}_2\|^2 \\
&+ 4(\sqrt{1-t^2} - \frac{t^2}{\sqrt{1-t^2}})(\mathrm{u}_1^\top\mathrm{u}_2)(\|\mathrm{u}_1\|^2 - \|\mathrm{u}_2\|^2) + O(t^2).
\end{aligned}$$

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 $\nabla L_\theta(\mathrm{U}) = \mathrm{o}$ and $\mathrm{diag}(\mathrm{U}^\top\mathrm{U}) = \frac{\|\mathrm{U}\|_F^2}{d_1}\mathrm{I}$.

Lemma 9.3.7. *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 9.3.7. 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:

$$\begin{aligned}
&\nabla L_\theta(\mathrm{U}) = 4(\mathrm{U}\mathrm{U}^\top - \mathrm{M})\mathrm{U} + 4\lambda\mathrm{U}\mathrm{diag}(\mathrm{U}^\top\mathrm{U}) = \mathrm{o} \\
\implies &\mathrm{U}\mathrm{U}^\top\mathrm{U} + \frac{\lambda\|\mathrm{U}\|_F^2}{d_1}\mathrm{U} = \mathrm{M}\mathrm{U} \\
\implies &\mathrm{W}\Sigma^3\mathrm{V}^\top + \frac{\lambda\|\Sigma\|_F^2}{d_1}\mathrm{W}\Sigma\mathrm{V}^\top = \mathrm{M}\mathrm{W}\Sigma\mathrm{V}^\top \\
\implies &\Sigma^2 + \frac{\lambda\|\Sigma\|_F^2}{d_1}\mathrm{I} = \mathrm{W}^\top\mathrm{M}\mathrm{W}
\end{aligned}$$

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

$$\mathrm{diag}(\Sigma^2) = (I - \frac{\lambda}{d_1 + \lambda r'})\vec{\lambda} = \vec{\lambda} - \frac{\lambda \sum_{i=1}^{r'} \lambda_i}{d_1 + \lambda r'} 1_{r'}. \tag{9.14}$$

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 (9.14):

$$\sigma_{r'}^2 = \lambda_{r'} - \frac{\lambda \sum_{i=1}^{r'} \lambda_i}{d_1 + \lambda r'}.$$

Since $\mathrm{Rank}\, U = r'$, 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 9.3.1 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 9.3.1. 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

$\mathrm{U}(t) = \mathrm{W}_{d_0}\Sigma'\mathrm{V}_{d_1}^\top$ 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 $i \leq d_1$. Observe that

$$\mathrm{U}(t)^\top \mathrm{U}(t) = \mathrm{V}\Sigma^2\mathrm{V}^\top + t^2\mathrm{V}_{d_1}^\top\mathrm{V}_{d_1} = \mathrm{U}^\top\mathrm{U} + t^2\mathrm{I}_{d_1}.$$

Thus, the parametric curve $\mathrm{U}(t)$ is equalized for all t. The population risk at $\mathrm{U}(t)$ equals:

$$\begin{aligned} L(\mathrm{U}(t)) &= \sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2 - t^2)^2 + \sum_{i=d_1+1}^{d_0}\lambda_i^2 \\ &= L(\mathrm{U}) + d_1 t^4 - 2t^2\sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2). \end{aligned}$$

Furthermore, since $\mathrm{U}(t)$ is equalized, we obtain the following form for the regularizer:

$$\begin{aligned} R(\mathrm{U}(t)) &= \frac{\lambda}{d_1}\|\mathrm{U}(t)\|_F^4 = \frac{\lambda}{d_1}\left(\|\mathrm{U}\|_F^2 + d_1 t^2\right)^2 \\ &= R(\mathrm{U}) + \lambda d_1 t^4 + 2\lambda t^2\|\mathrm{U}\|_F^2. \end{aligned}$$

Define $g(t) := L(\mathrm{U}(t)) + R(\mathrm{U}(t))$. We have that

$$g(t) = L(\mathrm{U}) + R(\mathrm{U}) + d_1 t^4 - 2t^2\sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2) + \lambda d_1 t^4 + 2\lambda t^2\|\mathrm{U}\|_F^2.$$

It is easy to verify that $g'(0) = 0$. Moreover, the second derivative of g at $t = 0$ is given as:

$$g''(0) = -4\sum_{i=1}^{d_1}(\lambda_i - \sigma_i^2) + 4\lambda\|\mathrm{U}\|_F^2 = -4\sum_{i=1}^{d_1}\lambda_i + 4(1+\lambda)\|\mathrm{U}\|_F^2 \quad (9.15)$$

