Applying statistical learning theory to deep learning

Cédric Gerbelot *
Courant Institute of Mathematical Sciences, New York, NY 10012, USA

Avetik Karagulyan
King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia

Stefani Karp
Carnegie Mellon University, Pittsburgh, PA, and Google Research, NY, USA

Kavya Ravichandran †
Toyota Technological Institute at Chicago, Chicago, Illinois 60637, USA

Nathan Srebro
Toyota Technological Institute at Chicago, Chicago, Illinois 60637, USA

Menachem Stern
Department of Physics & Astronomy, University of Pennsylvania, Philadelphia, PA 19104-6396, USA

November 28, 2023

Abstract

Although statistical learning theory provides a robust framework to understand supervised learning, many theoretical aspects of deep learning remain unclear, in particular how different architectures may lead to inductive bias when trained using gradient based methods. The goal of these lectures is to provide an overview of some of the main questions that arise when attempting to understand deep learning from a learning theory perspective. After a brief reminder on statistical learning theory and stochastic optimization, we discuss implicit bias in the context of benign overfitting. We then move to a general description of the mirror descent algorithm, showing how we may go back and forth between a parameter space and the corresponding function space for a given learning problem, as well as how the geometry of the learning problem may be represented by a metric tensor. Building on this framework, we provide a detailed study of the implicit bias of gradient descent on linear diagonal networks for various regression tasks, showing how the loss function, scale of parameters at initialization and depth of the network may lead to various forms of implicit bias, in particular transitioning between kernel or feature learning.

*cedric.gerbelot@cims.nyu.edu
†kavya@ttic.edu

Contents

1 Lecture 1: Applying statistical learning theory to deep learning

1.1 Preamble

We start by giving you Nati's view of supervised learning and statistical learning theory, and the main forces in learning. We'll mostly focus on the concept of inductive bias. We'll be starting by trying to explain what is inductive bias, or rather, what I mean by inductive bias, as this term is hard to define mathematically. We'll incorporate this idea of inductive bias in the introduction of statistical learning theory, and try to understand how it applies to deep learning. We'll explore how deep learning fits into the approach we developed to understand learning theory in the past five decades. The main goal for today is to mention what we need to understand (what questions we need to answer), and what notions we need to rethink.

1.2 Inductive bias in supervised learning

The goal of supervised learning is to find a predictor, a mapping h from inputs or instances $\mathcal{X}$ (e.g. images, sentences) to labels $\mathcal{Y}$ (e.g. classes). In the simplest setting we'll just think of $y \in \{\pm 1\}$. The goal is to find a predictor that has a small generalization loss $L(h)$. Crucially, we measure the loss with respect to a source distribution $\mathcal{D}$ and our goal is to find h which in expectation over this distribution has small loss, meaning its predictions tend to be correct.

$$\begin{aligned} \underline{\text{Supervised learning}}: \quad & \text{find } h : \mathcal{X} \to \mathcal{Y} \text{ with small } \textit{generalization error} \\ & L(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[loss(h(x); y)] \end{aligned}$$

Machine learning designs this predictor h, not based on knowledge of the population $\mathcal{D}$ but rather based on an IID sample from that population. We try to find good *learning rules* that produce a predictor with small error for any population.

$$\begin{aligned} \underline{\text{Learning rule}}: \quad & \text{(based on sample } S\text{)} \\ & A: \ S \to h \quad (\text{i.e. } A : (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}}) \end{aligned}$$

Unfortunately, this is impossible. The 'no free lunch' theorem tells us small generalization error requires knowledge about the population. For any learning rule, there exists some distribution $\mathcal{D}$ (that is, some reality) for which the learning rule yields an expected error that is tantamount to randomly guessing the answer (e.g. 1/2 for binary classes). More formally, for any A, m there exists $\mathcal{D}$ such that $\exists_{h^*} L(h^*) = 0$ but

$$\mathbb{E}_{S\sim\mathcal{D}^m}[L(A(S))] \geq \frac{1}{2} - \frac{m}{2|\mathcal{X}|}$$

This is true not only for independent x, y, but also if there exists a deterministic relation $y(x)$, so that a predictor does exist. The supposed improvement over 1/2 is proportional to the size of the dataset $m/|\mathcal{X}|$ is due to memorization of that dataset, and vanishes when the population size is large.

Thus, learning is impossible without assuming anything. This is where inductive bias becomes an essential part of learning. We assume that some realities (populations $\mathcal{D}$) are unlikely, and design the learning rule A to work for the more likely realities, e.g. by preferring certain models $h(x)$ over others. More practically, we assume reality $\mathcal{D}$ has a certain property which ensures the learning rule A has good generalization error. Typically, we assume that there exist models $h(x)$ in some class $\mathcal{H}$, or with low "complexity" $c(h)$ such that it has low generalization error $L(h)$. An example are models where the output y changes smoothly with the input x. Another example is ridge regression, that prefers linear models (with small norms of the weights).

A flat inductive bias embodies the assumption that some realities are possible and others are not, $\exists h^* \in \mathcal{H}$ with low $L(h^*)$. If we make this assumption, we know what is the best learning rule for supervised learning, which is *empirical risk minimization*:

$$ERM_{\mathcal{H}}(S) = \hat{h} = \arg\min_{h\in\mathcal{H}} L_S(h)$$

with L_S the empirical, or *training* loss over our sample S of size m:

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} loss(h(x_i); y_i).$$

For this learning rule, we can guarantee an upper bound on the generalization error. If the best model in our class is h^*, the error of the predictor achieved by the ERM learning rule is

$$L(ERM_{\mathcal{H}}(S)) \leq L(h^*) + \mathcal{R}_m(\mathcal{H}) \approx L(h^*) + \sqrt{\frac{O\left(\text{capacity}(\mathcal{H})\right)}{m}} \quad (*)$$

We see that the error is larger than the best error in our class $\mathcal{H}$ by a term that scales with a measure of its capacity (colloquially, the 'amount' of models in the class), divided by the number of samples. In other words, the number of training examples required to learn h^* scales with the capacity of class $\mathcal{H}$.

What is this capacity? For binary classification, the capacity is the Vapnik-Chrvonenkis (VC) dimension of the class, $capacity(\mathcal{H}) = VCdim(\mathcal{H})$. The VC dimension is the largest number of points D that can be labeled (by models $h \in \mathcal{H}$) in every possible way. This is quite natural. A model class with high VC dimension (i.e. that contains predictors allowing *any* possible labelling of the set), does not have any inductive bias, so that learning is impossible (no free lunch). Learning becomes possible when the model class can be falsified, and the number of samples needed for learning is the number required to falsify this assumption on the model class $\mathcal{H}$. For linear classifiers over d features, $VCdim(\mathcal{H}) = d$. In fact, if the model class $\mathcal{H}$ can be parameterized with d parameters, the VC dimension is usually $VCdim(\mathcal{H}) = \tilde{O}(d)$. It is always true that the VC dimension is bounded by the logarithm of the cardinality of $\mathcal{H}$:

$$VCdim(\mathcal{H}) \leq \log|\mathcal{H}| \leq \#\text{bits} = \#\text{params} \cdot \frac{\#\text{bits}}{\#\text{params}}.$$

Thus we expect that if we encode the parameters of our model with a fixed number of bits, the VC dimension of the model scales with the number of parameters. Another way to produce model classes with finite capacity is employ regularizers in our learning rule. For example, it can be shown that for linear predictors with norm $||w||_2 \leq B$ (with logistic loss and normalized data), $capacity(\mathcal{H}) = B^2$.

Looking back at $(*)$, we see that learning, and machine learning in particular, requires model (hypothesis) classes $\mathcal{H}$ that are expressive enough to approximate reality well (contain h^* with low generalization error), but also have a small enough capacity to allow for good generalization. The approximation error is defined by the error of the best model in our class h^*, and the estimation error is the excess over it, as the learning rule can choose a different, worse model given the empirical data.

Usually however, our learning protocols do not represent a flat inductive bias over some model class. We often think in terms of a complexity measure $c: \mathcal{Y}^{\mathcal{X}} \to [0, \infty]$, which is any ordering of predictors h. Some measures of complexity include the degree of polynomials is our model class, assumptions of sparsity or low norm $||w||$. The associated inductive bias is that $\exists h^*$ with low complexity $c(h^*)$ and low error $L(h^*)$. This inductive bias suggests another learning rule, *structural risk minimization*:

$$SRM_{\mathcal{H}}(S) = \arg\min_{h\in\mathcal{H}} L_S(h), c(h)$$

This learning rule attempts to minimize two functions, which naturally introduces a trade-off between them. At best, the learning rule achieves a predictor that sits on the Pareto frontier that trades-off generalization error and complexity. Any predictor on that line cannot improve either the error or complexity without worsening the other. You can get to this frontier by considering a regularization path, i.e. minimizing $L_S(h) + \lambda c(h)$, and varying the regularization amplitude λ in the range $[0, \infty]$. Equivalently, one can attempt to minimize the error L_S such that $c(h) \leq B$. Note that this learning rule retrieves a multitude of candidates along the Pareto frontier. We can choose the best of them according to their performance on a cross validation sample.

For the SRM learning rule, we get a similar guarantee to $(*)$ on the generalization error, although

$$L(SRM_{\mathcal{H}}(S)) \leq L(h^*) + \sqrt{\frac{O\left(capacity\left(\mathcal{H}_{c(h^*)}\right)\right)}{m}}.$$

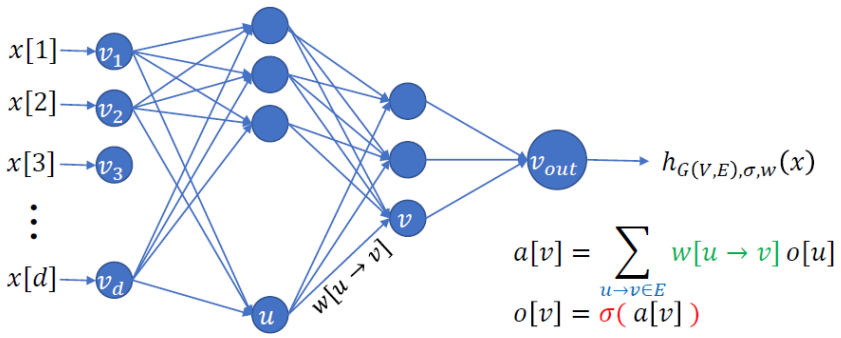

Figure 1: Feed-forward neural network.

Another way to think of this complexity measure as opposed to a flat hypothesis class is that it gives rise to a hierarchy of hypothesis classes which are sub-level sets of the complexity measure. The guarantee for SRM gives us an upper bound on the loss based on the best model in the class h^*, and its complexity measure in that class; better predictors are obtained if h^* lives in a small level set of the complexity measure. A good model class $\mathcal{H}$ in this approach not only contains a model that approximates reality h^*, but does so at lower level-set of a complexity measure.

1.3 Inductive bias in deep learning

Deep learning is learning with a particular inductive bias, a flat hypothesis class in the form of a feed-forward neural network. A feed-forward neural network (Fig. 1) is described by a directed graph $G(V, E)$ with nodes (neurons) indexed by vertices V. These nodes are subdivided into three types:

- *Input nodes* $v_1, ..., v_d \in V$ with no incoming edges, whose is $o[v_i] = x_i$.
- *Output node* $v_{out} \in V$, whose output is the model function $h_w(x) = o[v_{out}]$.
- *Hidden nodes* are all the rest of the nodes, which receive inputs from incoming edges (from a previous layer) and produce outputs to outgoing edges (to the next layer).

The network also has an *activation function* $\sigma : \mathbb{N} \to \mathbb{N}$ that describes the non-linearity of the neural network; a popular choice is the rectified linear unit (ReLU), $\sigma_{ReLU} = [z]_+$. Finally, the edges of the network, each connecting 2 nodes u, v, are called weights $w[u \to v]$ for each edge $u \to v \in E$. A choice of architecture, weights and activation function uniquely describe a predictor function $h_w(x)$. These models were historically developed by McCulloch and Pitts to describe computation in the 1940's [19]. They where able to show these models can perform complex computations.

In deep learning, we fix the architecture and activation function σ, and learn the weights from data. Thus the model class is given by $\mathcal{H}_{G(V,E),\sigma} = \{f_w(x) = \text{outputs of a network with weights } w\}$. We want to understand the capacity of these models, as well as their *expressivity* (how well they represent reality).

As noted before, the capacity is roughly given by the number of learned parameters, which here is the number of edges $|E|$. The VC dimension of these networks with threshold activation is $VCdim(\mathcal{H}_{G(V,E),sign}) = O\left(|E| \log |E|\right)$. However, for other activation functions, it is actually possible to get a capacity which is much higher than the number of parameters. For example, the VC dimension of these networks with sine activation is infinite, even for a single hidden node. We do not use these kind of activation functions. More useful activation functions are, for example the sigmoid function $\sigma(z) = (1 + \exp(z))^{-1}$, whose VC dimensions is bounded by $VCdim(\mathcal{H}_{G(V,E),sigmoid}) \leq O\left(|E|^4\right)$, or ReLU function, for which $VCdim(\mathcal{H}_{G(V,E),ReLU}) \leq O\left(|E| \log |E| l\right)$, with l the network depth. One can limit the capacity by discretizing the weights, e.g. if $w \in [-B, ..., B]$, $VCdim \leq 2|E|B$. As we've seen, the fact that these network models can have a large capacity is not necessarily good, because capacity comes at the expense of inductive bias.

What about expressivity? Feed-forward neural networks can represent any logical gate, and thus any function over $\mathcal{X} = \{\pm 1\}^d$ (as proved by Turing). Define the class $CIRCUIT_n[depth, size]$ as all functions

$f : \{\pm 1\}^n \to \{0,1\}$ that can be implemented with at most *size* AND, OR and NOT gates, and longest path from input to output at most *depth*. We know that circuits can represent any function, but only if we are allowed to select an appropriate gate architecture. In neural networks, we keep the architecture fixed and only vary the weights.

Claim 1.1. *A neural network with fixed architecture can learn the function of any circuit:*

$$CIRCUIT_n[depth, size] \subseteq \mathcal{H}_{G(V,E),l=depth,k=size,\sigma=sign}$$

Where we use a fully connected neural net with $l = depth$ layers and $k = size$ nodes in every layer.

This can be done easily, if we choose the weights of each edge to be ± 1 if the edge is connected in the circuit (with /o without a NOT gate in between), 0 otherwise. The bias terms are chosen $fan\_in - 1$ for AND gates, $1 - fan\_in$ for OR gates. The weights essentially describe which wires exist in the circuit. Thus neural networks can represent any binary function.

More generally, we have a *universal representation theorem*: Any continuous function $f : [0,1]^d \to \mathbb{N}$ can be approximated to within ϵ by a feed-forward network with sigmoidal (or almost any other) activation function and a single hidden layer. This shows that as a model class, feed-forward neural nets are extremely expressive and can represent any reality. However, representing functions may require huge networks, e.g. with layer widths exponential in d. The relevant question is not what a network of arbitrary size can represent, but what small networks can represent.

Small networks can represent intersections of half-spaces (using single hidden layer, each neuron corresponds to a half-space and the output neuron performs AND) and unions of intersections of half-spaces (with two hidden layers: half-spaces→OR→AND). However, the main compelling reason to use them is *feature learning*: Linear predictors over (small number of) features, in turn represented as linear predictors over more basic features, that in turn are also represented as linear predictors. In essence, the network builds up a hierarchy of predictors that progressively manage more abstract features of the data. In the case of image data, this is typically presented as early layers learning simple features (edges in images), and later layers building up on the simpler features to represent higher-level, semantic ideas (cars, birds, etc.).

Interestingly, a feed-forward neural network can represent any time T computable function with network of size $\tilde{O}(T)$ using a depth-T network. This is true since anything computable in time T is also computable by a logical circuit of size $\tilde{O}(T)$. This realization has broad implications for machine learning.

Machine learning is an engineering paradigm (of being lazy): Use data and examples, instead of expert knowledge and tedious programming, to automatically create efficient systems that solve complex tasks. Therefore, we only care about a model (predictor) h if it can be implemented efficiently. A good learned model only needs to compete with a programmer, producing results that are at least as good as a programmed model in a competitive (model evaluation) time.

In this case we have a free lunch: the model class $TIME_T$ - all functions computable by at most time T, has capacity $O(T)$, and hence learnable with $O(T)$ parameters (e.g. using ERM). Even better: the model class $PROG_T$, all functions programmable up to length of code T, also has capacity $O(T)$. This is relatively clear, because the length T bounds the number of bits needed to represent all these functions. Unfortunately, ERM with respect to $PROG_T$ is uncomputable. Modified ERM for $TIME_T$ (truncating exec. time) can be computed, but is NP-complete. If $P = NP$, we can have universal learning (free lunch). If $P \neq NP$, i.e. if there exist one-way functions that are easy to compute but hard to learn, there is no poly-time algorithm for ERM over $TIME_T$.

We thus unfortunately conclude that the free lunch is only possible if $P = NP$. This realization gives rise to the computational no free lunch theorem: For every **computationally efficient** learning rule A, there is some reality $\mathcal{D}$ such that there is some computationally efficient (poly-time) h^* with $L_S(h^*) = 0$, but $\mathbb{E}[L(A(S))] \approx 1/2$. In other words, our learning rule A can find an efficient h^*, but there are no guarantees on its generalization.

This leads us to revise our requirements of inductive bias; we have to assume that not only that reality $\mathcal{D}$ supports good generalization, but also that the learning algorithm A runs efficiently. The capacity of $\mathcal{H}$ or the complexity measure $h(c)$ are not sufficient inductive bias if ERM / SRM are not efficiently implementable, or if implementation does not always work (i.e.runs quickly but does not achieve ERM / SRM). Note that we switched from discussing learning rules (arbitrary mappings from sample to model), to talking about *learning algorithms*, an actual implementable process that chooses such a model.

Going back to neural networks, we completely understand them from a statistical perspective (in terms of capacity and expressivity). The problem with them relates to computation; computing the ERM for feed-forward neural nets is a non-convex optimization problem, and no known algorithm is guaranteed to work. We know that learning in neural nets, even in the simplest cases (2-hidden units in one hidden layer), is NP-hard. Even if reality is well-approximated by a small neural net, and you tried optimizing a larger neural net (which has more degrees of freedom), optimization is easy but ERM is still NP-hard. Unfortunately, there is nothing one can do to efficiently solve this computational problem, which is essential to neural nets precisely because of their expressive power (computational no-free lunch). Even if a function is exactly representable with single hidden layer and $\Theta(\log d)$ nodes, even with no noise, and even if we take a much larger network or use any other method when learning: no poly-time algorithm can ensure better-than-chance prediction, see e.g. [14, 10].

And nevertheless, deep learning does work! We have seen that from a statistical and computational perspectives, performing ERM on short programs (or short runtime programs) and learning with deep networks is equivalent. Both approaches are universal and approximate reality with reasonable sample complexity. They are both NP-hard, and provably hard to learn with any learning rule (subject to cryptographic assumptions). However there is no practical way to optimize over short programs, as e.g. there is no practical local search over programs. In contrast, deep neural nets are often easy to optimize; they are continuous models, amenable to local search (gradient descent, SGD), and enjoy much empirical success. In the worst case, deep learning is provably impossible, and yet, we are constantly reminded that deep learning is possible. There is a certain magical property of reality that makes feed-forward neural networks, and we have just started to scratch the surface of what that property is. However, we know for sure what it isn't: it is not the property that reality is well approximated by neural networks.

1.4 Deep learning in practice

As we have seen, deep neural networks can represent any function, and indeed have been shown to fit random data with perfect (training) accuracy [34]. However, when trained on real data, these networks do successfully generalize, even when over-parameterized.

We thus have a learning rule $A(S)$ that is able to achieve perfect training accuracy for any data set, even with random labels $L_S(A(S)) = 0$. On the other hand, it is able to generalize for real data $S \sim \mathcal{D}^m$ sampled from a reasonable reality $\mathcal{D}$, achieving low $L(A(S))$.

Perhaps we should not be surprised about this, as other learning algorithms do show similar behavior. A 1-Nearest Neighbor classifier, if realizable by a continuous h^* (i.e. $L_S(h^*)$), then for an infinite sample size ($|S| \to \infty$), it is consistent with zero generalization error $L(1 - \mathrm{NN}(S)) \to 0$. Similarly, a Hard Margin Support Vector Machine (SVM) with a Gaussian kernel (or some other universal kernel), or more generally, minimization of a norm for consistent solutions, also tend to generalize despite having vanishing training error: $\arg\min ||h||_K$ such that $L_S(h) = 0$. Let us consider a linear case where

$$w = \arg\min ||w||_2 \quad s.t. \quad \langle w, \phi(x_i) \rangle = y_i$$

In this case, our SVM model does not have a flat inductive bias, but the norm of the weights w adapt to the level of complexity inherent in the data. If reality is represented by a solution with small norm, then the learning rule will achieve a solution with low complexity measure and therefore generalize. However, if we try to fit random labels, we can only fit a model with a high norm (high complexity measure), and it will fail to generalize. We can always train SVMs with zero training error $L_S(h) = 0$. If $\exists h^*$ with zero generalization error $L_{\mathcal{D}}(h^*) = 0$, it will be achieved we sample complexity $|S| = O\left(||h||_K^2\right)$. Another example for this generalization is found in Minimum Description Length (MDL): A program optimized for its length $\arg\min |\text{program}|$ with $L_S(\text{program}) = 0$, is able to achieve a generalization error $L(MDL(S)) \leq O\left(\frac{|\text{program}|}{|S|}\right)$. That is, a short program only requires a sample complexity proportional to its length.

