MSRI's *Mathematics of Machine Learning* Summer School

Nico Courts

July 29-Aug 9, 2019

Introduction

These notes were the ones I took while attending MSRI's "Mathematics of Machine Learning" summer school at the University of Washington during the two weeks above.

1 Statistical Learning

This series of lectures was given by Robert Shapire from Microsoft Research.

The topic of the talks here are somewhere at hte nexus of supervised and statistical learning. The big idea concerning what we want o do is to learn how to do better in the future based on experience from the past. The hope is that these methods can be fully automated. This is a proper subset of AI, but definitely a core area of how we do it.

We are looking for easy to use, flexible algorithms that give us useful results. Ideally they would be interpretable, but this is usually a secondary result.

He believes that ML can help us to understand how learning happens in non-machine entities (cool!). For instance, one can consider the question of nature vs nuture and to ask the question "what is simplicity and why is it useful?"

1.1 The Problem

In this course, we will be focusing on a single kind of (simple) learning problem: learning from example. Given a set of examples, we are looking for an algorithm that can classify instances given objects.

For instance, character recognition. We have a set of handwritten characters and the letter it depicts. Then we want to feed this into a learning algorithm that gives us a prediction rule, that given a handwritten character, outputs the predicted character.

We were then given a couple of examples to try. The particulars weren't important, but the takeaway here is

- We needed enough data to say something meaningful.

- We looked for a rule that fit the observations. We want the rule to be consistent or to contain few mistakes.
- We were looking for rules that were "simple." Importantly, our notion of simplicity changed when we converted one set of numbers to binary.

The first condition is always the case: "more is more." But the latter two fit into a tradeoff between fit and simplicity.

The main questions we will be trying to answer:

- How much data do we need?
- How do we define simplicity?
- How much complexity do we need to represent our problem?
- What is the tradeoff mentioned above?

1.2 Studying the Problem Mathematically

We have to come up with a formal model for the problem so that we can do mathematics with it. The learning model should answer some basic questions: *What is the goal of learning?* and *How is the learning happening?*.

We begin the description of the model:

1.2.1 Definition: An **instance** (or sometimes informally **example**) x is an object in a space called the **instance space** or **domain** X.

1.2.2 Remark: To each instance we assign a **label** or **class**. In this course we will be simplifying to only consider two classes, denoted 0 and 1 or $+$ and $-$.

1.2.3 Remark: We work under an assumption that there is an underlying function

$$c : X \to \{0, 1\}$$

called the (target) **concept** that we are trying to learn. Later we will relax this assumption.

1.2.4 Definition: We define the **hypothesis** to be (similar to the concept) a map

$$h : X \to \{0, 1\}$$

that holds our current beliefs.

We write $\mathcal{C}$ to denote the **concept class**, the class of all possible functions we are exploring (we assume we know a class to which c belongs). We will further assume that our data were generated *independently* and at random. It is limiting to make an assumption about the distribution from which these were drawn. So we will just say there is one.

1.2.5 Definition: We define

$$\text{err}_D(h) = \Pr_{x\sim D}[h(x) \neq c(x)]$$

to be the (generalization) **error**.

Then we obviously ask that $\text{err}_D(h) < \varepsilon$ for some small ε, but furthermore we want that this happens with high probability:

$$\Pr[\text{err}_D(h) \leq \varepsilon] \geq 1 - \delta$$

and in this case, we say that that the rule is probably approximately correct (PAC).

1.2.6 Definition: The class $\mathcal{C}$ is **PAC-learnable** by $\mathcal{H}$ (a hypothesis space) if there is an algorithm A such that for all $c \in \mathcal{C}$ and for all distributions D and for all positive ε and δ, then A takes "polynomial many" random variables $X_i \sim D$ which outputs $h \in \mathcal{H}$ such that

$$\Pr[\text{err}_D(h) \leq \varepsilon] \geq 1 - \delta$$

1.2.7 Remark: This is not an easy problem, per se, but it is one of the most simple classes of algorithms to work with.

Example 1.1
Let $X = \mathbb{R}$ and let $\mathcal{C}$ be of the form c_r for $r \in \mathbb{R}$ where

$$c(x) = \begin{cases} 1, & x \geq r \\ 0 & \text{otherwise} \end{cases}$$

so $\mathcal{C}$ is the *set of positive half lines*. Then if we have some data, we can set our hypothesis $h = c_b$ where b is the leftmost positive value. Then if $c = c_r$, then $\text{err}_D(h)$ has to do with the interval $[r, b]$. Let $b - r = \varepsilon$.