We use $\|\mathrm{U}\|_F^2 = \sum_{i=1}^{r'}\sigma_i^2$ and Equation (9.14) to arrive at

$$\|\mathrm{U}\|_F^2 = \mathrm{tr}\Sigma^2 = \sum_{i=1}^{r'}(\lambda_i - \frac{\lambda\sum_{j=1}^{r'}\lambda_j}{d_1 + \lambda r'}) = (\sum_{i=1}^{r'}\lambda_i)(1 - \frac{\lambda r'}{d_1 + \lambda r'}) = \frac{d_1\sum_{i=1}^{r'}\lambda_i}{d_1 + \lambda r'}$$

Plugging back the above equality in Equation (9.15), 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 U to be a strict saddle point, it suf-

fices 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 $\lambda_{r'+1}, \ldots, \lambda_{d_1}$. 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. □

9.4 *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 9.8), 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 [**?**] 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:

$$\langle \mathrm{U}\mathrm{V}^\top, \mathrm{A}\rangle = \langle \mathrm{U}^\top, \mathrm{V}^\top\mathrm{A}^\top\rangle = \langle \mathrm{U}^\top, (\mathrm{I}\otimes\mathrm{V}^\top)\mathrm{A}^\top\rangle.$$

10
Unsupervised learning: Overview

Much of the book so far concerned supervised learning —i.e., learning to classify inputs into classes, where the training data consists of sampled inputs together with their correct labels. This chapter is an introduction to unsupervised learning, where one has randomly sampled datapoints but no labels or classes.

10.0.1 **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 actualyy *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 distribution of data is $p_\theta(h, x)$ where θ is some unknown vector of parameters, x is the observable (i.e., data) and h are some 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 $1 - \rho_2$ fraction. Then θ vector consists of parameters of the two Gaussians

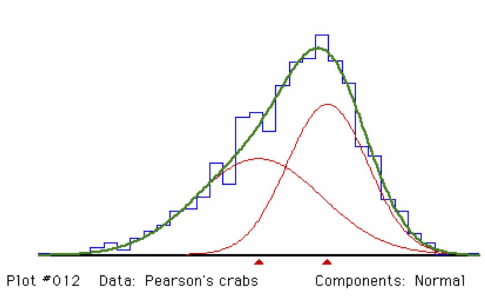


Figure 10.1: Visualization of Pearson's Crab Data as mixture of two Gaussians. (Credit: MIX homepage at McMaster University.)

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.

The goal is to learn this description of the distribution given i.i.d. samples of 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 done via distribution learning: 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 sometimes called an *autoencoder* in deep learning.

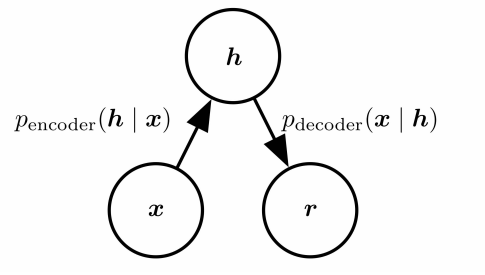


Figure 10.2: Autoencoder defined using a density distribution $p(h, x)$, where h is the latent feature vector corresponding to visible vector x. The process of computing h given x is called "encoding" and the reverse is called "decoding." In general applying the encoder on x followed by the decoder would 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 laterr —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 is perhaps why people sometimes conflate representation learning with density estimation.

10.1 Training Objective for Density estimation: Log Likelihood

From the above description it is clear that one way to formalize "structure "of data is to think of it as being i.i.d. samples $S = \{x_1, x_2, ...x_m\}$ samples from an underlying distribution, and to learn this underlying distribution. We assume the unknown distribution has a known parametric form $p_\theta(x)$ which is the probability of observing x given the parameters of the model, θ. But θ is unknown. For example, θ could be parameters of an unknown deep net g_θ with a certain prescribed architecture, and x is generated by using an $h \sim \mathcal{N}(0, I)$ and outputting $g_\theta(h)$.