These examples and the ability of deep nets to generalize implies that the size of the network is not a good measure of model complexity. This is not a new idea; it was already realized in the 1990s that kernel regression works for infinitely many features, because we rely on norm for complexity control rather than the dimensionality. It was shown by P. Bartlett in 1996 [2] that the complexity of a neural network is not controlled by the number of weights but by their magnitude. In fact, neural networks have many solutions

for w so that $L_S(w) = 0$, many of which have high generalization errors. These solutions tend to have high w-norms. However, the solutions found in practice for neural networks using gradient descent do generalize well, and tend to have small norms, even without explicitly regularizing for low norm solutions.

Where is this implicit regularization coming from? We will try to understand this in the simplest model possible - linear regression. Consider an under-constrained least squares problem ($n > m$):

$$\min_{w \in \mathbb{R}^n} ||Aw - b||^2 \quad , \quad A \in \mathbb{R}^{m \times n}$$

In under-constrained cases there are many choices of w for which the sum of squares vanish. Imagine solving this problem with gradient descent, initialized at $w = 0$. Gradient descent will definitely succeed, as this is a convex problem, and find a vanishing solution, but which solution?

Claim 1.2. *Gradient descent (or SGD, conjugate gradient descent, BFGS) will converge to the least norm solution* $\min_{Aw=b} ||w||_2$. *The proof follows from the iterates always being spanned by the rows of A (more details soon).*

While we did not tell the algorithm to prefer solutions with small norms, it does in fact find the solution that minimizes this norm. This implicit regularization comes directly through the optimization process. In general we find that the optimization algorithm minimizes some norm or complexity measure, but which complexity measure?

2 Lecture 2 : Implicit bias and benign overfitting

2.1 Finishing lecture 1

To review, the high-level questions we're trying to answer are:

1. How much of what we've seen so far (in Lecture 1) fits within our classic understanding of statistical learning?
2. What questions do we need to answer to put it within our standard understanding?
3. And what goes *beyond* our standard understanding?

One thing we've seen so far is that huge models (so large that they *could* fit even random labels) can still generalize. This is fine; it's the same type of behavior we get with something like a hard-margin SVM or a minimum-norm predictor. What's going on in these cases is that the x-axis is not actually *capacity*. The real *measure of complexity* is some kind of norm.

What is this norm? Well, we are really abusing the term "norm"; what we *really* mean is some *measure of scale* that might not satisfy the rigorous definition of a norm. We can get implicit complexity control just from the optimization algorithm. For example, when we optimize an underdetermined least squares problem using gradient descent, we get the minimum-norm solution; that just comes from the optimization algorithm. So, we need to ask: what *complexity measure* is being minimized, and how does gradient descent minimize it?

We ended last time by saying that if we change our optimization algorithm without changing the objective function, we're actually implicitly minimizing some *other complexity measure*, which will change our inductive bias and thus our generalization properties. As some examples, we can compare the test performance of SGD and Path-SGD as in [25] and SGD vs. Adam as in [32]. In all cases, they reach the same final training error, but they have different final test performance values. In other words, they're reaching different global minima of the training objective. The story we're seeing is that the inductive bias is determined by the bias of the optimization algorithm. In other words, if we have a training loss landscape with *many global minima*, and we start optimizing on some hill, *different* optimization algorithms will move down the "hill" *differently* and reach *different* "beaches" (0 loss).

An illustrative example of different optimization algorithms inducing different regularizers can be seen by studying gradient descent vs. coordinate descent. Gradient descent will get to the minimum ℓ_2 norm, whereas coordinate descent will get to an approximately minimum ℓ_1 norm solution. In a high-dimensional system, these two norms are extremely different; ℓ_1 induces sparsity, and ℓ_2 does not. This difference is significant, especially when we think about deep learning in terms of feature learning (which Nati thinks is what's really going on). *Feature selection* (i.e., from a long list of features, select the relevant ones) is just sparsity, and ℓ_1 regularization can achieve it. In deep learning, we want to do feature *learning*, not just feature selection. But what *is* finding new features? We can think of a continuous set of possible features, and we want to select good features from that infinite feature set. So as long as we're not too worried about infinities, there's not much difference between feature selection and feature learning. And you can do that better with ℓ_1; ℓ_2 will never give you anything that is sparse. Thus, this can be a huge difference, and all the inductive bias is coming from the algorithm here. In fact, here is the perspective on deep learning we shall take here:

Deep networks can approximate any function. We kind of dismissed our universal approximation results last time because they require huge networks with seemingly unrealistic capacity. But maybe we actually *are* using networks that are large enough to capture all functions (since we are, at least on the data we see, able to capture all functions). So maybe we *are* essentially optimizing over the space of all functions. In that case, minimizing empirical error with respect to all functions doesn't make any sense; it's really easy to optimize the empirical error with respect to all functions, since we can just memorize the training examples and not do anything anywhere else in the domain. But in deep learning, we optimize over all functions *with particular search dynamics*, and although we do get to a function that has 0 training error, we don't get to just *any* 0 error solution. *How* we optimize over the space of all functions determines which directions we like to take. Roughly speaking, we'll get to a 0-error solution that's "close" (to the initialization point) in some sense with respect to our geometry.

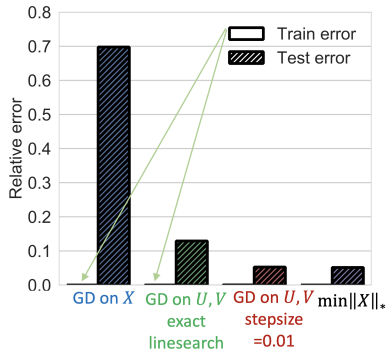

Figure 2: Matrix completion. Relative error is the squared error compared to the squared error of the null predictor.

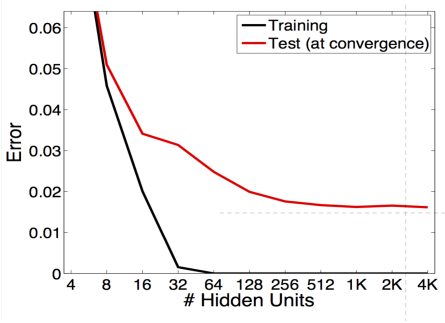

Figure 3: Error vs. number of hidden units.

2.2 Examples

With this perspective in mind, we will now begin the main content of Lecture 2. We'll start by examining some examples where we're optimizing over the space of all functions, but because of the *way* we're optimizing over the space of all functions, we're actually getting good generalization.

2.2.1 Matrix completion

In matrix completion, we have some observations from a matrix. We want to complete the matrix and uncover the remaining entries. You should think of the matrix as having some structure (e.g., some low-rank structure). Formally, the matrix completion problem is:

$$\min_{X \in \mathbb{R}^{n \times n}} \|\text{observed}(X) - y\|_2^2. \tag{1}$$

In some sense, it's very easy to solve this optimization problem. We can complete the observed entries and put 0 everywhere else, but it won't help us recover the unobserved entries. The problem is underdetermined. So what do we do? We can run gradient descent directly on X, or - alternatively - we can replace X with $UV^\top$ (U, V full dimensional, therefore no rank constraint on $X = UV^\top$) and run gradient descent on U, V. Figure 2 compares the results of these two procedures when the ground-truth matrix X^* has rank 2. We also see how slight variations in gradient descent on U, V change which solution we converge to. But the bigger effect is coming from the reparameterization (from X to $UV^\top$).

So what is going on here? In this case, we have a good understanding, though not complete. The initial proposal in [12] was that gradient descent is converging to a low nuclear norm solution - i.e., nuclear norm is the relevant complexity measure that gradient descent is minimizing. We know that minimizing the nuclear norm can give you good generalization when you have an approximate low-rank matrix. Minimization of the nuclear norm is proved rigorously in some cases (e.g., under restricted isometry property (RIP) in [15]) and turns out to not always be the case (e.g., see counterexample in Example 5.9 in [16]).

2.2.2 Single overparameterized linear unit: $\mathbb{R}^d \to \mathbb{R}$

If we train a single unit with gradient descent using the logistic ("cross entropy") loss, we converge to the max-margin separator (the hard-margin SVM predictor), which has an implicit ℓ_2 norm in it:

$$w(\infty) \propto \arg\min \|w\|_2 \quad \text{s.t.} \quad \forall i \quad y_i \langle w, x_i \rangle \geq 1. \tag{2}$$

This holds regardless of initialization. We will go over this result in detail in a future lecture.

2.2.3 Deep linear network: $\mathbb{R}^d \to \mathbb{R}$

Now, we can ask what happens in a deeper network (with only linear activations). Let w denote the weights of all the layers. Then our deep linear network implements the same linear mapping as above:

$$f_w(x) = \langle \beta_w, x \rangle. \tag{3}$$

When we run gradient descent on w (vs. β, as we did above), one might think that this reparameterization could affect the search geometry. However, in this case, the inductive bias is actually the same as above:

$$\beta_{w(\infty)} \propto \arg\min \|\beta\|_2 \quad \text{s.t.} \quad \forall i \quad y_i \langle \beta, x_i \rangle \geq 1. \tag{4}$$

2.2.4 Linear convolutional networks

Things get more interesting in the linear convolutional case, though. Let's consider the following linear convolutional network:

$$h_l[d] = \sum_{k=0}^{D-1} w_l[k] h_{l-1}[d + k \bmod D], \qquad h_{\text{out}} = \langle w_L, h_{L-1} \rangle, \tag{5}$$

which is still just a reparameterization of our original linear function from $\mathbb{R}^d \to \mathbb{R}$. Now we can ask what happens when we train this model using gradient descent.

Single layer ($L = 2$). With a single hidden layer, training the weights with gradient descent implicitly minimizes the ℓ_1 norm in the frequency domain:

$$\arg\min \|\operatorname{DFT}(\beta)\|_1 \quad \text{s.t.} \quad \forall i \quad y_i \langle \beta, x_i \rangle \geq 1, \tag{6}$$

where DFT denotes the discrete Fourier transform. In other words, we obtain sparsity in the frequency domain.

Multiple layers. With L layers, training the weights with gradient descent converges to a critical point of

$$\|\operatorname{DFT}(\beta)\|_{2/L} \quad \text{s.t.} \quad \forall i \quad y_i \langle \beta, x_i \rangle \geq 1, \tag{7}$$

where $\|\cdot\|_{2/L}$ denotes the $2/L$ quasinorm. It is not technically a norm, but it *is* formally defined, and it's even more sparsity-inducing than ℓ_1. Thus, increasing the depth induces more and more sparsity in the frequency domain. See [11] for more details.

2.2.5 Infinite-width ReLU network

Now let us look at all functions (not just linear) from $\mathbb{R}^d \to \mathbb{R}$. In order to represent them with a neural network, we have to introduce nonlinearities (e.g., ReLU). If we let a single-hidden-layer ReLU network be wide enough, we can approximate all functions. So let us learn using infinite-width ReLU networks.

Functions h from $\mathbb{R} \to \mathbb{R}$. Gradient descent on the weights implicitly minimizes

$$\max\left(\int |h''| dx, |h'(-\infty) + h'(+\infty)| \right). \tag{8}$$

This would be a very sensible penalty to choose, since it is kind of a smoothness-inducing penalty. The interesting thing is that we didn't explicitly choose it; it came to us just from gradient descent on this parameterization.

Functions h from $\mathbb{R}^d \to \mathbb{R}$. Gradient descent on the weights implicitly minimizes

$$\int |\partial_b^{d+1} \operatorname{Radon}(h)|. \tag{9}$$

Once again, we get some kind of sensible smoothness penalty. This result is rigorous for logistic loss (doesn't depend on initialization or learning rate). For squared loss, we don't know how to analyze it exactly, although we expect something similar. See [27], [26], [7] for more details.

2.2.6 Takeaways from these examples

What have we been doing? The game here is that we want to understand what happens in the space of functions. The inductive bias in parameter space is relatively simple (ℓ_2 or something similar, often). But what we really care about is what happens in function space, which can be very rich. A large part of the optimization problem is the architecture. The classical view is that the architecture is important because it limits what functions you can get; however, that's not the case here. The architecture is important because it determines the mapping from parameter space to function space and is the *biggest* contributor to the geometry by which you're searching in function space.

The next most significant thing that affects the inductive bias is the geometry of the local search in parameter space (e.g., ℓ_2 vs. ℓ_1 in parameter space).

And the least significant thing (though still relevant) that affects the inductive bias is the set of optimization choices (e.g., initialization, batch size, step size, etc.).

Does gradient descent always minimize the ℓ_2 norm in parameter space? In all of these examples, we can get the same thing as gradient descent's implicit regularization using a minimum ℓ_2 norm on the weights (subject to fitting the data). In all of these examples, the complexity control in parameter space is very simple (it is just ℓ_2), and everything is coming from the parameterization. So is all this discussion of the implicit bias of gradient descent just ℓ_2 in parameter space, with everything coming just from the parameterization?

The answer is: sort of. You can get some asymptotic results showing that, for some restricted class of models with the logistic loss, everything boils down to ℓ_2. However, we'll also see that in many cases, this is not true, and you'll get something very different (e.g., under squared loss, or even under logistic loss non-asymptotically).

2.3 Why the red curve goes *down* in Figure 3

We now understand that, in Figure 3, we have complexity control coming from the algorithm, and this is what stops us from overfitting. The main question this led us to ask is: what is this complexity control? (which we just studied, in several examples.)

But there is *another* thing that is going on here. What we have studied so far explains why we might be able to generalize well even at a large number of hidden units (i.e., we have complexity control coming from the algorithm, even though we're optimizing in the space of all functions). However, we haven't yet explained why the red curve (test error) actually goes *down* as the number of hidden units increases. Recall, the x-axis is not complexity (complexity is something else - some norm-based complexity, probably).

Gaussian kernel. We can actually see similar behavior even with kernel methods. Let us consider the Gaussian kernel, which corresponds to an infinite-dimensional feature space. Let us think of what happens if we use a finite approximation to the Gaussian kernel. Concretely, we have the Gaussian kernel $\langle \phi_\infty(x), \phi_\infty(x') \rangle = e^{-\|x-x'\|^2}$ and the finite-dimensional feature mapping

$$\phi_d(x)[i] = \frac{1}{\sqrt{d}} \cos(\langle \omega_i, x \rangle + \theta_i).$$

The algorithm returns $A(S) = \arg\min \|w\|$ s.t. $L_S\left(x \mapsto \langle w, \phi_d(x) \rangle\right) = 0$, i.e., $\forall (x_i, y_i) \in S$, $y_i = \langle w, \phi_d(x_i) \rangle$. As $d \to \infty$, we approach the Gaussian kernel. Once we have more features than data points, we

can already get 0 training error. But we are not doing it with the Gaussian kernel yet; we are doing it with an approximation of the Gaussian kernel. If the RKHS norm induced by the Gaussian kernel is our "correct" complexity measure, then as $d \to \infty$, we are approximating it better and better. So we are minimizing a complexity measure that's a better and better approximation of the complexity measure we want. So as the dimensionality increases, the test error improves.

Matrix completion. We can also see this in matrix completion. Suppose X is $n \times n$, we observe nk entries, and we parameterize X as $UV^\top$, where $U, V \in \mathbb{R}^{n \times d}$ (thereby adding a rank-d constraint). As above, d captures the quality of our approximation. Suppose the "right" complexity measure is nuclear norm. We have two different regimes:
- If $d < k$, our algorithm returns $\arg\min \hat{L}(X)$ s.t. $\text{rank}(X) \leq d$.
- If $d > k$, our algorithm returns $\arg\min \|X\|_*$ s.t. $\hat{L}(X) = 0,\ \text{rank}(X) \leq d$.
So as we increase the rank constraint, the test error becomes better and better. As we increase the rank, we're getting closer to what we really want, which is a full-rank low nuclear norm matrix. So d is not our complexity measure; rather, it's the dimensionality of the approximation to the infinite-dimensional system.

Commentary. We claim that this is what's happening in neural networks too. The real object we should be learning with is an infinite-size network. But we cannot really represent an infinite-size network? Instead, we represent a truncated, finite-dimensional representation of it. As our truncation becomes finer and finer, our representation becomes better and better. We want the size to be so big that we essentially have a good approximation to the infinite-size model. Our methods should not rely on the size of the approximation being small. So we should start by understanding the infinite size and then worry about the question: "how large do we need our model to be in order for it to be considered infinite?".

2.4 What we don't yet fully understand: benign overfitting

However, there's still one thing that doesn't really fit our understanding. And this is something we were completely blind to for many years, even though it was right in front of us. It was pointed out only fairly recently by Misha Belkin. This one thing is that we are getting good generalization even though we are insisting on a 0-training error solution in *noisy* situations (situations where the approximation error is nonzero). In particular, in Figure 4, we want to balance complexity and training error, so we know we want to be somewhere on this regularization frontier. But the solutions we are finding are the minimum-complexity 0-error solutions - an extreme point of the frontier, and we are seeing this even in fairly noisy cases, where we would expect to be somewhere on the frontier that strikes more of a balance between complexity and training error.

To understand this a bit better, let us return to fitting noisy data with polynomials, where complexity is degree of the polynomial. As we increase the degree of the polynomial, we can decrease the training error. In this case, as seen in Figure 5, we can get 0 training error with a degree-9 polynomial. But we are fitting the noise and getting bad generalization. This is captured by the classic U-shaped red curve in Figure 5. At some point, we start overfitting - which we will define as fitting the noise. At that point, the test error starts to become worse because the estimation error begins to dominate. The conventional wisdom is that we should never insist on 0 training error, because it will fit the noise and generalize poorly.

Connection with double descent. Arguably, the first paper to discuss the above phenomenon was [4], which introduced the notion of "double descent" (Figure 6). At this point, we probably understand about 95% of double descent, which has little to do with the question we just asked about fitting the noise. So let us briefly discuss the double descent phenomenon, before reaching the remaining 5% that we do not yet understand.

Why are we getting double descent? Let us think of a least squares problem in dimensionality d and a fixed number of training examples. The x-axis is the dimensionality. The y-axis is the error of the ERM solution - but not just any ERM solution: if the problem is overdetermined, find the solution that minimizes the reconstruction error; if the problem is underdetermined, find the minimum Euclidean norm 0-training-error solution. Until the interpolation threshold, the x-axis really is complexity control. After the interpolation

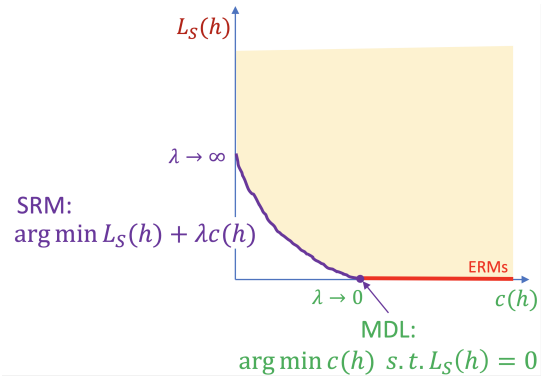

Figure 4: Trading off training loss and complexity.

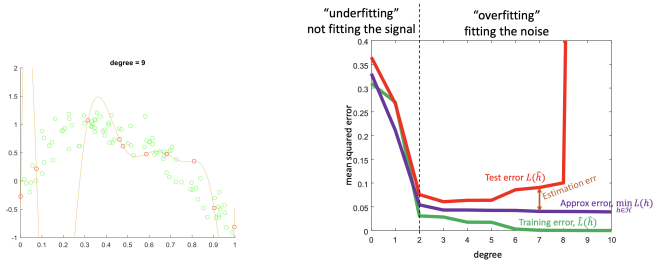

Figure 5: Our classical understanding of overfitting.

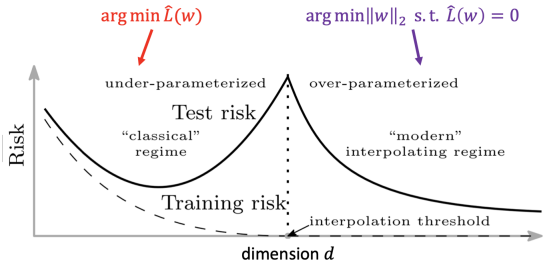

Figure 6: The double descent phenomenon.

threshold, the x-axis is no longer complexity control; rather, it is the degree of approximation. As we just discussed, it is not surprising that the test error improves as the approximation becomes better.