Now restrict attention to $X = [0, 1]$ and notice that if we got a training point within ε of c (to the right), then we would have gotten a smaller difference. Thus

$$\Pr[\text{err}_D(h) < \varepsilon] \leq \Pr[\text{no } x_i \text{ are in } [b, b + \varepsilon]]$$

$$
\begin{aligned}
&= \prod_1^m \Pr[x_i \in R] \\
&= (1-\varepsilon)^m \\
&\leq e^{-\varepsilon m}
\end{aligned}
$$

using the fact that $1 + x \leq e^x$ for all x.

1.3 Sufficient conditions for learning

Now we consider the case when our hypothesis space $\mathcal{H}$ is finite.

1.3.1 Theorem
Let A be an algorithm that finds a hypothesis $h_A \in \mathcal{H}$ which is consistent with m random training examples (as before) where

$$m \geq \frac{1}{\varepsilon}(\ln|\mathcal{H}| + \ln\frac{1}{\delta})$$

then $\Pr[\text{err}_D(h_A) > \varepsilon] \leq \delta$.

1.3.2 Remark: Note that we have completely done away with trying to get a handle on the concept class here.

1.3.3 Remark: Equivalently, with probability of $1-\delta$,

$$\text{err}_D(h_A) \leq \frac{\ln|\mathcal{H}| + \ln|\frac{1}{\delta}|}{m}$$

The idea to take away here is that $\ln|\mathcal{H}|$ is in some way measuring "description length" of a hypothesis: you have to describe how to distinguish the correct one from the wrong ones. Think popping these in a binary tree. Then the m bound gives you some sort of way to manage the complexity of the algorithm.

1.3.4 Theorem
Assume we have $m \geq \frac{1}{\varepsilon}(\ln|\mathcal{H}| + \ln|\frac{1}{\delta}|)$ examples. Then with probability $1 \geq 1-\delta$, for all $h \in \mathcal{H}$ if h is consistent then $\text{err}_D(h) \leq \varepsilon$ (here we often write "h is ε-good.")

Proof
We prove the converse: the probability of there being an $h \in \mathcal{H}$ that is consistent and ε-bad is less than δ. Notice that whether h is consistent with the training set is a random variable (since it depends on choice of training set). Whether it is ε-bad is not random. So if $\mathcal{B}$ is the set of ε-bad hypotheses, then we really want

$$\Pr[\exists h \in \mathcal{B} : h \text{ is consistent}] \leq \sum_{g \in \mathcal{B}} \Pr[h \text{ is consistent}]$$

using the union bound.

Now fix $g \in \mathcal{B}$ and we want to compute the probability of it being consistent. That is, $h(x_i) = c(x_i)$ for all i. Since the samples are independent, this probability is

$$\prod_1^m \Pr[h(x_i) = c(x_i)] \leq (1-\varepsilon)^n$$

Therefore the original probability is

$$|\mathcal{B}|(1-\varepsilon)^m \leq |\mathcal{H}|e^{-\varepsilon m} \leq \delta$$ ♠

1.4 When does this work when we have infinite hypothesis classes?

We saw in the first example that infinite classes can still be PAC-learnable, but where is the line drawn?

To see this, consider classifications of finite subsets. But any choice of threshold between any two points on $\mathbb{R}$ will give the same classification! So really we only have something like 5 different hypothesis (in the form of a threshold function at least). Compare that with 2^m behaviors one could consider in general!

More formally, if we are given a set $S = \langle x_1, \ldots, x_m \rangle$ of instances, then we can consider the collection of all $\langle h(x_1), \ldots, h(x_m) \rangle$ for all $h \in \mathcal{H}$. We call this set $\Pi_{\mathcal{H}}(S)$ and define

$$\Pi_{\mathcal{H}}(m) = \max_{|S|=m} |\Pi_{\mathcal{H}}(S)|$$

and call this the **growth function.** We can use this function to get some sort of handle on the complexity of the problem, and in fact it plays a similar role in an analogous theorem to the one we saw earlier:

1.4.1 Proposition
Give m training examples, with probability $1-\delta$ for all $h \in \mathcal{H}$ if h is consistent then

$$\text{err}_D(h) \leq \mathcal{O}\left(\frac{\ln \Pi_{\mathcal{H}}(2m) + \ln \frac{1}{\delta}}{m}\right)$$

1.4.2 Remark: The proof of this result can't quite continue in the same way as before because we need to select a training set *before we can know the hypothesis space*, so the dependence gets in the way. There is a fancier proof but we won't do it here (although there will be another proof later that is similar).