We wish to infer the best θ given the i.i.d. samples ("evidence") from the distribution. One standard way to quantify "best" is pick θ is according to the *maximum likelihood principle.*

$$\max_\theta \prod_{x^{(i)} \in S} p_\theta(x^{(i)}) \tag{10.1}$$

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:

$$\max_\theta \sum_{x^{(i)} \in S} \log p_\theta(x^{(i)}) \quad \textit{(log likelihood)} \tag{10.2}$$

Thus it is always possible to fit such a parametric form. The question is how well this learnt distribution fits the data distribution. We need a notion of "goodness" for unsupervised learning that is analogous to *generalization* in supervised learning. The obvious one is log likelihood of *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 10.1.1. *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 loglikelihood for the average training and held-out sample is insufficient for good distribution learning. The gaussian model only has $d + 1$ parameters and simple concentration bounds show under fairly general conditions (such as coordinates of x_i's being bounded) that one the number of training samples is moderately high then the log likelihood of the average test sample is similar to that of 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 — one that outputs random labels—has excellent generalization as well.

As in supervised learning, one has to keep track of training log-likelihood in addition to generalization, and choose among models that maximise it. In general this is computationally intractable for even simple settings.

Theorem 10.1.2. *The θ maximizing (10.2) minimizes the KL divergence $KL(Q||P)$ where P is the true distribution and Q is the learnt distribution.*

Proof. TBD □

10.2 Variational methods

The variational method leverages *duality*, a widespread principle in math. You may have seen LP duality in an algorithms class. The name "variational"in the title refers to calculus of variations, the part of math where such principles are studied.

This method maintains some estimate $q(h|x)$ of $p(h|x)$ and improves it. One useful fact is that:

$$\log p(x) \geq \mathbb{E}_{q(h|x)}[\log(p(x,h))] + H[q(h|x)], \quad \forall q(h|x) \tag{10.3}$$

where H is the Shannon Entropy.

We would like to prove this bound on $\log p(x)$ and resort to maximizing the lower bound given in (10.3), referred to as the evidence lower bound (ELBO). Towards this end we will introduce the Kullback Leibler divergence (KL) between two distributions given by

$$KL[q(h|x) \,||\, p(h|x)] = \mathbb{E}_{q(h|x)}\left[\log \frac{q(h|x)}{p(h|x)}\right] \tag{10.4}$$

Moreover, $p(x)p(h|x) = p(x,h)$ is true by Bayes Rule. Then we can see that

$$KL[q(h|x)|p(h|x)] = \mathbb{E}_{q(h|x)}[\log \frac{q(h|x)}{p(x,h)} \cdot p(x)] \tag{10.5}$$

$$= \underbrace{\mathbb{E}_{q(h|x)}[\log(q(h|x))]}_{-H(q(h|x))} - \mathbb{E}_{q(h|x)}[\log(p(x,h))] + \mathbb{E}_{q(h|x)}[\log p(x)] \tag{10.6}$$

But we know that the KL divergence is always nonnegative, so we get:

$$\mathbb{E}_{q(h|x)}[\log(p(x))] - \mathbb{E}_{q(h|x)}[\log(p(x,h))] - H(q(h|x)) \geq 0 \tag{10.7}$$

which is the same as ELBO (10.3) since $\log(p(x))$ is constant over $q(h|x)$, hence is equal to its expectation.

The variational methods use some form of gradient descent or local improvement to improve $q(h|x)$. For details see the blog post on offconvex.org by Arora and Risteski.

10.3 Autoencoders

Autoencoders are a subcase of density estimation popular in the neural nets world. It is assumed that the model first generates a latent representation from a simple distribution, and then uses some simple circuit to map it to the visible datapoint x. This is called *decoder*. There is a companion *encoder* circuit that maps x to a latent representation. The pair of circuits is called an autoencoder.

In other words, the autoencoder is trying to learn an approximation to the identity function, so as to output x' that is similar to x. In order to force the algorithm to learn a function different from the identity function, we place constraints on the model, such as by limiting the representation z to be of low dimensionality. In the simplest example, assume k vectors $u_1, ..., u_k \in \mathbb{R}^n$, where $k \ll n$, and $x = \sum_i \alpha_i u_i + \sigma$, where σ is Gaussian noise. By applying rank-k PCA, one could recover values in $span(u_1, ..., u_k)$. Thus PCA/SVD can be seen as a simple form of autoencoder learning.

An autoencoder can also be seen as an example of the so-called *manifold* view, whereby data is assumed to have a latent representation z which has a simpler distribution than x.