The fact that we get an increase followed by a decrease is not surprising. What *is* surprising is that we get good generalization even though we insist on 0 training error in a fairly noisy situation. The main experiment that Misha and coauthors did that probably convinced many people is from [5], in which high levels of noise were added to synthetic data. When perfectly fitting this noisy data using different methods (all methods get 0 training error), the test error is substantially below the null risk. In particular, the phenomenon we are seeing is that we *are* perfectly fitting the noise, but this overfitting is not harmful, as in Figure 5. Rather, the overfitting is *benign*. We are fitting the noise in a way that has a kind of measure-0 effect; fitting the noise does not ruin the fit in other places. We are maybe not *gaining* anything from fitting this noise, but it doesn't hurt us either (it's benign).

Open Questions In what situations is overfitting harmful, and it what situations is it benign? In least squares, we can get both kinds of behaviors in a now-predictable way (the result of research in the past 2-3 years). We know that it relates to a measure of effective rank. Characterizing more generally (beyond least squares) when overfitting is harmful vs. benign is a big challenge that we have now, which is fairly different from how we used to think about overfitting.

In some sense, the most practical implication of benign overfitting is that we don't have to worry too much about selecting the right value of the regularization parameter λ; we have a whole regime of good values of λ. In many cases in practice, though, we see something in between benign and harmful overfitting (but much closer to benign). This matches what we see empirically: that adding a bit of regularization *can* be helpful by a bit, but we can still have good performance without explicit regularization.

2.5 Summary

What fits our understanding and what doesn't?

In linear models, which is the only case we really understand, we can see how we can generalize even

when we are able to fit random labels. We can understand how the complexity control comes purely from the optimization algorithm and how we can generalize better with bigger and bigger models, since they provide a better and better approximation to what we really want. Therefore, to fit each case within our classic understanding, we "just" need to ask: 1. What is the complexity measure? 2. How is it minimized by the optimization algorithm? 3. How does minimizing it ensure good generalization? Thus, the only thing that truly goes *beyond* our standard understanding is the phenomenon of benign overfitting.

3 Lecture 3 : Statistical learning and stochastic convex optimization

In this lecture, we will explore the connections between optimization geometry and generalization in the well-understood convex case. Specifically, we will derive generalization guarantees based on this geometry. Many of the concepts and proofs reproduced here can be found in classical references on statistical learning [30, 28, 20] and optimization [6, 24].

3.1 Learning and optimization for convex problems

Recall that the goal of supervised learning is, for given input and output spaces $\mathcal{X}, \mathcal{Y}$, to find a predictor function $h_w : \mathcal{X} \to \mathcal{Y}$, parametrized by a vector w, with low population error defined by:

$$L(h_w) = \mathbb{E}_{x,y}\left[l\left(h_w(x); y\right)\right], \tag{10}$$

for some hidden joint density $p(x, y)$ and a chosen loss l function measuring the prediction error for a given pair (x, y). In what follows, we will denote $\mathcal{H}$ the chosen hypothesis class of functions that can be represented with the parameter w, and will consider the same notation for optimization on $\mathcal{H}$ or the corresponding parameter space. Since we do not have direct access to the joint distribution $p(x, y)$, we collect a dataset $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of m i.i.d. samples from p, and use it to estimate $L(h_w)$ (equivalently denoted $L(w)$) with the corresponding empirical distribution. This leads to the *empirical risk minimization* (ERM) problem to estimate a parameter vector $\hat{w}$:

$$\hat{w} = \arg\min_{h \in \mathcal{H}} \hat{L}(w) := \arg\min_{h \in \mathcal{H}} \frac{1}{m} \sum_i l\left(h_w\left(x_i\right); y_i\right) + \lambda \Psi(w). \tag{11}$$

Here, the term $\Psi(w)$ is the regularizer or penalty, while the parameter $\lambda > 0$ tunes the regularization strength. The purpose of this term is mainly to prevent overfitting. Intuitively, a well chosen Ψ will increase if a corresponding complexity measure of the model increases. Typical examples include the ℓ_2 norm, enforcing regularity at the functional level, or the ℓ_1 norm, inducing sparsity at the level of the parameters w. Equivalently, (11) can be written as

$$\hat{w} = \arg\min \frac{1}{m} \sum_i l\left(h_w\left(x_i\right); y_i\right) \tag{12}$$

$$\text{such that} \quad \Psi(w) \leq B \tag{13}$$

where B depends on the regularization coefficient λ. The main two sources of error that need to be controlled in empirical risk minimization are the optimization and generalization error. The latter is handled using uniform convergence tools to control the convergence rate of the empirical risk towards the population one, when the parameters are constrained to the sublevel sets defined by the penalty.

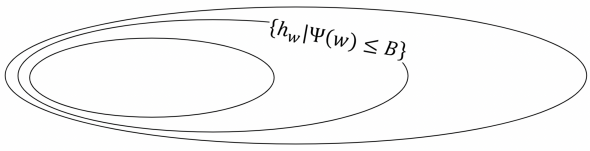

Figure 7: Graphical representation of sublevel sets of the complexity measure.

By uniform convergence we mean the following. Let us look back at the ERM learning rule (12). To ensure that it performs well with respect to the population error, we can bound the difference between the population and empirical errors uniformly over all predictors in the class. That is, for any $\epsilon > 0$, we need to quantify how many samples m are needed to ensure that $\sup_{\|\Psi(w)\| \leq B} \left|\hat{L}(w) - L(w)\right| < \epsilon$. For a given learning

problem, a dataset that ensures the latter inequality is said to be $\epsilon-$*representative*. It is straightforward to show that an $\frac{\epsilon}{2}$ representative training set ensures that

$$L(\hat{w}) \leq \min_{w} L(w) + \epsilon, \tag{14}$$

ensuring that the predictor $\hat{w}$ is a good proxy for the true minimizer. The standard way to achieve uniform control of the deviation between $\hat{L}(w)$ and $L(w)$ is to quantify the complexity of the hypothesis class $\mathcal{H}$ and regularity of the loss function, and to relate these quantities to the required sample complexity. A useful complexity measure for hypothesis classes is the *Rademacher complexity*, defined in its empirical form as

$$\mathcal{R}(\mathcal{H} \circ S) = \frac{1}{m}\mathbb{E}_{\boldsymbol{\sigma}\sim\{\pm 1\}^m}\left[\sup_{h\in\mathcal{H}} \sum_{i=1}^{m} \sigma_i h(z_i)\right] \tag{15}$$

where $\boldsymbol{\sigma}$ is a random vector with i.i.d. Rademacher entries. The Rademacher complexity gives a distribution dependent alternative to the VC dimension discussed in Lecture 1, defined for any class of real-valued functions. We then have uniform convergence bounds similar to the ones presented using the VC dimension in Lecture 1, e.g., assuming that $|l(h, z)| < c$ for some positive constant c, for any $h \in \mathcal{H}$

$$L(h) - \hat{L}(h) \leq 2\mathcal{R}(l \circ \mathcal{H} \circ S) + c\sqrt{\frac{2/\delta}{m}} \tag{16}$$

with probability at least $1 - \delta$. The interested reader can find more details on the Rademacher complexity along with examples in Chapter 26 of [28]. In this lecture, we will directly derive optimization guarantees for the learning problem formulated as a stochastic optimization problem.

Example: Let us assume that the reality is captured by a low-norm linear predictor. Mathematically, this goes as follows:

$$\mathcal{H} = \{h_w(x) \to \langle w, x\rangle \mid \|w\|_2 \leq B\}. \tag{17}$$

For simplicity, let us assume that the data and the derivative of the loss function are bounded: $\|x\|_2 \leq 1$ and $\|\nabla l\| < 1$, and that l is convex. In that case, it can be shown straightforwardly that the empirical Rademacher complexity of the corresponding ERM problem verifies

$$\mathcal{R}(l \circ \mathcal{H} \circ S) \leq \sqrt{\frac{B^2}{m}}. \tag{18}$$

If we denote by $\hat{w}$ the $\arg\min$ of the empirical loss $\hat{L}$, we then reach the following generalization result:

$$L(\hat{w}) \leq \inf_{\|w\|\leq B} L(w) + O\left(\sqrt{\frac{B^2}{m}}\right). \tag{19}$$

In order to compute $\hat{w}$ we perform gradient descent on $\hat{L}(w) = \frac{1}{m}\sum_i l(\langle w, x_i\rangle, y_i)$, where $l(\cdot,\cdot)$ is the loss function. The iteration of the GD goes as follows

$$w^{k+1} = w^k - \eta\nabla\hat{L}(w^k).$$

The convergence rate of this algorithm with the optimal step-size, see e.g. [24], is described as:

$$\hat{L}(\bar{w}^T) \leq \inf_{\|w\|\leq B} \hat{L}(w) + O\left(\sqrt{\frac{B^2}{T}}\right)$$

However, in each iteration of the gradient descent, we need m gradient computations. Depending on the data this may be computationally costly. Instead, one may use stochastic gradient descent (SGD). In this case, we uniformly pick an example (x_i, y_i) and only calculate its corresponding gradient term. SGD iteration is written as

$$\bar{w}^{k+1} = \bar{w}^k - \eta\nabla l(\langle \bar{w}^k, x_i\rangle, y_i).$$

Thus at each iteration we subtract an unbiased estimator of the full gradient. Indeed, at one may check that

$$\mathbb{E}[\nabla l(\langle \bar{w}^k, x_i\rangle, y_i)] = \nabla \hat{L}(\bar{w}^k).$$

For the SGD algorithm we have the same convergence guarantee:

$$\hat{L}(\bar{w}^T) \leq \inf_{\|w\|\leq B} \hat{L}(w) + O\left(\sqrt{\frac{B^2}{T}}\right).$$

Combining this bound with (19) we have the following:

$$L(\bar{w}^T) \leq \inf_{\|w\|\leq B} \hat{L}(w) + O\left(\sqrt{\frac{B^2}{m}}\right) + O\left(\sqrt{\frac{B^2}{T}}\right).$$

Thus the error magnitude of the SGD and approximation error is the same. This means that we need to do at most m iteration of SGD because when $m > T$, the dominant term in the previous bound becomes $O(\sqrt{\frac{B^2}{m}})$.
The one pass SGD can also be viewed as an algorithm to minimize the population risk $L(w) = \mathbb{E}[l((w, x), y)]$. Indeed, the gradient term satisfies the following:

$$\nabla L(w^k) = \mathbb{E}_{x_i, y_i}\big[\nabla l(\langle w^k, x_i\rangle, y_i)\big].$$

The latter means that instead of this two-step scheme, we can analyze the generalization using the optimization guarantee directly for the population risk. Therefore we obtain the following

$$L(\bar{w}^T) \leq \inf_{\|w\|\leq B} L(w) + O\left(\sqrt{\frac{B^2}{T}}\right).$$

We cannot do more iterations than the number of data points, as we need to have independent samples from the population. Thus the number of iterations is again bounded by the sample size and therefore we get the same bound.

3.2 Stochastic optimization

Stochastic optimization problem is written as

$$\min_{w\in\mathcal{W}} F(w) = \mathbb{E}_{z\sim\mathcal{D}}[f(w, z)] \tag{20}$$

based on i.i.d. samples $z_1, z_2, \ldots, z_m \sim \mathcal{D}$. Here the distribution $\mathcal{D}$ is unknown and we do not have access to $F(w)$. But using the samples we can have estimates of F and ∇F. An instance of this problem is the general learning problem. It can be formulated as

$$\min_h F(h) = \mathbb{E}_{z\sim\mathcal{D}}[f(h, z)],$$

using the data samples $z_i \sim \mathcal{D}$, for $i = 1, \ldots, m$, where h is a mathematical object adapted to the particular model. Here is a short list of examples.

- In *supervised* learning we have $\mathcal{Z} = \mathcal{X} \times \mathcal{Y} = \{z = (x, y) \mid x \in \mathcal{X}, y \in \mathcal{Y}\}$. The function $h : \mathcal{X} \to \mathcal{Y}$ and $f(h, z) = l(h(x), y)$, where l is the loss function.
- In the *unsupervised* k-means clustering problem we have $z = x \in \mathbb{R}^d$ and $h = (\mu[1], \ldots, \mu[n]) \in \mathbb{R}^{d\times k}$. Here $h[i]$ is the center of i-th cluster. The objective function for this problem is defined as

$$f((\mu[1], \mu[2], \ldots, \mu[k]), x) = \min_i \|\mu[i] - x\|^2.$$

- The problem of *density estimation* can also seen as a stochastic optimization. Consider $z = x$ in some measurable space $\mathcal{Z}$ (e.g. $\mathbb{R}^d$). Then for each h, we define the probability density $p_h(z)$ and the objective function $f(h, z) = -\log p_h(z)$. The function F in this case is the KL divergence.

The fields of stochastic optimization and statistical learning have been developed in parallel in the 60s and 70s [31, 23].

Let us get back to the stochastic optimization problem (20). We saw two ways of solving this problem. The first is based on Sample Average Approximation (SAA) or the ERM. It essentially consists of collecting data $z_1, z_2, \ldots, z_m$ and estimating the expectation term with the empirical mean

$$\hat{F}_m(w) = \frac{1}{m} \sum_i f(w, z_i).$$

The other method is the Stochastic Approximation (SA) e.g. SGD. Here, we update w^i using $f(w^i, z_i), \nabla f(w^i, z_i)$ and previous iterates. In particular in SGD we have: $w^{i+1} = w^i - \eta \nabla f(w^i, z_i)$.

As mentioned previously, in machine (supervised) learning the objective function is the population risk $L(w)$. In this setting, the SGD can be applied in two ways. The first follows the direct SA approach (one-pass SGD).

1: Initialize $w^{(0)} = 0$
2: At iteration $t = 0, 1, \ldots, T$
3: Draw $(x_t, y_t) \sim \mathcal{D}$
4: $w^{(t+1)} \leftarrow w^{(t)} - \eta_t \nabla l(\langle w^{(t)}, x_t \rangle, y_t)$
5: Return $\bar{w}^{(T)} = \frac{1}{T} \sum_{t=1}^{T} w^{(t)}$.

For this algorithm we may obtain the following convergence guarantee[1]:

$$L(\bar{w}^{(T)}) \leq L(w^*) + 2\sqrt{\frac{B^2}{m}} + \sqrt{\frac{B^2}{T}}$$

. However, we may perform SGD on ERM. That is our optimization problem has the following form:

$$\min_{\|w\|_2 \leq B} \hat{L}(w) = \min_{\|w\|_2 \leq B} \frac{1}{m} \sum_{i=1}^{m} l(\langle w, x_i \rangle, y_i). \tag{21}$$

The alternative approach suggests the minimization scheme below.

1: Draw $(x_1, y_1), \ldots, (x_m, y_m) \sim \mathcal{D}$
2: Initialize $w^{(0)} = 0$
3: At iteration t, we pick randomly $i \in \{1, 2, \ldots m\}$.
4: $w^{(t+1)} \leftarrow w^{(t)} - \eta_t \nabla l(\langle w^{(t)}, x_i \rangle, y_i)$
5: Then, we may perform the step $w^{t+1} \leftarrow \text{proj}_{\|w\| \leq B} w^{t+1}$. although we may show that implicitly our iterate will converge to this set.
6: Return $\bar{w}^{(T)} = \frac{1}{T} \sum_{t=1}^{T} w^{(t)}$.

For this algorithm we may obtain the following convergence guarantee:

$$L(\bar{w}^{(T)}) \leq L(w^*) + 2\sqrt{\frac{B^2}{m}} + \sqrt{\frac{B^2}{T}}.$$

The several differences between these two schemes.

[1] All the guarantees are satisfied up to a constant.

- Since we need independent samples for the first scheme, it has as many samples as iterations, ($m = T$). This is not the case for the second method, as we fix initially the m samples and then choose from them. Thus it can have $T > m$ iterations.
- The direct SA approach does not require regularization and thus it does not have a projection step.
- The SGD on ERM method is explicitly regularized. The regularization of the direct SA approach hides in the step-size. Indeed, in order for the parameter w to have larger norm, one needs to choose larger step-sizes. In particular, if we choose $\eta_t = \sqrt{B^2 t}$, we get the following:

$$L(\bar{w}^{(T)}) \leq L(w^*) + \sqrt{\frac{B^2}{T}}$$

On the other hand, the SGD on ERM has the following generalization error bound:

$$L(\bar{w}^{(T)}) \leq L(w^*) + 2\sqrt{\frac{B^2}{m}} + \sqrt{\frac{B^2}{T}}$$

In both cases $L(w^*)$ is the value of the objective at the optimal point in the class: $L(w^*) = \min_{\|w\|_2 \leq B} L(w)$.

Where is the regularization? Although we mentioned the effect of the step-size on the regularization in the direct approach, it is still not clear why we observe this phenomenon. Let us look back at the standard GD. The gradient descent minimizes the norm $\|w\|_2$. Indeed. At each step of the gradient descent we minimize its linear approximation given by the gradient:

$$w^{(t+1)} = \arg\min_w \left\{ F(w^{(t)}) + \langle g^{(t)}, w - w^{(t)} \rangle \right\}.$$

The latter is linear function and thus its minimum is at infinity. Also, on the other hand, the linear approximation is not valid when we get far from the iterate $w^{(t)}$. That is why we add a square regularizer $\|w - w^t\|_2^2/2\eta$. We obtain

$$\begin{aligned} w^{(t+1)} &\leftarrow \arg\min_w \left\{ F(w^{(t)}) + \langle g^{(t)}, w - w^{(t)} \rangle + \frac{1}{2\eta} \|w - w^{(t)}\|_2^2 \right\} \\ &= \arg\min_w \left\{ \langle g^{(t)}, w - w^{(t)} \rangle + \frac{1}{2\eta} \|w - w^{(t)}\|_2^2 \right\} \\ &= w^{(t)} - \eta g^{(t)}. \end{aligned} \tag{22}$$

In order to better understand this for the SGD, let us first introduce the notion of stability.

3.3 Stability

Here we will be studying a notion of stability for loss functions, generically denoted $\hat{F}(w)$, defined as empirical sums of the form $\frac{1}{m}\sum_{i=1}^m f(w, z_i)$, for a given dataset $(z_i)_{1 \leq i \leq m}$. We start by defining the leave-one-out stability and replace-one-out stability, and then derive generalization bounds using these quantities, see e.g. [29].

Definition 3.1. *Let $\beta : \mathbb{N} \to \mathbb{R}$ be a monotonically decreasing function. A learning rule $\tilde{w}(z_1, \ldots, z_m)$ is* leave one out $\beta(m)$-stable *if*

$$|f(\tilde{w}(z_1, \ldots, z_{m-1}), z_m) - f(\tilde{w}(z_1, \ldots, z_m), z_m)| \leq \beta(m),$$

and replace one out $\beta(m)$-stable *if,*

$$|f(\tilde{w}(z_1, \ldots, z_{m-1}, z'), z_m) - f(\tilde{w}(z_1, \ldots, z_m), z_m)| \leq \beta(m),$$

where z' is a new independent sample from the hidden distribution $\mathcal{D}$.

For simplicity we will assume that the learning rule is symmetric. That is $\tilde{w}(z_1, \ldots, z_m) = \tilde{w}(\sigma(z_1), \ldots, \sigma(z_m))$, where σ is any permutation defined on $\{1, 2, \ldots, m\}$.

Theorem 3.1. *Define $\widehat{F}(w) := \frac{1}{m}\sum_{i=1}^{m} f_i(w)$. If $\tilde{w}$ is symmetric and $\beta(m)$ is stable then*

$$\mathbb{E}[F\left(\widetilde{w}\left(z_1, \ldots, z_{m-1}\right), z_m\right)] \leq \mathbb{E}[\widehat{F}\left(\widetilde{w}\left(z_1, \ldots, z_m\right), z_m\right)] + \beta(m)$$

Proof. By symmetry of $\widetilde{w}$ we have

$$\begin{aligned}
&\mathbb{E}_{z_1,\ldots,z_m}\left[f\left(\widetilde{w}\left(z_1, \ldots, z_{m-1}\right), z_m\right)\right] \\
&\qquad = \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}\left[f\left(\widetilde{w}\left(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_m\right), z_i\right)\right]
\end{aligned}$$

Using the stability of the function f, we get the following

$$\begin{aligned}
&\mathbb{E}_{z_1,\ldots,z_m}\left[f\left(\widetilde{w}\left(z_1, \ldots, z_{m-1}\right), z_m\right)\right] \\
&\qquad \leq \frac{1}{m}\sum_{i=1}^{m} \left(\mathbb{E}\left[f\left(\widetilde{w}\left(z_1, \ldots, z_m\right), z_i\right)\right] + \beta(m)\right) \\
&\qquad = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} f\left(\widetilde{w}\left(z_1, \ldots, z_m\right), z_i\right)\right] + \beta(m) \\
&\qquad = \mathbb{E}\left[\widehat{F}\left(\widetilde{w}_m\right)\right] + \beta(m)
\end{aligned}$$

□

This result yields generalization for stable learning rules. However, one needs to take into account that stability may be tricky in the learning problem. In the case, when the predictor interpolates the data, the empirical error is equal to zero. But most interpolators hardly satisfy the stability condition with a small $\beta(m)$. Thus, the right hand side will be very large and hence non-informative. On the other hand, let us consider the zero predictor. It is stable, as the rule does not depend on the data. However, it has a very large empirical error. We therefore need a different rule that is stable and has small empirical error, to guarantee generalization.