1.4.3 Remark: Notice that if $\Pi_{\mathcal{H}}(m)$ is polynomial, this reduces nicely and as m gets larger, the error drops. Thus learning is possible.

1.4.1 More on the growth function

There are at least some cases when $\Pi_{\mathcal{H}}(m) \in \mathcal{O}(m^d)$ for some constant d. In the worst case, $\Pi_{\mathcal{H}}(m) = 2^m$. In fact, these are the only two cases that can happen! Furthermore, it ends up that these two cases correspond exactly to when learning is possible!

To see why, we need a new definition:

1.4.4 Definition: A sample S of size m is **shattered** by $\mathcal{H}$ if all possible behaviors/labelings are possible: that is, $\Pi_{\mathcal{H}}(S) = 2^m$.

Example 1.2
Given $\mathcal{H}$ to be the set of all closed intervals $[a, b] \subseteq \mathbb{R}$, any set of two points in $\mathbb{R}$ is shattered by $\mathcal{H}$. But you cannot shatter a set of three points in $\mathbb{R}$.

1.4.5 Definition: The **Vapnik-Chervinenkis (or VC) dimension** is

$$\text{VCdim}(\mathcal{H}) = \max\{|S| : S \text{ is shattered by } \mathcal{H}\}$$

1.4.6 Remark: So in the exercise above, VCdim(intervals) = 2.

Another set of hypotheses are the linear threshold functions in $\mathbb{R}^n$, and the VC dimension here is $n + 1$. If you fix your planes to be subspaces, the VC dimension is n.

1.4.7 Remark: A good heuristic measure for VC dimension is the number of parameters. There are pathological examples, but they are truly that.

We have a nice little lemma:

1.4.8 Lemma (Sauer)
If $d = \text{VCdim}(\mathcal{H})$, then

$$\Pi_{\mathcal{H}}(m) \leq \sum_{0}^{d} \binom{m}{i}.$$

1.4.9 Remark: A "nicer" upper bound leverages

$$\sum \binom{m}{i} \leq \left(\frac{em}{d}\right)^d$$

whenever $m \geq d \geq 1$.

Plugging this into the bound on the error of a hypothesis, we get that (forgetting the relatively small log term) that the VC dimension d gives us a complexity measure—that is, it gives a bound on the error of a consistent hypothesis.

1.5 Non-consistent Hypotheses

The results so far have been great, but has always assumed that there is a consistent hypothesis to find. But what if our data is noisy? Then learning the concept is no longer about finding the concept that matches *all* the given data.

Now we consider data $(x, y) \sim D$ to be a pair jointly sampled from some distribution D. We do basically the same thing to compute error:

$$\operatorname{err}_D(h) = \Pr_{(x,y)\sim D}[h(x) \neq y].$$

We are still trying to minimize the generalization error $\operatorname{err}(h)$ over $\mathcal{H}$ given some sample (x_i, y_i) sampled from D.

1.5.1 Definition: Write the **empirical error** as

$$\widehat{\operatorname{err}}(h) = \frac{1}{m}\sum_{1}^{m} \delta_{h(x_i)\neq y_i}$$

And then we can follow the route called *empirical risk minimization* (ERM) that finds an empirical solution:

$$\hat{h} = \operatorname*{argmin}_{h\in\mathcal{H}} \widehat{\operatorname{err}}(h).$$

We want to get to the situation in which, with probability greater than $1-\delta$, for all $h \in \mathcal{H}$,

$$|\operatorname{err}(h) = \widehat{\operatorname{err}}(h)| \leq \varepsilon.$$

This result is a kind of **uniform convergence result** which is the kind of thing we are after. Obviously this is useful since it allows us to bridge the gap between imperfect information and the true concept.

Now $\delta_{h(x_i)\neq y_i} = 1$ with probability $\operatorname{err}(h)$ and zero otherwise.

Example 1.3
A quick aside: Consider IID variables $Z_1, \ldots, Z_m$ with $Z_i \in [0, 1]$. Let $p = \mathbb{E}[Z_i]$ and then let $\hat{p} = \frac{1}{m}\sum_i Z_i$ be the empirical mean.