10.3.1 Sparse autoencoder

Our argument above relied on the size of the encoded representation of the data to be small. But even when this is not the case, (i.e., $k > n$), we can still learn meaningful structure, by imposing other

constraints on the network. In particular, we could impose sparsity constraints on the mapped value. Essentially, if $x = \sum_i \alpha_i u_i + \sigma$ as above, we could enforce $\boldsymbol{\alpha}$ to be r-sparse, i.e., allowing only r non-zero values. Examples of "sparse coding" include [**?** **?**].

10.3.2 Topic models

Topic models are used to learn the abstract "topics" that occur in a collection of documents, and uncover hidden semantic structures. They can also been fitted into the autoencoder view. Assume the documents are given in a bag-of-words representation. As defined above, we have $u_1, ..., u_k \in \mathbb{R}^n$ vectors, with $k < n$. We enforce that $\sum_i \alpha_i = 1$, such that α_i is the coefficient of the i-th topic. For example, a news article might be represented as a mixture of 0.7 of the topic *politics* and 0.3 of the topic *economy*. The goal in topic modeling is, when given a large enough collection of documents, to discover the underlying set of topics used to generate them, both efficiently and accurately. A practical algorithm for topic modeling with provable guarantees is given by [**?**].

10.4 Variational Autoencoder (VAE)

In the context of deep learning, Variational Autoencoders (VAEs) [1] are one of the earliest models that have demonstrated promising qualitative performance in learning complicated distributions. 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.

[1]

Recall that in density estimation we are given a data sample $x_1, \ldots, x_m$ 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 $KL(q(z|x) \,||\, p(z|x)) \geq 0$

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x, z)] - \mathbb{E}_{q(z|x)}[\log q(z|x)] = ELBO \qquad (10.8)$$

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,

$$q(z|x) = \mathcal{N}(z; \mu_x, \sigma_x^2 I_d), \quad \mu_x, \sigma_x = E_\phi(x) \tag{10.9}$$

$$p(x|z) = \mathcal{N}(x; \mu_z, \sigma_z^2 I_n), \quad \mu_z, \sigma_z = D_\theta(z), \quad p(z) = \mathcal{N}(z; 0, I_d) \tag{10.10}$$

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 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.

10.4.1 Training VAEs

As previously mentioned, the training of variational autoencoders involves maximizing the RHS of (10.8), the ELBO, over the parameters ϕ, θ under the model described by (10.9), (10.10). 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 [2] 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 ran-

[2]

domness 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

$$\sum_{j=1}^{s} \log p(x_i, z_{ij}) - \sum_{j=1}^{s} \log q(z_{ij}|x_i) = \sum_{j=1}^{s} [\log p(x_i|z_{ij}) + \log p(z_{ij}) - \log q(z_{ij}|x_i)] \tag{10.11}$$

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 (10.11) 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

$$-\frac{1}{2}\left[\frac{||x_i - \mu_\theta(z_{ij})||^2}{\sigma_\theta(z_{ij})^2} + n \log \sigma_\theta(z_{ij})^2 + ||z_{ij}||^2 - \frac{||z_{ij} - \mu_\phi(x_i)||^2}{\sigma_\phi(x_i)^2} - d \log \sigma_\phi(x_i)^2\right] \tag{10.12}$$

Notice that (10.12) 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 (10.11) 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.

10.5 Main open question

Main open question for this lecture is to design methods with provable guarantees for the learning problems discussed here. Can we show that VAEs correctly (and efficiently) learn simple families of probability distributions?

There were notable successes in analysis of method of moments for learning probability distributions, as mentioned above. Variational methods rely upon gradient descent, which seems harder to analyse as of now.

11
Generative Adversarial Nets

Chapter 10 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. For convenience in this chapter we assume the model is trying to generate images. The following would be one standard interpretation of what it means for the distribution produced by the model to be realistic.

Interpretation 1: *The distributions of real and synthetic images are, as distributions in $\Re^d$, are close in some statistical measure.*

The main novelty in GANs is a different interpretation that leverages the power of supervised deep learning.

Interpretation 2: *If we try to train a powerful deep net to distinguish between real and synthetic images, by training it to output "1" on a training set of real images and "0" on a training set of synthetic images from our model, then such a net fails to have significant success in distinguishing among held-out real vs synthetic images at test time.*

Is it possible that the two interpretations are interrelated? The simplistic answer is yes: a rich mathematical framework of transportation distances can give a relationship between the two intepretations. A more nuanced answer is "maybe", at least if one treats deep nets as black boxes with limited representation power. Then a simple but surprising result shows that the richness of the synthetic distribution can be quite limited —and one has to worry about *mode collapse.*

11.1 Basic definitions

A generative model G_θ (where θ is the vector of parameters) is a deep net that maps from $\Re^k$ to $\Re^d$. Given a random seed h —which is usually a sample from a multivariate Normal distribution—it produces a vector string $G_\theta(h)$ that is an image.