3.4 Strong convexity

Let us define strong convexity for real valued functions.

Definition 3.2. *The function $\Psi : \mathcal{W} \to \mathbb{R}$ is α-strongly convex w.r.t. to a norm $\|w\|$ if*

$$\Psi(w') \geq \Psi(w) + \langle \nabla\Psi(w), w' - w\rangle + \frac{\alpha}{2}\|w' - w\|^2. \tag{23}$$

Strong convexity essentially means that the function is bounded below by a quadratic function. This property depends on the norm unlike the convexity which can be defined on a vector space. Let us apply the (23) for the optimum point $w_0 := \arg\min_w \Psi(w)$. Since $\nabla\Psi(w_0) = 0$ we get the following for every w:

$$\Psi(w) - \Psi(w_0) \geq \frac{\alpha}{2}\|w - w_0\|^2. \tag{24}$$

The latter means that the difference between the function value at any point and the optimal one is proportional to the distance from the optimal point. Let us now establish the connection of strong convexity and stability. For the dataset $S = \{z_1, \ldots, z_m\}$ we define the regularized ERM (RERM) as follows:

$$\mathsf{RERM}_{\lambda\Psi}(S) = \arg\min_{w\in\mathcal{W}} \left\{\widehat{F}(w) + \lambda\Psi(w)\right\}.$$

Here Ψ is an α-strongly convex function and $\widehat{F}$ is the empirical mean corresponding to the dataset S. We recall the definition of Lipschitz continuity.

Definition 3.3. *The function $f(w,z)$ is G-Lipschitz w.r.t. $\|\cdot\|$ if and only if for every $z \in \mathcal{Z}$ and $w, w' \in \mathcal{W}$*

$$|f(w,z) - f(w',z)| \leq G\|w' - w\|.$$

One may notice that Lipschitz continuity implies some kind of "stability", as it essentially means that small perturbation of the argument will not change the function value drastically. As for strong convexity, the Lipschitzness also depends on the norm. This yields that $\|\nabla_w f(w,z)\|_* \leq G$, where $\|\cdot\|_*$ is the dual norm. We will assume that the norm is the same for both properties.

Proposition 1. *If f is G-Lipschitz and Ψ is α-strongly convex then $\mathsf{RERM}_{\lambda\Psi}(S)$ is stable with a coefficient*

$$\beta(m) \leq \frac{2G^2}{m\lambda\alpha}. \tag{25}$$

Proof. In the following proof we will abbreviate $RERM$ with R. It is straightforward to check that if f is $G-$Lipschitz then so is $\hat{F}$ (with respect to w, conditionally on the z_i). Denote $S = (z_1, ..., z_m)$ the full dataset and $S^{(-m)} = (z_1, ..., z_{m-1})$ the dataset in which z_m is left out. Now define $h_S(w) = \hat{F}_S(w) + \lambda\Psi(w)$, where $\hat{F}_S(w) = \frac{1}{m}\sum_{i=1}^m f(w, z_i)$. Then h_S is $\lambda\alpha$ strongly convex and from equation (24) we have

$$h_S(w) - h_S(\mathsf{R}_{\lambda\Psi}(S)) \geq \lambda\alpha\|w - \mathsf{R}_{\lambda\Psi}(S)\|_2^2. \tag{26}$$

Now, denote $\mathsf{R}_{\lambda\Psi}(S^{(-m)})$ the solution to the ERM problem with the dataset $S^{(-m)}$. Then

$$\begin{aligned} h_S(\mathsf{R}_{\lambda\Psi}(S^{(-m)})) - h_S(\mathsf{R}_{\lambda\Psi}(S)) &= h_{S^{(-m)}}(\mathsf{R}_{\lambda\Psi}(S^{(-m)})) - h_{S^{(-m)}}(\mathsf{R}_{\lambda\Psi}(S)) \\ &+ \frac{1}{m}\left(f(\mathsf{R}_{\lambda\Psi}(S^{(-m)}), z_m) - f(\mathsf{R}_{\lambda\Psi}(S), z_m)\right). \end{aligned} \tag{27}$$

Since $h_{S^{(-m)}}(w)$ is positive and strongly convex,

$$h_{S^{(-m)}}(\mathsf{R}_{\lambda\Psi}(S^{(-m)})) - h_{S^{(-m)}}(\mathsf{R}_{\lambda\Psi}(S)) \geq 0 \tag{28}$$

so that

$$\begin{aligned} h_S(\mathsf{R}_{\lambda\Psi}(S^{(-m)})) - h_S(\mathsf{R}_{\lambda\Psi}(S)) & \\ &\leq \frac{1}{m}\left(f(\mathsf{R}_{\lambda\Psi}(S^{(-m)}), z_m) - f(\mathsf{R}_{\lambda\Psi}(S^{(-m)}), z_m)\right) \\ &\leq G\|\mathsf{R}_{\lambda\Psi}(S^{(-m)}) - \mathsf{R}_{\lambda\Psi}(S)\|_2 \end{aligned} \tag{29}$$

using the Lipschitz continuity of f. Combining this inequality with Eq.(26), we reach

$$\|\mathsf{R}_{\lambda\Psi}(S^{(-m)}) - \mathsf{R}_{\lambda\Psi}(S)\|_2 \leq \frac{G}{\alpha m\lambda}, \tag{30}$$

and

$$f(\mathsf{R}_{\lambda\Psi}(S^{(-m)}), z_m) - f(\mathsf{R}_{\lambda\Psi}(S), z_m) \leq \frac{G^2}{\alpha m\lambda}. \tag{31}$$

In the case of replace-one-out stability, we define $S^{'} = (z_1, ..., z_{m-1}, z')$ and the related estimator $\mathsf{R}_{\lambda\Psi}(S^{'})$. Eq.(27) then becomes

$$\begin{aligned} h_S(\mathsf{R}_{\lambda\Psi}(S^{'})) - h_S(\mathsf{R}_{\lambda\Psi}(S)) &= h_{S'}(\mathsf{R}_{\lambda\Psi}(S^{'})) - h_{S'}(\mathsf{R}_{\lambda\Psi}(S)) \\ &+ \frac{1}{m}\left(f(\mathsf{R}_{\lambda\Psi}(S^{'}), z_m) - f(\mathsf{R}_{\lambda\Psi}(S), z_m)\right) + \frac{1}{m}\left(f(\mathsf{R}_{\lambda\Psi}(S^{'}), z') - f(\mathsf{R}_{\lambda\Psi}(S), z')\right), \end{aligned} \tag{32}$$

leading to

$$\|\mathsf{R}_{\lambda\Psi}(S^{'}) - \mathsf{R}_{\lambda\Psi}(S)\|_2 \leq \frac{2G}{\alpha m\lambda}. \tag{33}$$

□

In the rest of the lecture, without loss of generality, we may assume that the regularization function Ψ is 1-strongly convex. This is easily achieved by tuning the parameter λ accordingly. Theorem 3.1 yields the following

$$\mathbb{E}\left[F\left(\mathsf{RERM}_{\lambda\Psi}(S)\right)\right] \leq \mathbb{E}\left[\widehat{F}\left(\mathsf{RERM}_{\lambda\Psi}(S)\right)\right] + \frac{2G^2}{\lambda m}.$$

Without loss of generality we may assume that Ψ is a positive function. Then using the definition of RERM, for every $w \in \mathcal{W}$ we obtain

$$\begin{aligned}
\mathbb{E}\left[F\left(\mathsf{RERM}_{\lambda\Psi}(S)\right)\right] &\leq \mathbb{E}\left[\widehat{F}\left(\mathsf{RERM}_{\lambda\Psi}(S)\right) + \lambda\Psi(\mathsf{RERM}_{\lambda\Psi}(S))\right] + \frac{2G^2}{\lambda m} \\
&\leq \mathbb{E}\left[\widehat{F}(w) + \lambda\Psi(w)\right] + \frac{2G^2}{\lambda m} \\
&= F(w) + \lambda\Psi(w) + \frac{2G^2}{\lambda m} \\
&\leq \inf_{w\in\mathcal{W}} F(w) + \sqrt{\frac{8G^2 \sup_w \Psi(w)}{m}},
\end{aligned}$$

where the last line is obtained by choosing $\lambda = \sqrt{\frac{2G^2}{\alpha m \sup_w \Psi(w)}}$. In particular, using the constraint $\Psi(w) \leq B$, the last inequality can be rewritten as

$$\mathbb{E}\left[F\left(\mathsf{RERM}_{\lambda\Psi}(S)\right)\right] - F(w^*) \leq O\left(\sqrt{\frac{\sup\{\|\nabla_w f\|_*^2\}B}{m}}\right).$$

The last inequality is obtain by optimizing the right-hand side w.r.t. to the parameter λ. Let us now look back at the Risk minimization problem in the convex case:

$$\min_{w\in\mathcal{W}} \mathbb{E}_{z\sim\mathcal{D}}[f(w,z)] = \min_{w\in\mathcal{W}} \mathbb{E}_{z\sim\mathcal{D}}[\mathsf{loss}(\langle w, \phi(x)\rangle, y)].$$

$\mathcal{W}$ is assumed to be convex. If $\mathsf{loss}(\hat{y}, y)$ is convex in $\hat{y}$, then the problem is convex. For a non-trivial loss e.g. $\mathsf{loss}(h_w(x), y)$ is convex in w **only** when $h_w(x) = \langle w, \phi(x)\rangle$. In this setting the Lipschitz continuity goes as follows. Assume that the loss function $\mathsf{loss}(\mathsf{y}, \mathsf{y}')$ is g-Lipschitz continuous w.r.t. y. Then

$$|f(w,(x,y)) - f(w',(x,y))| \leq g\|\phi(x)\|_* \cdot \|w - w'\|. \tag{34}$$

In particular, the learning problem becomes $G = gR$ Lipschitz continuous, if we assume that $\|\phi(x)\|_* \leq R$, for some $R > 0$. Hence the generalization bound becomes

$$\mathbb{E}\left[F\left(\mathsf{RERM}_{\lambda\Psi}(S)\right)\right] - F(w^*) \leq O\left(\sqrt{\frac{\Psi(w^*)\sup\|\phi(x)\|_*}{m}}\right).$$

In order for the right-hand side to be small we need to have an appropriate sample size. Below we derive the sample complexity for several examples.

- $\Psi(w) = \frac{1}{2}\|w\|_2^2$ is 1-strongly convex w.r.t. $\|w\|_2$ then $m \propto \|w\|_2^2 \cdot \|\phi(x)\|_2^2$.
- $\Psi(w) = \frac{1}{2}w^T Q w$ is 1-strongly convex w.r.t. $\|w\|_Q$ then $m \propto (w^T Q w)(x^T Q^{-1} x)$. Here we choose Q to be small in some direction, then we pay for it in its dual Q^{-1}.
- $\Psi(w) = \frac{1}{2(p-1)}\|w\|_p^2$ is 1-strongly convex w.r.t. $\|w\|_p$, then $m \propto \frac{\|w\|_p^2 \cdot \|x\|_q^2}{p-1}$. Here, we would like the q norm of the data to be small. Thus, we want q to be large. Hence p must be close to 1, which explodes the denominator of the sample complexity.
- $\Psi(w) = \sum_i w[i] \log\left(\frac{w[i]}{1/d}\right)$ is 1-strongly convex w.r.t. $\|w\|_1$. This problem is called the entropic minimizer. Its sample complexity satisfies $m \propto \frac{\|w\|_1^2 \cdot \|x\|_\infty^2}{p-1}$.

We see in this example that in order to have good sample complexity we need to have matching geometries for the data and the parameter.

Online learning The stochastic optimization resembles the online learning problem. The optimizer provides w_i. We give it to the adversary to compute $f(w_i, z_i)$ and then use the value of $f(w_i, z_i)$ to compute w_{i+1}.

The stability in online learning setting plays an important role. Consider the Follow The Leader (FTL) rule. It proposes to choose

$$\hat{w}_m(z_1, \dots, z_{m-1}) = \arg\min_{w \in \mathcal{W}} \sum_{i=1}^{m-1} f(w, z_i) \tag{35}$$

However, this is an unstable rule. A better method called Be the Leader (BTL) suggests the following:

$$\hat{w}_m(z_1, \dots, z_{m-1}) = \arg\min_{w \in \mathcal{W}} \sum_{i=1}^{m} f(w, z_i). \tag{36}$$

This has great convergence properties, but it is not implementable as we assume to have access to $f(w, z_m)$. Instead, we regularize the FTL. The Follow the Regularized Leader (FTRL) goes as

$$\hat{w}_m(z_1, \dots, z_{m-1}) = \arg\min_{w \in \mathcal{W}} \sum_{i=1}^{m} f(w, z_i) + \lambda_t \Psi(w).$$

This algorithm however, does not resemble to the one-pass SGD algorithm (see also equation (22)):

$$w_{t+1} = \arg\min_{w} \langle \nabla f(w_t, z_t), w \rangle + \lambda_t \|w - w_t\|_2^2/2. \tag{37}$$

With the increase of the iteration m, our FTRL needs to minimize a more complex sum-decomposable function. This becomes costly for large m's. For convex objectives we can relax the problem. We minimize the linear approximation of the objective (Linearized FTRL):

$$\hat{w}^{\lambda}_m(z_1, \dots, z_{m-1}) = \arg\min_{w \in \mathcal{W}} \frac{1}{m} \left\langle \sum_{i=1}^{m} \nabla f(w_i, z_i), w \right\rangle + \lambda_t \Psi(w). \tag{38}$$

This problem is simpler than the FTRL, but it is still more complex than the one-pass SGD because of its dependence on the previous gradient evaluations. To fill this gap we introduce the mirror descent method.

3.5 Mirror descent

In this part of the lecture, we will present the mirror descent algorithm. We will get a better understanding of it in the upcoming lectures. Let us define the Bregman divergence. For a given strictly convex, continuously differentiable function Ψ defined on a convex set Ω, the Bregman divergence between two points x, y in Ω is given by

$$D_\Psi(x \mid y) = \Psi(x) - (\Psi(y) + \langle \nabla \Psi(y), x - y \rangle).$$

Intuitively, it corresponds to the distance, at point x, between the function Ψ and its linearization at point y, see Figure 8 for an illustration. In particular, for α-strongly convex functions, it holds that $D_\Psi(x \mid y) \geq \alpha \|x - y\|^2/2$. The mirror descent is then defined as the following iterative scheme:

$$w_{t+1} = \arg\min_{w \in \mathcal{W}} \langle \nabla f(w_t, z_t), w \rangle + \lambda_t D_\Psi(w \mid w_t) \tag{39}$$

$$= \Pi^{\mathcal{W}}_{\Psi} \left(\nabla \Psi^{-1} \left(\nabla \Psi(w_t) - \frac{1}{\lambda_t} \nabla f(w_t, z_t) \right) \right), \tag{40}$$

where $\Pi^{\mathcal{W}}_{\Psi}(w) = \min_{w' \in \mathcal{W}} D_\Psi(w' \mid w)$ is a projection set on $\mathcal{W}$ with respect to the Bregman distance. One may easily verify that if Ψ is a quadratic function, then its Bregman divergence is also quadratic, and we recover the standard gradient descent algorithm, assuming the constraint set $\mathcal{W}$ is the domain of definition of Ψ. Indeed, taking $\Psi(x) = \frac{1}{2}\|x\|_2^2$, we obtain

$$D_\Psi(x|y) = \frac{1}{2}\|x - y\|_2^2 \tag{41}$$

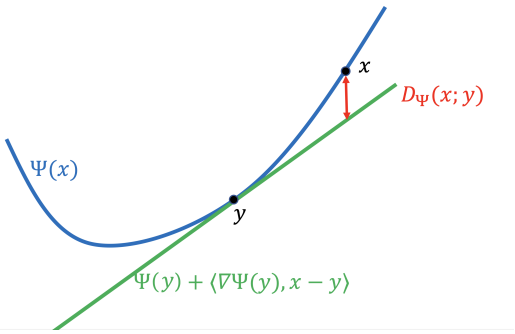

Figure 8: Graphical representation of the Bregman divergence.

so that

$$\arg\min_{w\in\mathcal{W}} \langle \nabla f(w_t, z_t), w\rangle + \lambda_t D_\Psi(w \mid w_t) \tag{42}$$

$$= w_t - \frac{1}{\lambda_t}\nabla f(w^t, z^t) \tag{43}$$

Hence the one-pass SGD is also an instance of the mirror descent. Let us now look at the minimization problem (39). The optimality condition is the following:

$$\begin{aligned} 0 &= \nabla_w\big(\langle \nabla f(w_t, z_t), w\rangle + \lambda_t D_\Psi(w \mid w_t)\big) \\ &= \nabla f(w_t, z_t) + \lambda_t\big(\nabla\Psi(w) - \nabla\Psi(w_t)\big), \end{aligned} \tag{44}$$

which leads to

$$\nabla\Psi(w) = \nabla\Psi(w_t) - \frac{1}{\lambda_t}\nabla f(w_t, z_t).$$

For differentiable, strongly convex functions the gradient is invertible, and we may write the following formula for w_{t+1}

$$w_{t+1} = \Pi_\Psi^{\mathcal{W}}\left((\nabla\Psi)^{-1}\left(\nabla\Psi(w_t) - \frac{1}{\lambda_t}\nabla f(w_t, z_t)\right)\right).$$

Here $\Pi_\Psi^{\mathcal{W}}(w_0)$ is the projection of w_0 on $\mathcal{W}$:

$$\Pi_\Psi^{\mathcal{W}}(w_0) = \arg\min_{w\in\mathcal{W}} D_\Psi(w \mid w_0).$$

See Figure 9 for illustration of the mirror descent algorithm. Suppose that $\mathcal{W}$ is the entire space: $\mathcal{W} = \mathcal{B}$.

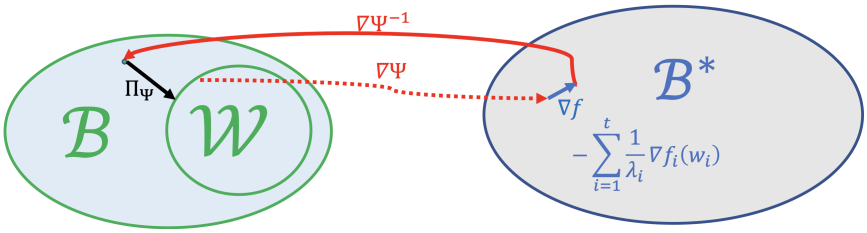

Figure 9: Suppose that the hypothesis class $\mathcal{W}$ is a subset of $\mathcal{B}$, where $\mathcal{B}$ is a Banach space. As we have seen above, the gradients live in the dual space $\mathcal{B}^*$. Then each iteration of the mirror descent consists of four steps. First we take the map $\nabla\Psi : \mathcal{B} \to \mathcal{B}^*$. Then we do a step in $\mathcal{B}^*$ in gradient direction ∇f. At the third step, we take Ψ^{-1}, which maps $\mathcal{B}^*$ to $\mathcal{B}$. The final step is the projection of this point to the hypothesis set $\mathcal{W}$.

Then in the dual space $\mathcal{B}^*$ we get a sum of the gradients, as there is no projection in the primal space. We

take the initial point $w_0 = \arg\min_w \Psi(w)$. This choice is intuitive as one would like to start with a model with lowest complexity. Thus the mirror descent iteration goes as follows:

$$\begin{aligned} w_{t+1} &= \big(\nabla\Psi\big)^{-1}\Big(\nabla\Psi(w_0) - \sum_{i=1}^{t} \frac{1}{\lambda_i}\nabla f(w_i, z_i)\Big) \\ &= \arg\min_w \Big(\sum_{i=1}^{t} \frac{1}{\lambda_i}\langle \nabla f(w_i, z_i), w\rangle + \nabla\Psi(w)\Big), \end{aligned}$$

and we recover the linearized FTRL iteration (38).

3.6 Summary

We presented guarantees for general and stochastic optimization under simple assumptions on the function being optimized and combined these with generalization guarantees to understand how these interact (in particular, is optimization or generalization the "harder" thing to do, in terms of sample complexity?). We studied stability and used it to motivate mirror descent, an abstraction that covers both one-pass SGD and linearized FTRL. Finally, we saw two different ways to see mirror descent. The first is based on the link between the primal and the dual spaces. However, it is less transparent as the link function Ψ does not include information about the reality or model complexity. The other way, is the connection with the linearized FTRL. Here, Ψ acts as a regularizer which naturally describes the link function as a complexity measure.

4 Lecture 4 : Mirror descent and implicit bias of descent algorithms

To recap, we talked last time about learning being a stochastic optimization problem. We are going to use z to get unbiased estimators of the gradient of the population loss. For learning, we consider an objective of the form:

$$f(w,(x,y)) = loss(\langle w, \phi(x)\rangle, y)$$

so that we have convexity with respect to w. This is suitably generic because any data set is realizable in this form: Let ϕ map x to an indicator about its identity, and w selects the appropriate label.

There are two reasons to study convex optimization in a deep learning school: one is because it relates to mirror descent, and the second is that it allows us to talk about the geometry of the optimization problem and therefore the inductive bias.

Recall that if we are exploring the loss landscape with gradient descent, we are implicitly staying close in ℓ_2 norm to the initialization.

4.1 Mirror Descent

We derived mirror descent as regularizing this object with respect to optimizing the first order object. We wrote it in the following form:

$$w_{k+1} = \arg\min_{w\in\mathcal{W}} \langle \nabla f(w_k, z_k), w\rangle + \lambda_k D_\Psi(w||w_k) \tag{45}$$

$$= \Pi_\Psi^{\mathcal{W}}\left(\nabla\Psi^{-1}\left(\nabla\Psi(w_t) - \frac{1}{\lambda_t}\nabla f(w_t, z_t)\right)\right) \tag{46}$$

where

$$D_\Psi(w||w') = \Psi(w) - (\Psi(w') + \langle \nabla\Psi(w'), w - w'\rangle)$$

we replace λ_k with η_k since the regularization is linked to the step size, as was discussed in the previous lecture.

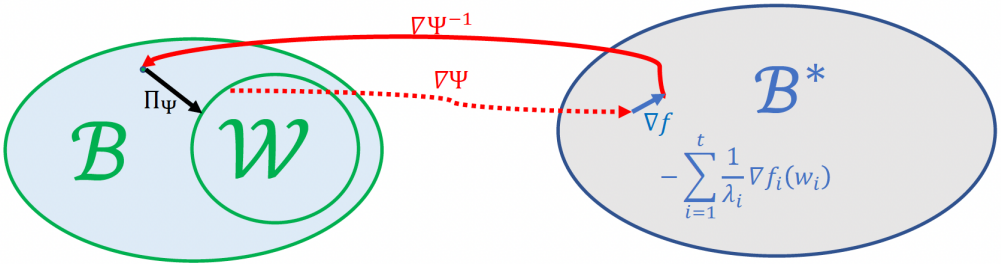

Figure 10: Graphical representation of mirror descent, see Figure 9.

We cannot directly add points in the primal and dual spaces, so we use a link function. To go from primal space to dual space, we use the gradient of the potential, and to go back to the primal space from the dual space,

When the step size is small, we can take a low-order approximation of the Bregman divergence:

$$D_\Psi(w||w_k) \underset{w-w_k \text{small}}{\approx} \langle \nabla\Psi(w_k), w - w_k\rangle - (w-w_k)^T\nabla^2\Psi(w_k)(w-w_k) - \langle \nabla\Psi(w_k), w - w_k\rangle$$

where we have ignored terms not dependent on w, as they do not affect optimizing with respect to w.

We can write the gradient descent problem approximately as:

$$w_{k+1} = \arg\min_{w \in \mathcal{W}} \langle \nabla f(w_k, z_k), w \rangle + \frac{1}{\eta_k}(w - w_k)^T \nabla^2 \Psi(w_k)(w - w_k) \tag{47}$$

$$= w_k - \eta_k \nabla^2 \Psi(w_k)^{-1} \nabla f(w_k, z_k)\,, \quad \text{define}\, \rho(w_k) \coloneqq \nabla^2 \Psi(w_k)^{-1} \tag{48}$$

The latter version of the update rule is known as natural gradient descent, which was first popularized in the context of information geometry by S. Amari (see [1]). It is valid for any potential Ψ (or really for any metric tensor). For intuition, consider that there is a manifold with some local metric ρ, which is the Hessian of Ψ, and we optimize on the manifold. Thus, here we take the view that natural gradient descent is an approximation to mirror descent.

Next, we can take the step size to 0 and write the gradient flow equation:

$$\dot{w} = -\nabla^2 \Psi(w(t))^{-1} \nabla f(w(t), z)\,.$$

As the step size goes to 0, the stochasticity goes away: in any time interval, as the step size goes to 0, we take more steps and so we see more samples, so the stochasticity goes away. Thus, instead of writing the stochastic gradient, we can write the population gradient:

$$\dot{w} = -\nabla^2 \Psi(w(t))^{-1} \nabla F(w(t))\,. \tag{49}$$

This is either very convenient because we do not have to worry about the stochasticity, or it is bothersome because we want to study the stochasticity but cannot.

There are two ways to interpret what is hapening here: when the step size goes to 0, both mirror descent and natural gradient descent (which are the same in the infinitesimal step size limit) converge to the Riemannian gradient flow on the population objective. (Note: this is another way to think about what we are doing with SGD, because as the step size gets smaller and smaller, we *are* approaching optimizing the population loss.) The other way to think about natural gradient descent and mirror descent is: we endow the space with some local geometry and take steps based on that local geometry. Natural gradient descent corresponds to a forward Euler discretization of the natural gradient flow rule (Eqn. 49):

$$\dot{w}(t) = -\nabla^2 \Psi(w(\lfloor t \rfloor_\eta))^{-1} \nabla F(w(\lfloor t \rfloor_\eta)) \tag{50}$$

$$= -\nabla^2 \Psi(w(\lfloor t \rfloor_\eta))^{-1} \nabla f(w(\lfloor t \rfloor_\eta), z_{\lfloor t \rfloor_\eta)} \tag{51}$$

$$w(k\eta + \Delta t) = \nabla \Phi^{-1}(\nabla \Psi(w(k\eta)) + \Delta t \nabla f(w(k\eta), z_k)) \tag{52}$$

$$\Leftrightarrow \nabla \Psi(w(k\eta + \Delta t)) = \nabla \Psi(w(k\eta)) + \Delta t \nabla f(w(k\eta), z_k) \tag{53}$$

Mirror descent represents a slightly more sophisticated discretization of this equation. Suppose we start with any arbitrary metric tensor

$$\dot{w}(t) = -\rho(w(t))^{-1} \nabla F(w(t))\,. \tag{54}$$

What happens if we start with a manifold and seek a complexity measure to optimize over this manifold? In order to determine the complexity measure that underlies the geometry, we require the metric tensor to also be a Hessian map. Unfortunately, most metric tensors (smooth mapping from general vector space to $d \times d$ positive definite matrix) are not Hessian maps. In order to have it be a Hessian, we require that:

$$\frac{\partial \rho_{ij}}{\partial w_k} = \frac{\partial \rho_{ik}}{\partial w_j}\,.$$

This turns out to be almost sufficient as well. Suppose we define:

$$\rho(w) = \mathbb{I} + ww^T\,.$$

The manifold that we get from looking at the squared norm of w appended to w turns out to be the above. It turns out that this is not a Hessian map, as the above symmetry doesn't hold. Thus, starting from a metric tensor, it's quite special to actually be a Hessian map, and that is what allows us to determine the complexity measure that is underlying the geometry. To summarize, NGD is piecewise linear in the primal space, and MD is piecewise linear in the dual space.

4.1.1 Examples of Mirror Descent

Let us now look at how the dynamics look like for some examples of mirror descent. How should we choose the geometry? We know that:

$$\mathbb{E}_{S\sim D^m}\left[F(\bar{w}_k)\right] \leq F(w^\star) + O\left(\sqrt{\frac{\Psi(w^\star)\,\sup||\nabla f||_\star}{k}}\right)$$

which holds as long as Ψ is 1-strongly convex with respect to our choice of norm. Here, $w^\star$ is no longer the best in the norm bounded class, since we do not wish to limit the norm explicitly. The guarantee above means that we can compete with any $w^\star$, we just have to pay for its complexity in the second term.

Question When the metric tensor doesn't correspond to a Hessian, does that mean we cannot come up with a global complexity measure? **Answer** The answer is a combination of "no" and "I don't know."

Example 1: Euclidean potential

$$\Psi(w) = \frac{1}{2}\,||w||_2^2 \quad \text{strongly convex wrt} \quad ||w||_2\,.$$

Here, $k \propto ||w^\star||_2^2\,||\phi(x)||_2^2$. The metric tensor is just the identity, and the dynamics are the standard GD dynamics

Example 2: Mahalanobis potential

$$\Psi(w) = \frac{1}{2}w^TQw \quad \text{strongly convex wrt}\,||w||_Q = \sqrt{w^TQw}$$

Here, $k \propto (w^{\star T}Qw^\star)\phi(x)^TQ^{-1}\phi(x)$. The dynamics correspond to preconditioned gradient descent:

$$\dot{w} = -Q^{-1}\nabla F(w)\,.$$

Example 3: Entropic potential

$$\Psi(w) = \sum_i w[i]\log\frac{w[i]}{i/d}$$

here, $k \propto ||w||_1^2\log d\,||\phi(x)||_\infty^2$. Here, the Hessian will be diagonal with $1/w$ on each component, and so the dynamics will look like:

$$\dot{w} = -\nabla^2\Psi(w)^{-1}\nabla F(w) = -\text{diag}(w)\nabla F(w) = -w_i\partial_i F(w)$$

The local geometry is not constant, nor is it uniformly penalized in the same directions everywhere. It penalizes changing coordinates that are already small.

4.1.2 Smoothness and Batching

Definition 4.1. *If $F(w') \leq F(w) + \langle\nabla F(w), w'-w\rangle + \frac{H}{2}\,||w'-w||^2$, we say that the convex function F is H-smooth.*

For a smooth function, we can get a better convergence rate, which depends on $1/k$ rather than $1/\sqrt{k}$:

$$\mathbb{E}\left[F(\bar{w}_k)\right] \leq F(w^\star) + O\left(\frac{H\Psi^\star(w)}{k} + \sqrt{\frac{\sigma^2\Psi^\star(w)}{k}}\right)\text{, where } \sigma \text{ s.t. } \mathbb{E}\left[||\nabla f(w,z) - \nabla F(z)||^2\right] \leq \sigma^2 \qquad (55)$$

The first term above is the optimization term that only depends on the discretization (how well does linearization match the true function), and the second term is the stochasticity term, which depends on the distance between the stochastic approximation and the population objective. Accordingly, it makes sense to

take more samples than discretization steps. We can do this via using batches. In particular, if instead we use:

$$\frac{1}{b}\sum_{i=1}^{b}\nabla f(w,z_i)\,,$$

then the variance scales with $1/b$, giving us:

$$\mathbb{E}\left[F(\bar{w}_k)\right] \leq F(w^\star) + O\left(\frac{H\Psi^\star(w)}{k} + \sqrt{\frac{\sigma^2\Psi^\star(w)}{bk}}\right)\,. \tag{56}$$

We can see here is that optimization is easier than the statistical aspect. It never really helps us to take more steps than the number of samples. In the non-convex case, this is substantially different: in particular, the optimization might be harder than the statistical part, and so it might be worth viewing the samples multiple times.

Remark It is sufficient to look just at variance of the gradients at $w^\star$. This is important because if the problem is realizable, then at the optimum, $w^\star$ is correct for any z, and therefore the gradients are 0. So if $f \geq 0$,

$$\mathbb{E}\left[F(\bar{w}_k)\right] \leq F(w^\star) + O\left(\frac{H\Psi^\star(w)}{k} + \sqrt{\frac{HF(w^\star)\Psi^\star(w)}{bk}}\right)\,.$$

One more observation: this analysis relies not only on the choice of potential function but also on the choice of norm: the variance and smoothness depend on the norm, but the algorithm itself doesn't rely on the norm. Recent analyses alleviate this by going through relative smoothness [3, 17].

Definition 4.2. *A function F is relatively smooth to Ψ up to smoothness parameter H if:*

$$\nabla^2 F(w) \preceq H\nabla^2\Psi(w)\,.$$

The guarantee in Eqn. 55 holds when H is the relative smoothness parameter.

4.2 General Steepest Descent

So far, we have talked about mirror descent and have related it to Riemannian gradient flow. We can also think more generally about steepest descent methods that are not related to a metric.

$$w_{k+1} = \arg\min\langle\nabla f(w_k,z_k),w\rangle + \frac{1}{\eta}\delta(w,w')\,.$$

We can take $\delta(w,w') = ||w-w'||_1$, for example, and though it appears to correspond to using ℓ_1 geometry, it actually will correspond to coordinate descent. Using ℓ_∞ will take a step corresponding to the sign of the gradient.

4.3 Implicit bias of descent methods

So far, we have discussed the convex setting, and we related optimization guarantees and generalization guarantees. We get generalization guarantees with respect to $w^\star$ considering only the dynamics of optimization. We studied the connection between the geometry of searching the space and the corresponding complexity. However, there are some limitations to what we have seen so far. First of all, this is only the convex case. Second of all, this isn't what we're seeing in deep learning. Here, we are not driving training error to 0. The generalization seems to rely on the fact that we are using stochasticity. That is different from what we see in the examples from the previous lectures where we train with full batch, multi-pass gradient methods but still get good generalization.

Let us compare two approaches. First, suppose we actually select:

$$w_\lambda = \arg\min \hat{F}_k(w) + \lambda\Psi(w)\,.$$

And secondly, consider $\bar{w}_k$, the solution achieved after k iterations of mirror descent. In either of these cases, we get the same generalization guarantee, but for different solutions. In the Lipschitz case with optimally chosen λ, the distance (suboptimality, really) between w_λ and $w^\star$ is $1/\sqrt{k}$. The distance between $w^\star$ and $\bar{w}_k$ is also $1/\sqrt{k}$, but the distance between $w_\lambda, \bar{w}$ is also $1/\sqrt{k}$. We know this because we know that we can view one-pass of this as optimizing the training objective, which in k steps gets suboptimality $1/\sqrt{k}$. Thus, we are not saying that implicit regularization gets us to the same solution as explicit regularization, but rather we are saying that the generalization guarantees hold. If we keep repeating passes, we might get to minimizer of the training error, but it's unclear if this is beneficial.

4.3.1 Deriving Implicit Regularization for Gradient Descent

We will analyze gradient descent on the unregularized training objective:

$$\hat{F}(w) = \frac{1}{m}\sum_{i=1}^{m} loss(\langle w, x_i\rangle, y_i)\,.$$

We've dropped ϕ for ease of notation, but we should think of x as being some feature map. Let $w \in \mathbb{R}^d, d >> m$. Let us use gradient descent:

$$w_{k+1} = w_k - \eta\nabla\hat{F}(w_k)\,; w_0 = 0\,.$$

We know from previous discussion that this will result in implicit ℓ_2 regularization, but we will formally derive this. Additionally, this will later allow us to understand what happens when using mirror descent.

For the first step, we are going to argue that the iterates w_k all lie in a linear manifold given by the span of the data, i.e.:

$$w_k \in \mathcal{M} = \text{span}(X) = \left\{ w = X^T s \middle| s \in \mathbb{R}^m \right\}$$

This is because each update involves adding a scaled gradient, and the gradient is dependent on a residual. This already tells us a lot about what we will get: even though we have d parameters, we only will ever be in an m-dimensional subspace. We will also assume that we get to a global optimum. This is not always easy to show but are going to assume it. Since we are in the overparametrized setting, we know that $Xw = y$, where w is the predictor to which we finally converge. We claim that the point we get is exactly the minimizer of the following optimization problem:

$$\min \frac{1}{2}\,||x||_2^2 \quad \text{ such that } Xw = y\,. \tag{57}$$

We have the optimization that we are presumably optimizing: that is, minimizing the empirical error, and we have the optimization problem above. In general, these arguments are going to proceed by claiming that solving the first optimization problem actually implicitly solves the second optimization problem. To show this, we use the method of Karush-Kuhn-Tucker (KKT) conditions for optimality of a solution to a constrained optimization problem. The optimum for a constrained optimization problem is uniquely characterized by its KKT conditions. Let us look at the KKT conditions. We introduce dual variables ν for the constraints, giving us that the Lagrangian is:

$$L(w, \nu) = \frac{1}{2}\,||w||_2^2 + \nu^T(y - Xw)\,.$$

The KKT conditions are:

- **Stationarity** $0 = \nabla_w L = w - X^T\nu$
- **Primal feasibility** $y = Xw\,.$

We call ν the dual variables, or the variables in the dual space. We observe that the stationarity condition shows exactly what we claimed earlier, that w_k are in the span of the data, and the primal feasibility condition finds the interpolator. Since a point that satisfies these two conditions is optimal for the optimization problem in Eqn. 57, and since gradient descent that arrives at a 0 training error predictor satisfies these two conditions, gradient descent finds an optimal solution to the problem in Eqn. 57, i.e., a minimum Euclidean norm solution.

4.3.2 Similar Argument for Mirror Descent

With this argument in mind, we can apply a similar procedure to analyze the output of mirror descent. We are going to show that with mirror descent, we get to a solution that is good with respect to the stated potential. To show this, we will use essentially the same proof. The optimization problem is still the same as before: we are minimizing the training objective, but the optimization algorithm will differ. Now, we optimize $\hat{F}$ using mirror descent with respect to some potential function Ψ. Let us reproduce the iterates:

$$w_{k+1} = \arg\min_{w \in \mathcal{W}} \langle \nabla f(w_t, z_t), w \rangle + \lambda_t D_\Psi(w||w_t)$$

In mirror descent, we accumulate the gradients in the dual space.

$$w_{k+1} = \nabla\Psi^{-1}\left(\nabla\Psi(w_0) - \sum_{i=1}^{k} \nabla\hat{F}(w_k)\right)$$

That is, the $k+1$ iterate is the mapping back into the primal space of the accumulation of the gradients in the dual space. As before, we can see that the gradients lie in the span of the data (in the dual space). Thus, again, $w_k \in \mathcal{M} = \{\nabla\Psi^{-1}(\nabla\Psi(w_0) + X^T s) \,\forall\, x \in \mathbb{R}^m\}$. This is not a flat manifold (zero curvature) in the primal space, but it is in the dual space. To see what point we converge to, we use the fact that in the end, we converge to a global optimum. This imposes m linear constraints, which when intersected with an m-dimensional manifold gives us a unique point. To determine which unique point that is, we write an optimization problem whose KKT conditions match the two sets of constraints (global optimality and lying in the manifold):

$$\min D_\Psi(w||w_0) \quad \text{such that } Xw = y\,.$$

Then the Lagrangian is:

$$L(w, \nu) = D_\Psi(w||w_0) + \nu(y - X\mu)$$

To see this, we write the KKT conditions:

- **Stationarity** $0 = \nabla\Psi(w) - \Psi(w_0) - X^t\nu$
- **Primal Feasibility** Xw = y

This is exactly what was specified earlier.

4.3.3 General Method

We started from the optimization problem of minimizing the training objective and then saw trajectory stays within a manifold. The second set of constraints we imposed came from the global optimality of the solution reached. We then matched these two sets of constraints to KKT conditions for some other optimization problem.

If w_0 were the minimizer of the potential function, then the optimization problem becomes minimizing the potential function subject to $Xw = y$.

5 Lecture 5: Implicit bias with linear functionals and the square loss

The goal of this lecture is to provide analytical evidence to the lazy [8] and rich regimes in learning problems with gradient descent. We will consider a simple model for which the dynamics of gradient descent may be solved exactly and we will study these trajectories as a function of the magnitude of the initialization and model architectures, leading to various implicit biases.

5.1 Setting

We consider models parametrized by a weight vector $\mathbf{w} \in \mathbb{R}^p$ acting on an input space $\mathcal{X}$ and denote $F(\mathbf{w}) \in \{f : \mathcal{X} \to \mathbb{R}\}$ the predictor implemented by $\mathbf{w}$, such that $F(\mathbf{w})(\mathbf{x}) = f(\mathbf{w}, \mathbf{x})$. We will focus on models linear in $\mathbf{x}$ represented by a linear functional in the dual space of $\mathcal{X}$, denoted $\mathcal{X}^*$ represented by a vector $\boldsymbol{\beta}_{\mathbf{w}}$ such that $f(\mathbf{w}, \mathbf{x}) = \langle \boldsymbol{\beta}_{\mathbf{w}}, \mathbf{x} \rangle$, i.e. $\boldsymbol{\beta}_{\mathbf{w}}$ is some transform, potentially non-linear of $\mathbf{w}$. We consider the supervised learning problem with n sample pairs $(\mathbf{x}_i, y_i)$ where $y_i \in \mathbb{R}$ is a response vector. The parameters are learned by minimizing the empirical loss

$$\frac{1}{n} \sum_{i=1}^{n} \text{loss}(f(\mathbf{w}, \mathbf{x}_i), y_i) \tag{58}$$

with the corresponding population loss

$$\mathbb{E}_{\mathbf{x},y} \left[\text{loss}(f(\mathbf{w}, y), \mathbf{x}) \right]. \tag{59}$$

We assume the model is homogeneous of order D, i.e., for any constant $c > 0$ $F(c\mathbf{w}) = C^D h_{\mathbf{w}}$. The order D is related to the depth of networks with homogeneous activations (e.g. a linear or Relu). A linear model is homogeneous of order 1, a factorization model of order 2. Our goal is three-fold :

- first, we want to characterize the implicit bias of gradient descent in this setup. In regards to the previous lectures, is optimzing in the parameters space using gradient descent equivalent to optimizing in the functional space w.r.t. some metric tensor and potential ? What is this potential
- secondly is it equivalent to explicit regularization ? For instance, we saw that GD on a least-squares problem converges towards the minimum ℓ_2 norm solution, equivalent to explicitly penalizing the norm. Same here ?
- finally, we would like to study the transition between kernel (a.k.a. lazy) regime and rich regime. What parameters govern this behaviour ? We have seen in the previous lectures that in many cases we obtain implicit biases that cannot be reached with kernel methods (sparsity, nuclear norm, ...). Can we get a more precise picture on simple models ?