Model G_θ will be trained using a finite set of training examples from some real-life distribution of images $\mathcal{D}_{real}$ (e.g., pictures of celebrities from gossip magazines). Let D_{synth} be the distribution it generates at the end of training (meaning the distribution of $G_\theta(h)$ where h is drawn from the normal distribution).

12
Representation Learning

13
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.

13.1 Example of Theorems and Lemmas

We provide some examples

Theorem 13.1.1 (d-dimension sparse Fourier transform)**.** *There is an algorithm (procedure* FourierSparseRecovery *in Algorithm 2) 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 13.1.2 (Upper bound on the gradient)**.**

Theorem 13.1.3 (Main result)**.**

13.2 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 $\mathrm{OPT} := \|Ax^* - b\|_2$.

13.3 *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 2 Fourier Sparse Recovery Algorithm

procedure FOURIERSPARSERECOVERY(x, n, k, μ, R^*) ▷ Theorem 13.1.1
  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

13.4 *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.

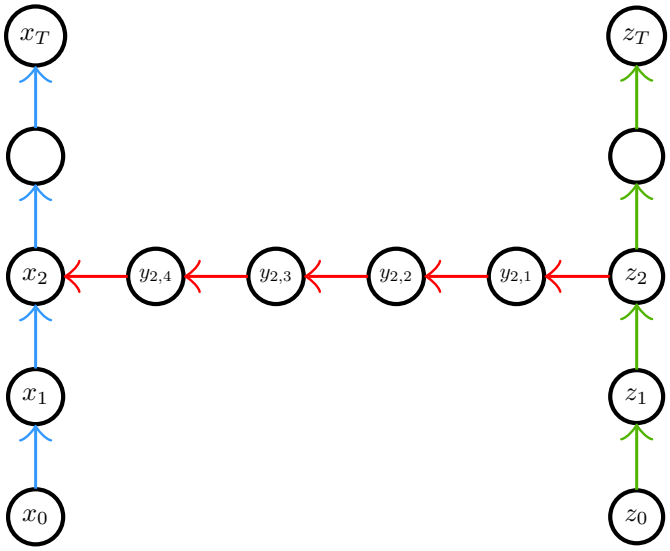


Figure 13.1: A chasing sequence

13.5 Example of Tables

Reference	Samples	Time	Filter	RIP
[**?**]	$k \log^{O(d)} n$	$k \log^{O(d)} n$	Yes	No
[**?**]	$k \log^6 n$	$\text{poly}(n)$	No	Yes
[**?**]	$k \log^2 k \log(k \log n) \log n$	$\widetilde{O}(n)$	No	Yes
[**?**]	$k \log^d n \log(n/k)$	$k \log^d n \log(n/k)$	Yes	No
[**?**]	$k \log^3 k \log n$	$\widetilde{O}(n)$	No	Yes
[**?**]	$2^{d \log d} k \log n$	$\widetilde{O}(n)$	Yes	No
[**?**]	$k \log k \log^2 n$	$\widetilde{O}(n)$	No	Yes
[**?**]	$k \log^2 k \log n$	$\widetilde{O}(n)$	No	Yes
[**?**]	$2^{d^2} k \log n$	$2^{d^2} k \log^{d+O(1)} n$	Yes	No
[**?**]	$k^3 \log^2 k \log^2 n$	$k^3 \log^2 k \log^2 n$	Yes	Yes
[**?**]	$k \log k \log n$	$\widetilde{O}(n)$	No	No

Table 13.1: 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 [**? ? ? ? ?**] obtain improved analyses of the Restricted Isometry property; the algorithm is suggested and analyzed (modulo the RIP property) in [**?**]. The work in [**?**] does not explicitly state the extension to the d-dimensional case, but can easily be inferred from the arguments. [**? ? ? ? ?**] work when the universe size in each dimension are powers of 2.

13.6 Exercise

This section provides several examples of exercises.

Exercises

Exercise 13.6-1: Solve the following equation for $x \in C$, with C the set of complex numbers:

$$5x^2 - 3x = 5 \tag{13.1}$$

Exercise 13.6-2: Solve the following equation for $x \in C$, with C the set of complex numbers:

$$7x^3 - 2x = 1 \tag{13.2}$$