5.2 Reminder on the kernel regime

Consider the function computed by the model $f(\mathbf{w}, \mathbf{x})$. We can take its first order approximation around the initalization of GD $\mathbf{w}_0$.

$$f(\mathbf{w}, \mathbf{x}) = f(\mathbf{w}_0, \mathbf{x}) + \langle \mathbf{w} - \mathbf{w}_0, \nabla_{\mathbf{w}} f(\mathbf{w}_0, \mathbf{x}) \rangle + \mathcal{O}\left(\|\mathbf{w} - \mathbf{w}_0\|_2^2 \right) \tag{60}$$

In what follows, we will sometimes write $\phi_0(\mathbf{x}) = \nabla_{\mathbf{w}} f(\mathbf{w}_0, \mathbf{x})$. In certain regimes, this linear approximation is always valid across training and the model behaves as an affine model $f(\mathbf{w}, \mathbf{x}) = f_0(\mathbf{x}) + \langle \mathbf{w}, \phi_0(\mathbf{x}) \rangle$ with feature map $\nabla_{\mathbf{w}} f(\mathbf{w}_0, \mathbf{x})$ corresponding to the tangent kernel $K_0(\mathbf{x}, \mathbf{x}') = \langle \nabla_{\mathbf{w}} f(\mathbf{w}_0, \mathbf{x}), \nabla_{\mathbf{w}} f(\mathbf{w}_0, \mathbf{x}') \rangle$. GD then learns the corresponding minimum RKHS distance to the initialization $F(\mathbf{w}_0)$ solution, i.e. $\arg\min_h \|h - F(\mathbf{w}_0)\|_{K_0}$ s.t. $h(X) = \mathbf{y}$. We can avoid the $F(\mathbf{w}_0)$ by choosing a familiy of functions verifying $F(\mathbf{w}_0) = 0$ (unbiased initialization from [8]). Then are we just studying an uninformative regime or can we really replace

neural nets with linear models ? In what case do we get this kernel regime ?

Initially, was linked to the width, taking width to infinity leads to a Gaussian process which is a kernel regime. See e.g. [13, 9]. Closer to what we want to do is [8]. Regardless of the width, can always reach the kernel regime when the scale of the initialization goes to infinity. In what follows, we will mainly consider gradient flow, i.e.

$$\frac{d\mathbf{w}}{dt} = -\nabla_{\mathbf{w}} \hat{L}(\mathbf{w}) \tag{61}$$

Initialize at different scales, i.e. $\mathbf{w}(0) = \alpha \mathbf{w}_0$ where $\alpha > 0$ and $\mathbf{w}_0$ can be any $\mathbf{w}_0$ which can be random, and such that $F(\mathbf{w}_0) = 0$ (i.e. $\mathbf{w}_0$ maps to a null function in function space). For any α, we will write the dynamics

$$\frac{d\mathbf{w}_\alpha}{dt} = -\nabla_{\mathbf{w}} \hat{L}(\mathbf{w}_\alpha) \tag{62}$$

The result from [8] then states that when α goes to infinity, after a appropriate rescaling of time, the entire trajectory converges to the kernel one :

$$\lim_{\alpha \to \infty} \sup_t \|\mathbf{w}_\alpha(\frac{1}{\alpha^{D-1}} t) - \mathbf{w}_K(t)\|_\infty = 0 \tag{63}$$

where $\mathbf{w}_K$ represents the vector obtained by running gradient descent on the tangent kernel model : $\dot{\mathbf{w}}(K) = -\nabla_{\mathbf{w}} \hat{L}(\langle \mathbf{w}, \phi_0(\mathbf{x}) \rangle)$.

5.3 A simple model : 2-layer linear diagonal network

We would like a simple model going beyond the linear case (for which we understand the phenomenology), that can still be studied analytically and where the implicit bias is interesting. We are going to look at element-wise squaring of parameters, i.e.

$$f(\mathbf{w}, \mathbf{x}) = \langle \mathbf{w}^2, \mathbf{x} \rangle \quad \beta = F(\mathbf{w}^2) \tag{64}$$

where for a given vector, ^{2} denotes the elementwise squaring. This is equivalent to a depth 2 diagonal linear network. Here we cannot get all linear functions this way because the coefficients cannot be negative. In order to allow negative coefficients, we used $2d$ parameters $\mathbf{w} = \begin{bmatrix} \mathbf{w}_+ \\ \mathbf{w}_- \end{bmatrix}$ and $\boldsymbol{\beta}_{\mathbf{w}} = \mathbf{w}_+^2 - \mathbf{w}_-^2$. This is equivalent to a depth-2 diagonal linear network with $2d$ parameters on the first layer $f(\mathbf{w}, \mathbf{x}) = \mathbf{w}^\top \mathrm{diag}(\mathbf{w}) \begin{bmatrix} \mathbf{x} \\ -\mathbf{x} \end{bmatrix}$, as in the following figure

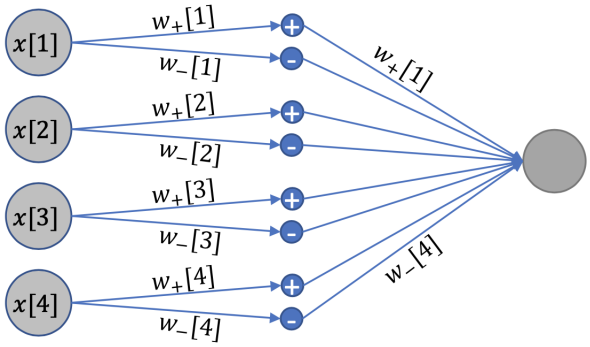

Figure 11: Depth-2 diagonal linear net with replicated an signed units in the first layer

Negative functions can thus be represented and we can also choose the initialization such that $F(\mathbf{w}_0) = 0$. If $\mathbf{w}_0 = 1_d$, then at initialization, $\boldsymbol{\beta}_0 = 0$ whatever the scale of initialization α, where $\mathbf{w}_{+,\alpha}(0) = \mathbf{w}_{-,\alpha}(0) = \alpha \mathbf{w}_0$. What's the implicit bias of doing gradient descent on this model, when considering the square loss ?

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \tag{65}$$

5.3.1 Analytical study of GD in parameter space

Applying the chain rule gives, denoting $\mathbf{X} \in \mathbb{R}^{n \times d}$ the design matrix

$$\dot{\mathbf{w}}_+ = -\frac{d\boldsymbol{\beta}}{d\mathbf{w}^+}\frac{dL(\boldsymbol{\beta})}{d\boldsymbol{\beta}} = -2\mathbf{X}^\top \mathbf{r}(t) \odot \mathbf{w}_+(t) \tag{66}$$

$$\dot{\mathbf{w}}_- = -\frac{d\boldsymbol{\beta}}{d\mathbf{w}^-}\frac{dL(\boldsymbol{\beta})}{d\boldsymbol{\beta}} = 2\mathbf{X}^\top \mathbf{r}(t) \odot \mathbf{w}_-(t) \tag{67}$$

$$\tag{68}$$

where we defined the residual $\mathbf{r}(t) = \mathbf{X}\boldsymbol{\beta}(t) - \mathbf{y}$ and $\odot$ denotes the elementwise product.

Consider that the residuals are known, then these are differential equations that can be solved. Integrating yields

$$\mathbf{w}_+(t) = \mathbf{w}_+(0) \odot \exp\{-2\mathbf{X}^\top \int_0^t \mathbf{r}(\tau)d\tau\} \tag{69}$$

$$\mathbf{w}_-(t) = \mathbf{w}_-(0) \odot \exp\{2\mathbf{X}^\top \int_0^t \mathbf{r}(\tau)d\tau\} \tag{70}$$

where $\mathbf{r}(t) = \mathbf{X}^\top\left(\mathbf{w}_+^2 - \mathbf{w}_-^2\right) - \mathbf{y}$. Although this is just a rewriting as integral equations, we can get important information from this. Letting $\mathbf{S}(t) = \int_0^t \mathbf{r}(\tau)d\tau$:

$$\boldsymbol{\beta}(t) = \mathbf{w}_+^2(0) \odot \exp\left(-4\mathbf{X}^\top \mathbf{S}(\mathbf{t})\right) - \mathbf{w}_-^2(0) \odot \exp\left(4\mathbf{X}^\top \mathbf{S}(\mathbf{t})\right) \tag{71}$$

$$= \alpha^2 1_d \odot \left(\exp\left(-4\mathbf{X}^\top \mathbf{S}(\mathbf{t})\right) - \exp\left(4\mathbf{X}^\top \mathbf{S}(\mathbf{t})\right)\right) \tag{72}$$

$$= 2\alpha^2 1_d \odot \sinh\left(-4\mathbf{X}^\top \mathbf{S}(\mathbf{t})\right) \tag{73}$$

where sinh is the hyperbolic sine. We are interested in the regime where $n << d$ i.e. number of samples much lower than dimensionality d, so they are many solutions to the problem $\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$. However, Eq.(71)-(72) shows that we have reduced the set of solutions to a lower dimensional manifold of dimension n. This is similar to the study of GD on linear regression, i.e. the predictor is always spanned by the data. Here we observe the same thing on a non-linear model. This manifold is defined by

$$\mathcal{M} = \left\{\boldsymbol{\beta} = \alpha^2 1_d \odot \left(\exp\left(-4\mathbf{X}^\top \mathbf{s}\right) - \exp\left(4\mathbf{X}^\top \mathbf{s}\right)\right) | \mathbf{s} \in \mathbb{R}^n\right\} \tag{74}$$

$$= \left\{\boldsymbol{\beta} = 2\alpha^2 1_d \odot \sinh\left(-4\mathbf{X}^\top \mathbf{s}\right) | \mathbf{s} \in \mathbb{R}^n\right\} \tag{75}$$

where we have n additional constraints defined by the equation $\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$ leading to a unique solution. Note that we are not studying the convergence of gradient flow, we assume it converges. Let's write an equivalent optimization problem giving the same set of solutions. We want to find a function $Q_\alpha(\boldsymbol{\beta})$ that is implictly minimized by the GF dynamics such that

$$\boldsymbol{\beta}^* \in \min_{\boldsymbol{\beta}} Q_\alpha(\boldsymbol{\beta}) \tag{76}$$

$$\text{s.t.} \quad \mathbf{X}\boldsymbol{\beta} = \mathbf{y} \tag{77}$$

The Lagrangian formulation for this problem reads

$$\min_{\boldsymbol{\beta}} \max_{\boldsymbol{\nu}} Q_\alpha(\boldsymbol{\beta}) + \boldsymbol{\nu}^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \tag{78}$$

The KKT conditions then read

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{y} \tag{79}$$

$$\nabla_{\boldsymbol{\beta}} Q_\alpha(\boldsymbol{\beta}) = \mathbf{X}^\top \boldsymbol{\nu} \tag{80}$$

Using Eq.(73), we may write

$$\sinh^{-1}\left(\frac{1}{2\alpha^2}\boldsymbol{\beta}\right) = -4\mathbf{X}^\top \mathbf{s} \tag{81}$$

where the inverse hyperbolic sine is applied elementwise. Matching this with the optimality condition then gives

$$\boldsymbol{\nu} = -4\int_0^\infty \mathbf{r}_\alpha(t)\,(= -4\mathbf{s}) \tag{82}$$

$$\nabla_{\boldsymbol{\beta}} Q_\alpha(\boldsymbol{\beta}) = \sinh^{-1}\left(\frac{1}{2\alpha^2}\boldsymbol{\beta}\right) \tag{83}$$

Integrating this expression, remembering that the gradient of an element-wise function is applied element-wise, yields

$$Q_\alpha(\boldsymbol{\beta}) = \sum_{i=1}^{d} \alpha^2 q(\frac{\beta_i}{\alpha^2}) \tag{84}$$

where $q(z) = \int_0^z \sinh^{-1}(\frac{t}{2})dt = 2 - \sqrt{4+z^2} + z\sinh^{-1}(\frac{z}{2})$.
We can now study this function for different scalings of α. Plotting q gives the following figure :

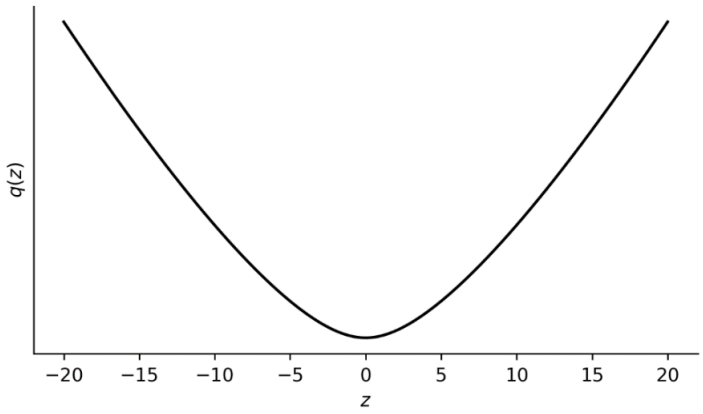

Figure 12: $q(z) = \int_0^z \sinh^{-1}(\frac{t}{2})dt = 2 - \sqrt{4+z^2} + z\sinh^{-1}(\frac{z}{2})$

For $\alpha \to \infty$, we may look at q around zero, a second order Taylor expansion shows that q is quadratic around zero. Thus for large α, the regularization is effectively quadratic on each coordinate. In this model, we can show that the tangent kernel at initialization is the linear kernel $K_0(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$ thus the solution converges to the minimum ℓ_2 norm solution

$$\boldsymbol{\beta}_\alpha(\infty) \xrightarrow{\alpha\to\infty} \hat{\boldsymbol{\beta}}_{L_2} = \underset{\mathbf{X}\boldsymbol{\beta}=\mathbf{y}}{\arg\min} \|\boldsymbol{\beta}\|_2 \quad \text{Kernel regime} \tag{85}$$

For small values of α however, q becomes close to a ℓ_1 norm and we obtain an effective ℓ_1, sparsity inducing implicit regularization which does not correspond to a kernel regime, but a rich regime.

$$\boldsymbol{\beta}_\alpha(\infty) \xrightarrow{\alpha\to 0} \hat{\boldsymbol{\beta}}_{L_1} = \underset{\mathbf{X}\boldsymbol{\beta}=\mathbf{y}}{\arg\min} \|\boldsymbol{\beta}\|_1 \quad \text{Rich regime} \tag{86}$$

We thus have a transition from kernel inductive bias to a sparsity inducing inductive bias. The rich learning regime corresponds to the ℓ_1 regime, which corresponds to learning : learning feature corresponds to selecting a small number of features among an infinite amount of features. A more precise characterization of this transition can be found in Theorem 2 from [33], which we reproduce here.

Theorem 5.1 (Theorem 2 from [33])**.** *For $0 < \epsilon < d$, under the above setting,*

$$\alpha \leqslant \min\left\{\left(2(1+\epsilon)\|\boldsymbol{\beta}^*_{\ell_1}\|_1\right)^{-\frac{2+\epsilon}{2\epsilon}}, \exp(-d/(\epsilon)\|\boldsymbol{\beta}^*_{\ell_1}\|_1)\right\} \implies \|\boldsymbol{\beta}^\infty_{\alpha,1}\|_1 \leqslant (1+\epsilon)\|\boldsymbol{\beta}^*_{\ell_1}\|_1$$

$$\alpha \geqslant \sqrt{2(1+\epsilon)(1+2/\epsilon)\|\boldsymbol{\beta}^*_{\ell_2}\|_2} \implies \|\boldsymbol{\beta}^\infty_{\alpha,1}\|_2^2 \leqslant (1+\epsilon)\|\boldsymbol{\beta}^*_{\ell_1}\|_2^2$$

The sparsity inducing bias is fundamentally different from the kernel regime. Consider the following example of sparse regression

$$y_i = \langle \boldsymbol{\beta}^*, \mathbf{x}i \rangle + \gamma \quad \text{where} \quad \gamma \sim \mathcal{N}(0, 0.01) \tag{87}$$

where $d = 1000; \|\boldsymbol{\beta}^*\|_0 = 5$ and we have $n = 100$ samples. A kernel method cannot solve this problem : we can see this on the following figure For large α, we are in the kernel regime and the excess ℓ_2 norm is small,

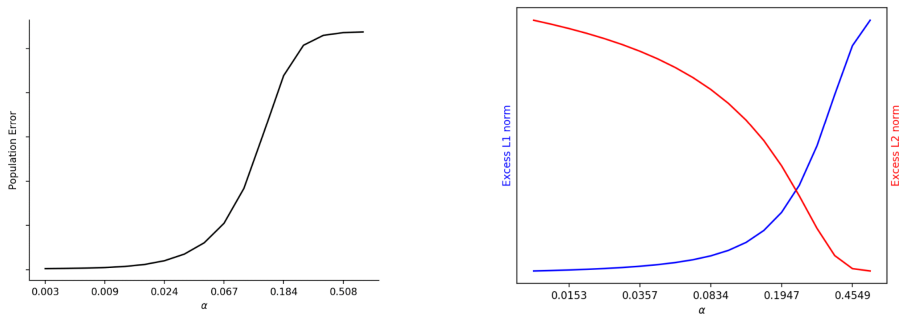

Figure 13: Generalization error of kernel and rich regime on a sparse regression problem with $n << d$

but the population error is large. For small alpha however, both the excess ℓ_1 norm and the population error are small.
Getting to the ℓ_1 regime is actually quite difficult. See this from Theorem 2 : α has to be exponentially small. What's the sample complexity as a function of α, or conversely, how small does α has to be to reach good performance, let's say $L(\boldsymbol{\beta}_\alpha(\infty)) \leqslant 0.025$ Thus, for very low number of samples, doing the exact ℓ_1 is

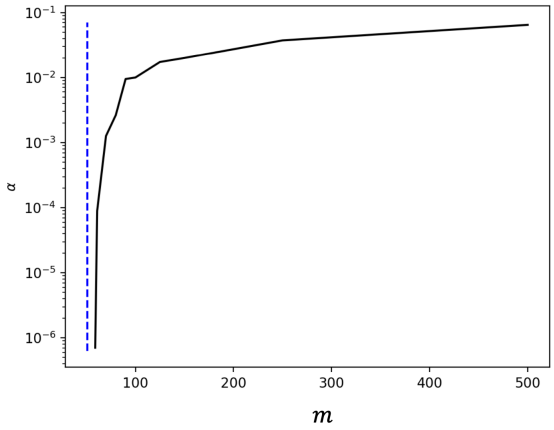

Figure 14: Threshold value of alpha varies with the number of samples

impossible (vertical asymptote), but for a reasonable amount of samples, we can get good performance with an approximate ℓ_1. This concludes the link between the scaling of initialization α to the kernel and rich regimes.

5.3.2 Studying the dynamics in function space

Whats do the dynamics look like in function space ? Recall that the function space is parametrized by $\boldsymbol{\beta}$, thus we want to write the solution directly on $\boldsymbol{\beta}$ instead of $\mathbf{w}$, and $\boldsymbol{\beta} = F(\mathbf{w})$.

$$\dot{\boldsymbol{\beta}} = \frac{d\boldsymbol{\beta}}{d\mathbf{w}}\dot{\mathbf{w}} = \nabla F(\mathbf{w}(t))^\top \dot{\mathbf{w}} \tag{88}$$

$$= \nabla F(\mathbf{w}(t))^\top \left(-\nabla_{\mathbf{w}} L(\mathbf{w}(t))\right) \tag{89}$$

where $\nabla F(\mathbf{w}(t)) \in \mathbb{R}^{p\times p}$ is the Jacobian of F. The chain rule then gives

$$\nabla_{\mathbf{w}} L(\mathbf{w}(t)) = \nabla_{\mathbf{w}} L(\boldsymbol{\beta}(\mathbf{w}(t))) \tag{90}$$
$$= \nabla F(\mathbf{w}(t)) \nabla L(\boldsymbol{\beta}) \tag{91}$$

giving the dynamics

$$\dot{\boldsymbol{\beta}} = -\nabla F(\mathbf{w}(t))^\top \nabla F(\mathbf{w}(t)) \nabla L(\boldsymbol{\beta}) \tag{92}$$

which corresponds to what the previously discussed Riemanian gradient flow, with metric tensor $\rho = \left(\nabla F(\mathbf{w}(t))^\top \nabla F(\mathbf{w}(t))\right)^{-1}$, i.e.

$$\dot{\boldsymbol{\beta}} = -\rho^{-1} \nabla L(\boldsymbol{\beta}). \tag{93}$$

Thus choosing a certain parametrization F induces a geometry in the search done by the gradient descent, governed by the metric tensor ρ. But ρ is a function of $\mathbf{w}(t)$: in the case of the model discussed previously,

$$\nabla F(\mathbf{w}(t)) = \begin{bmatrix} \mathrm{diag}(\mathbf{w}_+) \\ \mathrm{diag}(\mathbf{w}_-) \end{bmatrix} \in \mathbb{R}^{2d\times d} \tag{94}$$

$$\rho(\mathbf{w}(t)) = \mathrm{diag}\left(\mathbf{w}_+^2 + \mathbf{w}_-^2\right)^{-1} \tag{95}$$

We would now like to rewrite this entirely as functions of $\boldsymbol{\beta}$, i.e. write

$$\dot{\boldsymbol{\beta}} = -\rho(\boldsymbol{\beta}) \nabla L(\boldsymbol{\beta}) \tag{96}$$

This is problem dependent, and possible here. Recall the dynamics on $\mathbf{w}_+$ and $\mathbf{w}_-$ from the previous section, we have

$$\frac{d}{dt}\left(\mathbf{w}_+ \odot \mathbf{w}_-\right) = -2\mathbf{X}^\top \mathbf{r}(t) \left(\mathbf{w}_+ \odot \mathbf{w}_- - \mathbf{w}_- \odot \mathbf{w}_+\right) = 0 \tag{97}$$

We can thus evaluate this quantity at $t = 0$ which gives

$$\forall t,\ \mathbf{w}_+ \odot \mathbf{w}_- = \alpha^2 1_d \tag{98}$$

Note that this also appears immediately by considering Eq.(69)-(70) along with the fact that $\mathbf{w}_+(0) = \mathbf{w}_-(0) = \alpha 1_d$. Furthermore, by definition of $\boldsymbol{\beta}(t)$

$$\boldsymbol{\beta}(t) = \mathbf{w}_+^2 - \mathbf{w}_-^2, \tag{99}$$

which leads to an element-wise quadratic equation that we can solve. Since the equations are the same for each coordinate, we drop the coordinate index. Squaring both sides of Eq.(98) and replacing $w_+^2(t)$ with $\beta(t) + w_-^2(t)$ using Eq.(99), we obtain

$$w_-^4(t) - \beta w_-^2(t) + \alpha^4 = 0. \tag{100}$$

This is a quadratic equation in w_-^2 whose positive solution reads

$$w_-^2 = \frac{-\beta + \sqrt{\beta^2 + 4\alpha^4}}{2} \tag{101}$$

w_+^2 is obtained in similar fashion, leading to

$$w_+^2 = \frac{\beta + \sqrt{\beta^2 + 4\alpha^4}}{2} \tag{102}$$

Which leads to the following expression for the metric tensor ρ and the corresponding dynamics

$$\rho^{-1} = \mathrm{diag}\left(\sqrt{\beta^2 + 4\alpha^4}\right)^{-1} \tag{103}$$

$$\dot{\boldsymbol{\beta}} = -\mathrm{diag}\left(\sqrt{\beta^2 + 4\alpha^4}\right)^{-1} \odot \nabla L(\boldsymbol{\beta}) \tag{104}$$

We can recover the previously discussed phenomenology from this equation. For large α, the β^2 term is negligible and we recover standard gradient flow dynamics with ℓ_2 geometry in the function space. If $\alpha = 0$, the scaling in front of the gradient is proportional to the absolute value of β. Thus higher absolute value coefficients will decay more, which will promote sparsity. Now, does this metric tensor correspond to a Hessian map defining a mirror descent ? To check this we need to solve $\rho = \nabla^2\Psi$ where Ψ is the potential defining the Bregman distance used for mirror descent. In general, a metric tensor is not a Hessian map, but here it is the case, mostly thanks to the diagonal structure. We can then simply integrate each element on the diagonal twice. Performing this double integral yields the following potential

$$\Psi(\alpha, \beta) = \alpha^2 \sum_{i=1}^{d} \left(\frac{\beta}{2\alpha^2} \sinh^{-1}\left(\frac{\beta}{2\alpha^2}\right) - \sqrt{4 + \frac{\beta^2}{\alpha^4}} \right) \tag{105}$$

Up to a constant, this is the same potential as the one implicitly being minimized by the gradient descent.

5.3.3 Comparing explicit and implicit regularization

Is this equivalent to explicity regularization using the ℓ_2 norm ?

$$\boldsymbol{\beta}^R_{\alpha,\mathbf{w}_0} = F\left(\arg\min_{\mathbf{w}} \|\mathbf{w} - \alpha\mathbf{w}_0\|_2^2 \text{ s.t. } L(\mathbf{w}) = 0 \right) \tag{106}$$

$$= \arg\min_{\boldsymbol{\beta}} R_{\alpha,\mathbf{w}_0}(\boldsymbol{\beta}) \text{ s.t.} \mathbf{X}\boldsymbol{\beta} = \mathbf{y} \tag{107}$$

$$\text{where} \quad R_{\alpha,\mathbf{w}_0} = \min_{\mathbf{w}} \|\mathbf{w} - \alpha\mathbf{w}_0\|_2^2 \text{ s.t. } F(\mathbf{w}) = \boldsymbol{\beta}. \tag{108}$$

Using a similar analysis as before, we study the optimization problem in parameter space over $\boldsymbol{\beta}$ which is equivalent to the optimization problem in weight space over $\mathbf{w}$ where we use explicit ℓ_2 regularization. For standard linear regression this is a classical result, gradient descent converges to the min ℓ_2 norm solution (in that case $\boldsymbol{\beta} = \mathbf{w}$). When the initialization is $\mathbf{w}_0 = \mathbf{1}$, one can determine an analytical expression for $R_{\alpha,\mathbf{w}_0}$:

$$R_{\alpha,\mathbf{1}}(\boldsymbol{\beta}) = \sum_i r(\boldsymbol{\beta}_i/\alpha^2) \tag{109}$$

where $r(z)$ is the unique real root of $p_z(u) = u^4 - 6u^3 + (12 - 2z^2)u^2 - (8 + 10z^2)u + z^2 + z^4$. The next figure shows a plot of $r(z)$ next to $q(z)$ obtained from the implicit bias analysis. The functions are very close to

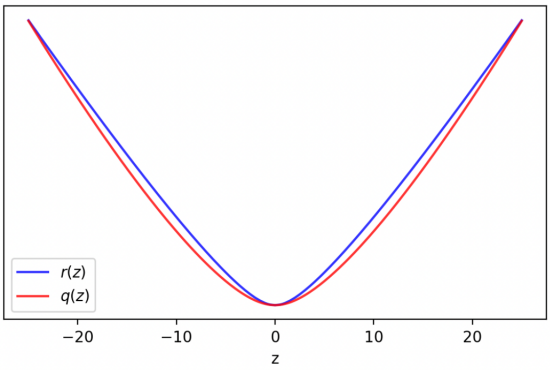

Figure 15: Comparing explicit and implicit regularization

one another, even if a more refined analysis shows that the rich regime can be reached with a polynomial scale of α with r instead of an exponential one with q. See the discussion in [33] for more detail.

5.4 The effect of width

For now we have studied the effect of the scale of the initialization on the regime in which the dynamics operate. What is the effect of the width ? To find out, consider now that the model we want to learn is the function

$$f((\mathbf{U},\mathbf{V}),\mathbf{x}) = \sum_{i=1,..,d,j=1,..,k} u_{i,j}v_{i,j}x[i] = \langle \mathbf{U}\mathbf{V}^\top, \mathrm{diag}(\mathbf{x})\rangle \tag{110}$$

where $\mathbf{U},\mathbf{V} \in \mathbb{R}^{d\times k}$, and we learn the model by minimizing

$$L(\mathbf{U},\mathbf{V}) = \sum_{n=1}^{N} \left(\langle \mathbf{X}_n, \mathbf{M}_{\mathbf{U},\mathbf{V}}\rangle - y_n\right)^2 = \tilde{L}(\mathbf{M}_{\mathbf{U},\mathbf{V}}) \tag{111}$$

using gradient flow, and we defined the map $M_{\mathbf{U},\mathbf{V}} = F(\mathbf{U},\mathbf{V}) = \mathbf{U}\mathbf{V}^\top$ in the notations of the previous section. This model can be considered as an extension of the linear model from above over matrix valued observations, with an additional width parameter k, i.e. a matrix factorization problem or a wide parallel linear network. The goal is to study the combined effect of α and k on the learning regime. Since the number

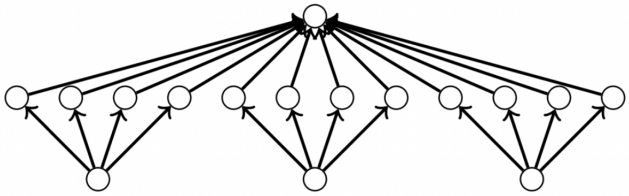

Figure 16: Wide parallel linear network

of parameters grows with the width k, we capture the scale of initialization with the parameter

$$\frac{1}{d}\|\mathbf{M}_{\mathbf{U},\mathbf{V}}\|_F \tag{112}$$

We will see that $\mathbf{M}_{\mathbf{U},\mathbf{V}}$ can be in the kernel regime even if σ goes to 0, depending on the relative scaling with k. In the symmetric case where $\mathbf{M}_{\mathbf{W}} = \mathbf{W}\mathbf{W}^\top$, the gradient flow reads

$$\dot{\mathbf{M}}_{\mathbf{W}(t)} = \nabla\tilde{L}(\mathbf{M}_{\mathbf{W}(t)})\mathbf{M}_{\mathbf{W}(t)} + \mathbf{M}_{\mathbf{W}(t)}\nabla\tilde{L}(\mathbf{M}_{\mathbf{W}(t)}) \tag{113}$$

thus the entire dynamics is described by $\mathbf{M}_{\mathbf{W}\mathbf{W}^\top}$. In the asymmetric case $\mathbf{M}_{\mathbf{U},\boldsymbol{V}}$ this is not true. We may then consider the following "lifted" problem defined by

$$\bar{\mathbf{M}}_{\mathbf{U},\boldsymbol{V}} = \begin{bmatrix} \mathbf{U}\mathbf{U}^\top & \mathbf{M}_{\mathbf{U},\boldsymbol{V}} \\ \mathbf{M}_{\mathbf{U},\boldsymbol{V}} & \boldsymbol{V}\boldsymbol{V}^\top \end{bmatrix} \tag{114}$$

and the corresponding "lifted" datapoints $\bar{\boldsymbol{X}}_n = \frac{1}{2}\begin{bmatrix} 0 & \boldsymbol{X}_n \\ \boldsymbol{X}_n^\top & 0 \end{bmatrix}$, where we consider that the datapoints are matrices in $\mathbb{R}^{d\times d}$, not necessarily diagonal. The implemented function is

$$\bar{f}((\mathbf{U},\boldsymbol{V}),\bar{\boldsymbol{X}}) = \langle \bar{\mathbf{M}}_{\mathbf{U},\boldsymbol{V}}, \bar{\boldsymbol{X}}\rangle \tag{115}$$

the output of which is the same as the original model but now $\bar{\mathbf{M}}_{\mathbf{U},\boldsymbol{V}}$ is the relevant matrix to study the problem. Assume that $\mathbf{U}(0),\boldsymbol{V}(0)$ are initialized with $\mathcal{N}(0,\sigma^2 = \frac{\alpha^2}{\sqrt{k}})$. This way $\mathrm{Var}\left[\mathrm{diag}(\mathbf{U}\boldsymbol{V}^\top)[i]\right] = \alpha^2$. In the case where the measurements commute, the following theorem is proven in [33] (we note that the definition of scaling parameters are different from the paper here, but ultimately the statements are equivalent)

Theorem 5.2. *Let $k \to \infty$, $\alpha(k) \to 0$ and $\mu := \lim_{k\to\infty} \alpha^4\sqrt{k} = \sigma(k)\sqrt{k}$ and suppose that $\mathbf{X}_1, ..., \mathbf{X}_n$ commute. If $\mathbf{M}_{\mathbf{U},\mathbf{V}}(t)$ converges to a zero error solution $\mathbf{M}^*_{\mathbf{U},\mathbf{V}}$, then*

$$\mathbf{M}^*_{\mathbf{U},\boldsymbol{V}} = \arg\min_{\mathbf{M}} Q_\mu(\textit{spectrum}(\mathbf{M})) \textit{ s.t. } \mathbf{L}(\mathbf{M}) = 0 \tag{116}$$

where Q_μ is the same function as before, now applied to the spectrum of $\mathbf{M}$.

We see that the parameter governing whether or not the function q behaves like a square or an absolute value is μ, which involves both the scale of initialization and the width of the problem :

- if $\alpha = o(1/k^{1/4})$, i.e. $\sigma = o(1/\sqrt{k})$, we have an ℓ_1 implicit bias and rich regime,
- if $\alpha = O(1/k^{1/4})$, i.e. $\sigma = O(1/\sqrt{k})$, we have an ℓ_2 implicit bias and kernel regime,
- $\sqrt{k}\alpha^2 \to 0$ leads to the kernel regime, even if $\|\boldsymbol{\beta}(0)\| \simeq \alpha^2 \to 0$

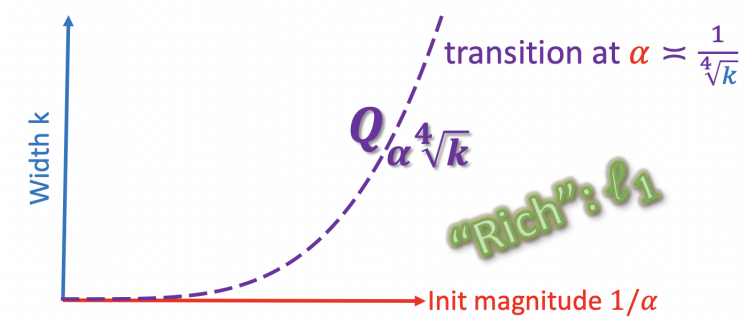

Figure 17: Rich and kernel regime in the matrix factorization problem

5.5 Deep diagonal networks

We now turn to the study of the effect of depth on the learning regime. To do so, we consider a deep variant of the model introduced above, a depth D diagonal linear network :

$$\boldsymbol{\beta}(t) = \mathbf{w}_+(t)^D - \mathbf{w}_-(t)^D \quad \text{and} f_D(\mathbf{w}, \boldsymbol{x}) = \langle \mathbf{w}_+(t)^D - \mathbf{w}_-(t)^D, \boldsymbol{x} \rangle \tag{117}$$

We assume that gradient flow is initialized with $\mathbf{w}_+(0) = \mathbf{w}_-(0) = \alpha\mathbf{1}$, and define the residual at each time

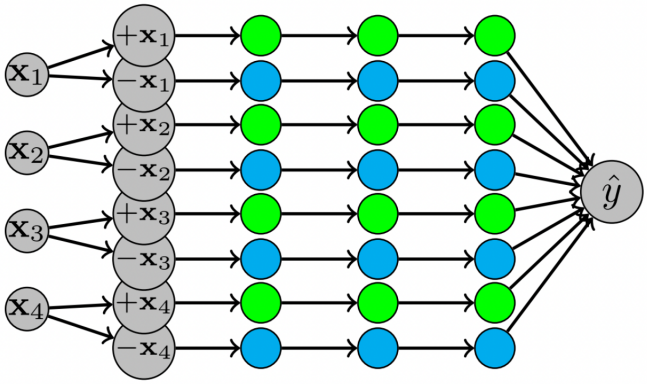

Figure 18: Deep diagonal linear network

step $\mathbf{r}(t) = \boldsymbol{X}\boldsymbol{\beta}(t) - \mathbf{y}$. Writing gradient flow on this model, with the square loss, reads

$$\dot{\mathbf{w}}_+ = -\frac{dL}{d\mathbf{w}_+} = -D\boldsymbol{X}^\top \mathbf{r}(t) \odot \mathbf{w}_+^{D-1} \tag{118}$$

$$\dot{\mathbf{w}}_- = -\frac{dL}{d\mathbf{w}_-} = D\boldsymbol{X}^\top \mathbf{r}(t) \odot \mathbf{w}_+^{D-1} \tag{119}$$

which integrates to

$$\mathbf{w}_+ = \left(\alpha^{2-D} + D(D-2)\boldsymbol{X}^\top \int_0^t \mathbf{r}(t)dt\right)^{-\frac{1}{D-2}} \tag{120}$$

$$\mathbf{w}_- = -\left(\alpha^{2-D} + D(D-2)\boldsymbol{X}^\top \int_0^t \mathbf{r}(t)dt\right)^{-\frac{1}{D-2}} \tag{121}$$

and

$$\boldsymbol{\beta}(t) = \alpha^D \left(1 + \alpha^{D-2}D(D-2)\boldsymbol{X}^\top \int_0^t \mathbf{r}(t)dt\right)^{-\frac{D}{D-2}} \tag{122}$$

$$-\alpha^D \left(1 + \alpha^{D-2}D(D-2)\boldsymbol{X}^\top \int_0^t \mathbf{r}(t)dt\right)^{-\frac{D}{D-2}} \tag{123}$$

Letting $\mathbf{s} = \int_0^\infty \mathbf{r}(\tau)d\tau$ and assuming a zero error solution is achieved, we may write

$$\boldsymbol{\beta}(\infty) = \alpha^D h_D\left(\mathbf{X}^\top \mathbf{s}\right) \text{ and } \boldsymbol{X}\boldsymbol{\beta}(\infty) = \mathbf{y} \tag{124}$$

where $h_D = \alpha^D \left(1 + \alpha^{D-2}D(D-2)z\right)^{-\frac{D}{D-2}} - \alpha^D \left(1 + \alpha^{D-2}D(D-2)z\right)^{-\frac{D}{D-2}}$ (we note that the function with different coefficients in the paper, but ultimately the statements are equivalent). Recall that we are searching for an equivalent problem of the form

$$\boldsymbol{\beta}^* \in \inf_{\boldsymbol{\beta}} Q(\boldsymbol{\beta}) \text{ s.t. } \boldsymbol{X}\boldsymbol{\beta} = \mathbf{y} \tag{125}$$

with the corresponding Lagrangian

$$\mathcal{L}(\boldsymbol{\beta}, \boldsymbol{\nu}) = Q(\boldsymbol{\beta}) + \boldsymbol{\nu}^\top (\boldsymbol{X}\boldsymbol{\beta} - \mathbf{y}) \tag{126}$$

for which the KKT optimality conditions read

$$\boldsymbol{X}\boldsymbol{\beta} = \mathbf{y} \quad \text{and} \quad \nabla Q(\boldsymbol{\beta}^*) = \boldsymbol{X}^\top \boldsymbol{\nu} \tag{127}$$

. We can then match $\mathbf{s}$ with $\boldsymbol{\nu}$ and identify the potential Q by matching its gradient with the inverse of h_D. Then, define $q_D = \int h_D^{-1}$ and $Q_D(\boldsymbol{\beta}) = \sum_i q_D\left(\frac{\beta[i]}{\alpha^D}\right)$. It is proven in [33] that

$$\forall t \quad \|\boldsymbol{X}^\top \int_0^t r(\tau)d\tau\|_\infty \leqslant \frac{\alpha^{2-D}}{D(D-2)} \tag{128}$$

so the domain of h_D is the interval $[-1, 1]$ upon which it is monotonically incresing, ensuring the existence of the inverse mapping h_D^{-1}. Then, for all depth $D \geqslant 2$, this equivalent cost induces a rich implicit bias for $\alpha \to 0$, i.e.

$$\lim_{\alpha\to 0} \boldsymbol{\beta}_{\alpha,D}^\infty = \boldsymbol{\beta}_{\ell_1}^* \lim_{\alpha\to\infty} \boldsymbol{\beta}_{\alpha,D}^\infty = \boldsymbol{\beta}_{\ell_2}^* \tag{129}$$

Although the same behaviour is observed as for the $D = 2$ case, there are actually two main differences. The first one is that, for $D > 2$, explicit regularization does not lead to a sparse bias. Indeed, for

$$R_\alpha(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}=\mathbf{w}_+^D - \mathbf{w}_-^D} \|\mathbf{w} - \alpha\mathbf{1}\|_2^2 \tag{130}$$

leads to

$$R_\alpha(\boldsymbol{\beta}) \xrightarrow{\alpha \to 0} \|\boldsymbol{\beta}\|_{2/D} \tag{131}$$

i.e. the $2/D$ quasi-norm, which leads to less sparse solution than the ℓ_1 norm for $D = 2$. The second difference concerns the intermediate regime, meaning how fast does the scaling at initialization go to zero for the sparsity inducing bias to kick in. We have seen above that, for $D = 2$, an exponentially small scale in α is required to enter the rich regime. As soon as $D = 3$ however, only a polynomially decreasing scale in α is required, and the deeper the network the faster we can reach the rich regime when decreasing α. This is illustrated by plotting the shape of q_D for different values of D

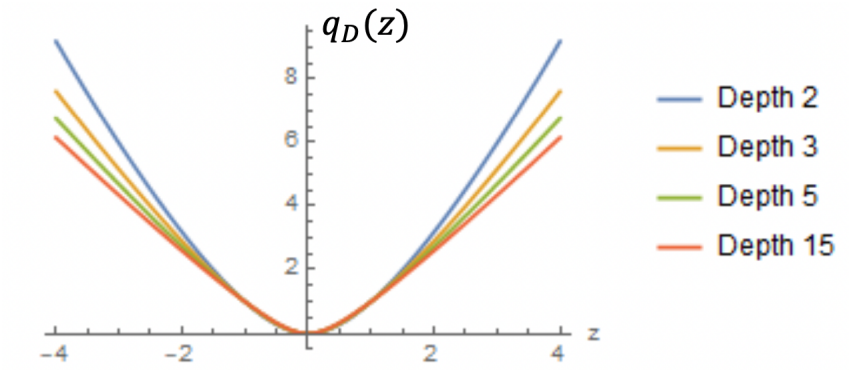

Figure 19: Implicit cost function for deeper networks

5.6 Beyond linear models

Recall our setup, minimizing a loss function defined over a dataset with gradient descent. The predictor function $h_{\mathbf{w}}(\mathbf{x}) = f(\mathbf{w}, \mathbf{x})$ is parametrized by $\mathbf{w} \in \mathbb{R}^p$. So far we have focused on linear models taking the form

$$h_{\mathbf{w}}(\mathbf{x}) = \langle \boldsymbol{\beta}_{\mathbf{w}}, \mathbf{x} \rangle \tag{132}$$

where $\boldsymbol{\beta}_{\mathbf{w}} = F(\mathbf{w}) \in \mathbb{R}^d$. Now consider the generic case where no linearity assumption is made on the predictor. The function F is now defined as a mapping from $\mathbb{R}^d$ to $\mathbb{R}^{\mathcal{X}}$. We may write the dynamics on $h_{\mathbf{w}}(\mathbf{x})$ in similar fashion as before using functional derivatives :

$$\dot{h}_{\mathbf{w}}(\mathbf{x}) = -\nabla F^\top \nabla F \nabla_h L(h) \tag{133}$$

where $\nabla_h L(h)$ is now an element of $\mathbb{R}^{\mathcal{X}}$ and $\nabla F^\top \nabla F$ is a linear map from $\mathbb{R}^{\mathcal{X}} : \mathbb{R}^{\mathcal{X}}$, with a kernel taking the form

$$\rho^{-1}(\mathbf{x}, \mathbf{x}') = \langle \nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{x}), \nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{x}') \rangle \tag{134}$$

Thus the dynamics in parameter space is a gradient flow according to the metric tensor defined by the tangent kernel at each time step. In the kernel regime, this metric tensor is fixed and remains the same as the one at initialization throughout the dynamics, whereas in the generic case it changes at each time step.

6 Lecture 6: Implicit bias with linear functionals and the logistic loss

Summary of the previous lecture We're trying to understand how the choice of optimization geometry (i.e., what our preferred metric is in local updates) affects where the optimization will lead us. Last time, we also spoke about geometry of parameters space (usually just Euclidean geometry), but really what mattered was the geometry in function space. The relationship between parameter and function space is:

$$h_w(x) = f(w, x) \text{ or } h_w(x) = \langle \beta_w, x \rangle \text{ where } \beta_w = F(w) .$$

We can describe what the geometry of our model looks like in function space based on the Jacobian of the model F. Now we can do gradient flow with this inverse metric tensor:

$$\dot{\beta} = -\nabla F^T \nabla F \nabla_\beta L_S(\beta) .$$

We also discussed how this relates to explicit ℓ_2 regularization and showed a setting where it was quite different. The metric tensor really depends on where we are in parameter space, which isn't conducive to studying the dynamics on the function, but in some cases, this can be circumvented.

We can write down the entries of ρ^{-1} :

$$\rho^{-1}(x, x') = \langle \nabla_w f(w, x), \nabla_w f(w, x') \rangle .$$

This is the tangent kernel at the position w. We are conditioning the dynamics on the position w at the given time. In the kernel regime, the location at which the Jacobian is evaluated doesn't change significantly and so the same kernel matrix governs the dynamics at every step / at all times.

To finish our recap from last lecture, the simplest example in which we can see non-trivial behavior is this squared parameterization model: $f(w, x) = \langle \beta_w, x \rangle$ with $\beta_w = F(w) = w_+^2 - w_-^2$, or in the deeper case: $\beta_w = F(w) = |w_+|^D - |w_-|^D$. We talked about initializing at $w(0) = \alpha \vec{1}$, which gives $\beta(0) = 0$. We saw that for $D = 2$, for large α, we got the kernel regime, and when we take α to 0, we get this ℓ_1 regularization. Even when $D \geq 2$, when we took $\alpha \to 0$, we still get ℓ_1 regularization, despite explicit ℓ_2 norm regularization on the parameters not giving ℓ_1 norm regularization in function space.

Today, we will look at logistic loss and see that there is an effect in the classification setting, as well.

6.1 Problem setting and equivalent reformulation

Consider a binary classification problem where we minimize the logistic loss over a data set using gradient descent

$$L_S(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \text{loss} \left(\langle \mathbf{w}, \mathbf{x}_i \rangle, y_i \right) \tag{135}$$

where $\mathbf{w} \in \mathbb{R}^d$,the y_i are binary and $\text{loss}(\hat{y}_i, y_i) = \log(1 + \exp(-\hat{y}_i y_i))$, and we minimize this loss with

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla \mathcal{L}_S(\mathbf{w}_k) \tag{136}$$

In the overparametrized case, i.e. $d > n$, the data is separable and we may minimize the loss by considering any separating hyperplane and taking its norm to infinity. Since the optimal solution diverges, we focus on the direction of the optimal solution :

$$\lim_{k \to \infty} \frac{\mathbf{w}_k}{\|\mathbf{w}_k\|_2} \tag{137}$$

which converges to the max margin separator, i.e. the furthest away (in Euclidian distance) to all the points. This can be equivalently rewritten as a convex opitmization problem under inequality constraints

$$\mathbf{w}^* \in \arg\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|_2 \tag{138}$$

$$\text{s.t.} \quad y_i \langle \mathbf{w}, \boldsymbol{x}_i \rangle \geqslant 1 \tag{139}$$

The Lagrangian for this problem reads

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\nu}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} \nu_i \left(1 - y_i \langle \mathbf{w}_i, \boldsymbol{x}_i \rangle\right) \tag{140}$$

where $\boldsymbol{\nu} \succeq 0$. Primal feasibility then requires

$$y_i \langle \mathbf{w}_i, \boldsymbol{x}_i \rangle \geqslant 1 \quad \text{and} \quad \nabla_{\mathbf{w}} \mathcal{L} = 0 \tag{141}$$

which implies $\mathbf{w} = \sum_i \nu_i \boldsymbol{x}_i$, meaning the separating hyperplane is supported by the data vectors. Complementary slackness then indicates that the only active coefficients ν_i that are non-zero are those associated with datapoints verifying $y_i \langle \mathbf{w}, \boldsymbol{x}_i \rangle = 1$, i.e. where the constraint is active. For any $\mathbf{x}_i$ verifying $y_i \langle \mathbf{w}_i, \boldsymbol{x}_i \rangle > 1$, the corresponding Lagrange multiplier ν_i will be zero.

6.2 Gradient flow dynamics

What does GF look like on this problem? Since we are interested in the interpolating regime, assume that the dynamics converge to a small error ϵ. In this regime, we may approximate the logistic loss by its right hand side tail, i.e.

$$\log(1 + \exp(-\hat{y}_i y_i)) \simeq \exp^{-y_i \hat{y}_i} \quad \forall i \tag{142}$$

The gradient reads

$$-\nabla_{\mathbf{w}} L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} e^{-\langle \mathbf{w}_i, \boldsymbol{x}_i \rangle} \boldsymbol{x}_i \tag{143}$$

Formal proof of what follows can be found in [18, 21]. Intuitively, GF finds a separating direction that will not change much after a certain number of iteration, and increase its norm. We may then write

$$\mathbf{w}_k = \mathbf{w}_\infty g(k) + \rho(k) \tag{144}$$

where $\rho(k) = o(1)$ (a Theorem from [Soudry Hoffer Srebro 18] actually shows $g(k) = log(k)$). Replacing in the expression of the gradient leads to

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} e^{-\langle \mathbf{w}, \boldsymbol{x}_i \rangle} \boldsymbol{x}_i \simeq \frac{1}{n} \sum_{i=1}^{n} e^{g(k) \langle \mathbf{w}_\infty, \boldsymbol{x}_i \rangle - O(1)} \boldsymbol{x}_i \tag{145}$$

which is a linear combination of the data points and gives an explicit expression for the ν_i coefficients. Now denote by γ the margin $\gamma = \min_i \langle \mathbf{w}_\infty, \boldsymbol{x}_i \rangle > 0$, and define the normalized separator

$$\hat{\mathbf{w}}_\infty = \frac{\mathbf{w}_\infty}{\gamma} \tag{146}$$

whose norm will remain finite. Primal feasibility is verified by construction $\langle \mathbf{w}, \boldsymbol{x}_i \rangle \geqslant 1$ is satisfied by construction, and we have seen that the zero gradient condition on $\mathbf{w}$ prescribing it as a linear combination of the data points is also verified. We also see that large values of $\langle \mathbf{w}_\infty, \boldsymbol{x}_i \rangle$ the corresponding coefficient ν_i will decrease very fast as $g(k) \to \infty$, leaving the main contribution to the lowest values which are the $\boldsymbol{x}_i$ for which $\langle \mathbf{w}_\infty, \boldsymbol{x}_i \rangle = 1$. Thus we recover the complementary slackness condition.

6.3 Comparing the squared, logistic and exponential loss

For squared loss, we go to minimum distance from initialization, so the initialization is still important. For the logistic loss, however, the initialization doesn't matter at all – we always go to the max margin solution. This makes sense because given that we diverge anyway, i.e., we go infinitely far from the starting point, it cannot matter where we start. Any finite initialization from far enough away looks like the origin. This is a significant factor that steers differences between squared loss and logistic loss. We could repeat this analysis and show that when you minimize the logistic loss, for any homogenous model, you always go to the minimum ℓ_2 norm solution. Any network with fixed depth and homogeneous activation. We can show this with basically the same proof.

Theorem 6.1 ([22, 18]). *If $L_S(w) \to 0$ and the step size is small enough to ensure convergence in direction, then:*

$$w_\infty \propto \textit{ first order stationary point of } \arg\min \|w\|_2 \textit{ s.t. } \forall i \, y_i f(w, x_i) \geq 1 \, .$$

The first order stationary points of this objective are exactly those that satisfy the KKT conditions. For convex problems, this implies optimality. Here, we have to be a bit more careful. The objective is convex but f is not in general a convex function of w. This suggests that the implicit bias is defined by $R_F(h) = \arg\min_{F(w)=h} \|w\|_2$, i.e., which function h is representable most cheaply in parameter space. These two problems have the same global minimum but this does not mean a first-order stationary point of the first problem are first-order stationary points of the second problem. Relating the two remains largely open.

For squared loss, as the scaling goes to infinity, the entire trajectory converges to the kernel trajectory. In particular, then, the implicit bias is the implicit bias of the kernel (i.e., the limit points are the same). For logistic loss, we can get a similar statement about infinite scale leads to the kernel regime, but the problem is that we can only get it for finite time. That is, if we fix the amount of time for which we optimize and then take the scale to infinity, then we stay inside the kernel regime. These results are presented formally in [8]. For logistic loss, if we change the order of limits for scale of initialization and time of optimization, we reach a first order stationary point of $\arg\min \|w\|_2$ such that the margin condition is satisfied (formal statement in [18] and [22]).

Thus, for logistic loss, the regime we are in is no longer just dependent on α, the initialization scale, but rather on the combination of scale and training loss. For each optimization accuracy, we can ask at what scale we would enter the kernel regime. The transition occurs at $\epsilon \sim \exp(-\alpha^2)$. On the boundary, (under several non-realistic assumptions), the behavior follows the Q_α function with parameter $\frac{\alpha}{\sqrt{\log(1/\epsilon)}}$. Thus, the transition depends on the ordering of ϵ, α limits. While it is true that we have an asymptotic result, it only kicks in when $\epsilon = 10^{-1700}$, so we do not get ℓ_1 regularization in practice.

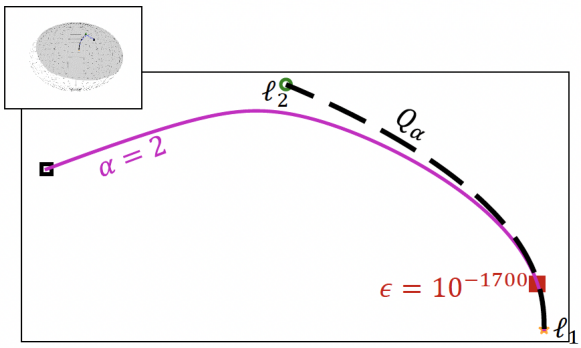

Figure 20: This figure depicts how the predictor behaves with finite but not large initialization scale ($\alpha = 2$). We know that in the long run, we will reach the minimum ℓ_1 norm solution. When the optimization accuracy is finite, we will traverse the whole Q_α path. The dashed black line is the path of minimum Q_α margin solution: solution with margin 1 that minimizes Q_α for corresponding α. As α increases, it seems we converge to this path. (Note, we don't know how to prove this but it seems to hold empirically.) Importantly, while it is true asymptotically in ϵ that minimizing the regularizer in function space corresponds to minimizing the norm of the weights, the ϵ at which it starts to hold is completely impractical.

When we consider deeper models, things only get worse: the asymptotic regime is even harder to reach. It is not well-understood yet what is happening here when α is small. Finally, the width of the network also makes a difference in effective initialization scale.

Other Control Parameters Other control parameters (aside from initialization scale) include how early we stop training, shape of the initialization (i.e., relative scale of the parameters), step size, and stochasticity. The latter has been studied through the lens of batch size and label noise.t

6.4 Matrix Factorization Setting and Commutativity

We are still going to look at a linear model in β, but now β is going to be a matrix. This is a standard least squares objective in β. The main difference is that now, instead of factorizing β element-wise, we are going to factorize $\beta = UV^T$. This formulation captures many things: matrix completion, matrix reconstruction from linear measurements, and even multi-task learning.

Let us study the implicit bias of gradient flow on the factorization in function space, i.e., in matrix space:

$$\dot{U}(t) = -\nabla_U \hat{L}(UV^T) \tag{147}$$

$$\dot{V}(t) = -\nabla_V \hat{L}(UV^T) \tag{148}$$

$$\Rightarrow \dot{\beta} = -(\nabla F^T \nabla F)\nabla \hat{L}(\beta) = -(UU^T \nabla \hat{L}(\beta) + \nabla \hat{\beta} VV^T)\,. \tag{149}$$

Observe that this is a linear transformation of the gradient. We would be interested in writing the transformation in terms of just β, and this is not possible here, as it turns out. Instead, we define:

$$W = \begin{bmatrix} U \\ V \end{bmatrix} \qquad \tilde{\beta} := WW^T = \begin{bmatrix} UU^T & UV^T \\ VU^T & VV^T \end{bmatrix} = \begin{bmatrix} UU^T & \beta \\ \beta^T & VV^T \end{bmatrix}$$

This is still a matrix factorization problem in W, except it is now a symmetric matrix factorization problem. This corresponds to a minimization problem over positive semidefinite matrices. Namely:

$$\min_{\tilde{\beta} \succeq 0} \hat{L}(\tilde{\beta}) = \|\tilde{\mathcal{X}}(\tilde{\beta}) - y\|_2^2\,.$$

for appropriately defined $\tilde{\mathcal{X}}$. The resulting dynamics are: $\dot{\tilde{\beta}} = -(\tilde{\beta}\nabla \hat{L}(\tilde{\beta}) + \nabla \hat{L}(\tilde{\beta})\tilde{\beta}^T)\,.$

What we've shown here is that whenever we have a *non-symmetric* matrix factorization problem, it is a special case of a higher-dimensional *symmetric* matrix factorization problem. In some sense, the real problem we should be looking at *is* this one, since the non-symmetric one hides information about the geometry.

The local geometry of the space in which we search is given by left and right multiplying by $\tilde{\beta}$. Now what we want to see if we can identify this as a Hessian map. Do the dynamics stay in a low-dimensional manifold?

As before, where we are depends on the integral of the residual so far. The residual explains how much to multiply by the observation. If matrices commute, we can solve the differential equation explicitly with the matrix exponential. The metric tensor is a Hessian map, which directly implies we are in low-dimension manifold. In this case, the result is very robust: it doesn't depend on the residuals, nor the loss under which the residuals are computed. It is not robust to step size: could do a ski jump off the manifold now that it is curved. If it is non-commutative, it matters which order in which I multiply on right and left. $\beta(t)$, is a "time-ordered exponential," and we cannot ignore the ordering of the residuals. Even with just two data points, we can enter the whole space. An analogy is parallel parking, where with the forward/backward and left/right controls we can somehow move net to the right.

This non-commutative case is the case we are in in general. This is a well-defined question but one for which the solution is not known.

7 Lecture Series Summary

The approach covered over this lecture series involved attempting to characterize the implicit bias of neural networks from a complexity theory perspective. The main question is to understand which zero-error solution is reached when optimizing an overparameterized problem, and determining this involved decomposing the problem into (1) identify what optimization biases toward based on identifying what complexity measure is being (at least approximately) minimized; followed by (2) evaluating whether this complexity measure is a good one to optimize for the task at hand, i.e., does a minimizer of it generalize well.

A natural question, then, is whether this is the right approach to take. Indeed, it is not always the best description for optimization in neural networks. It remains open whether minimization of this complexity measure, perhaps even approximately, is sufficient to explain most cases.

The ultimate questions: what is true Inductive Bias? What makes reality *efficiently* learnable by fitting a (huge) neural net with a specific algorithm?

Acknowledgements

These are notes from the lecture of Nathan Srebro given at the summer school "Statistical Physics & Machine Learning ", that took place in Les Houches School of Physics in France from 4th to 29th July 2022. The school was organized by Florent Krzakala and Lenka Zdeborová from EPFL.

References

[1] Shun-ichi Amari. *Differential-geometrical methods in statistics*, volume 28. Springer Science & Business Media, 2012.

[2] Peter Bartlett. For valid generalization the size of the weights is more important than the size of the network. *Advances in neural information processing systems*, 9, 1996.

[3] Heinz H. Bauschke, Jérôme Bolte, and Marc Teboulle. A descent lemma beyond lipschitz gradient continuity: First-order methods revisited and applications. *Mathematics of Operations Research*, 42(2):330–348, 2017.

[4] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. In *Proceedings of the National Academy of Sciences*, 2019.

[5] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In *International Conference on Machine Learning*, 2018.

[6] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[7] Lenaic Chizat and Francis Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. In *Conference on Learning Theory*, 2020.

[8] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] Amit Daniely. Sgd learns the conjugate kernel class of the network. *Advances in Neural Information Processing Systems*, 30, 2017.

[10] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From average case complexity to improper learning complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 441–448, 2014.

[11] Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Implicit bias of gradient descent on linear convolutional networks. In *Advances in Neural Information Processing Systems*, 2018.

[12] Suriya Gunasekar, Blake Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nathan Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, 2017.

[13] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[14] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[15] Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *Conference On Learning Theory*, 2018.

[16] Zhiyuan Li, Yuping Luo, and Kaifeng Lyu. Towards resolving the implicit bias of gradient descent for matrix factorization: Greedy low-rank learning. In *International Conference on Learning Representations*, 2021.

[17] Haihao Lu, Robert M. Freund, and Yurii Nesterov. Relatively smooth convex optimization by first-order methods, and applications. *SIAM Journal on Optimization*, 28(1):333–354, 2018.

[18] Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks. *arXiv preprint arXiv:1906.05890*, 2019.

[19] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[20] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[21] Edward Moroshko, Blake E Woodworth, Suriya Gunasekar, Jason D Lee, Nati Srebro, and Daniel Soudry. Implicit bias in deep linear classification: Initialization scale vs training accuracy. *Advances in neural information processing systems*, 33:22182–22193, 2020.

[22] Mor Shpigel Nacson, Suriya Gunasekar, Jason Lee, Nathan Srebro, and Daniel Soudry. Lexicographic and depth-sensitive margins in homogeneous and non-homogeneous deep models. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4683–4692. PMLR, 09–15 Jun 2019.

[23] AS Nemirovsky and DB Yudin. Problem complexity and optimization method efficiency. *M.: Nauka*, 1979.

[24] Yurii Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

[25] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, 2015.

[26] Greg Ongie, Rebecca Willett, Daniel Soudry, and Nathan Srebro. A function space view of bounded norm infinite width relu nets: The multivariate case. In *International Conference on Learning Representations*, 2020.

[27] Pedro Savarese, Itay Evron, Daniel Soudry, and Nathan Srebro. How do infinite width bounded norm networks look in function space? In *Conference On Learning Theory*, 2019.

[28] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[29] Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Learnability, stability and uniform convergence. *The Journal of Machine Learning Research*, 11:2635–2670, 2010.

[30] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[31] Vladimir Vapnik and Alexey Chervonenkis. Theory of pattern recognition, 1974.

[32] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 2017.

[33] Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In *Conference on Learning Theory*, pages 3635–3673. PMLR, 2020.

[34] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.