Then there is a result called **Hoeffding's inequality**, which claims

$$\Pr[\hat{p} \geq p + \varepsilon] \leq e^{-2e\varepsilon^2 m}.$$

Notice that what this says is that the empirical average converges to its expected value. So we can think of this as a function converging

$$f(Z_1, \ldots, Z_m) \to \mathbb{E}[f(Z_1, \ldots, Z_m)].$$

This idea leads us to:

1.5.2 Lemma (McDiarmid's Inequality)
Suppose $f(z_1, \ldots, z_m)$ is real valued such that chainging z_i changes f by at most c_i, and furthermore the z_i are independent, but not necessarily identically distributed. Then you get a nice bound on the error (I didn't get it in time but I am sure it is online.)

So if we let $Z_i = \delta_{h(x_i)\neq y_i}$ and let $p = \mathbb{E}Z_i = \operatorname{err}(h)$ and $\hat{p} = \widehat{\operatorname{err}}(h)$, then for any particular $h \in \mathcal{H}$,

$$\Pr[|\operatorname{err}(h) - \widehat{\operatorname{err}}(h)| \geq \varepsilon] \leq 2e^{-2\varepsilon^2 m}.$$

So when $\mathcal{H}$ is finite, we can use the union bound to say

$$\Pr[\exists h \in \mathcal{H} : |\operatorname{err}(h) - \widehat{\operatorname{err}}(h)| \geq \varepsilon] \leq 2|\mathcal{H}|e^{-2\varepsilon^2 m}$$

then setting the RHS equal to δ, we get that with probability $1-\delta$, for all $h \in \mathcal{H}$,

$$|\operatorname{err}(h) - \widehat{\operatorname{err}}(h)| \leq \mathcal{O}\left(\sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{m}}\right)$$

or in another (slightly weaker but sometimes more suggestive) form:

$$\operatorname{err}(h) \leq \widehat{\operatorname{err}}(h) + \mathcal{O}\left(\sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{m}}\right)$$

1.5.3 Remark: Notice that this really (finally) ties together how several aspects impact learning: it relates the true error to the sample size, the complexity class of the concept, and the consistency of the hypothesis with the sample data.

1.5.4 Remark: Notice that as the complexity of your hypotheses increase, we expect the empirical error to decrease, but if we assume that the above is truly a good bound for $\operatorname{err}(h)$, we would expect that the true error will begin by dropping, but eventually the growth term from $\ln|\mathcal{H}|$ will overtake things and cause the error to rise again.

This is one way to think about the concept of **overfitting**: the complexity of your hypothesis functions can incorporate more and more pathological functions that are less likely to be realistic.

1.6 Another Complexity Measure

Suppose we have a labelled sample S consisting of (x_i, y_i), where we can now assume the label space is ± 1. Then if we have a hypothesis, we can compute the training error $\widehat{\operatorname{err}}(h)$:

$$\widehat{\operatorname{err}}(h) = \frac{1}{m}\sum_i \delta_{h(x_i)\neq y_i} = \frac{1}{m}\sum_i \frac{1 - y_i h(x_i)}{2} = \frac{1}{2} - \frac{1}{2}\left(\frac{1}{m}\sum_i y_i h(x_i)\right)$$

and the term in parentheses is another quantity we can consider which is just a constant away from training error.

So we can consider computing

$$\mathbb{E}_\sigma[\max_{h\in\mathcal{H}} \sum_i \sigma_i h(x_i)]$$

where σ_i are ± 1 equally likely. In the case that $|\mathcal{H}| = 1$, we get that this value is zero and when $\mathcal{H}$ shatters S (the set of all m-tuples of ± 1), the value is 1. These serve as the two extremes, obviously.

We would like to work with this towards describing convergence (and rates thereof) towards the generalization error. To do this, we need to broaden the context slightly. Let $f : Z \to \mathbb{R}$ (standing in for the hypothesis), a space $\mathcal{F}$ ($\mathcal{H}$), and define the **(empirical) Rademacher complexity** to be

$$\hat{R}_{\mathscr{F}}(S) = \mathbb{E}_\sigma \left[\sup_{f\in\mathscr{F}} \frac{1}{m} \sum_i \sigma_i f(z_i) \right]$$

and the expected Rademacher complexity is

$$R_{\mathscr{F}}(m) = \mathbb{E}_S[\hat{R}_{\mathscr{F}}(S)]$$

Then we want to show that for all $f \in \mathscr{F}$ that $\hat{\mathbb{E}}_S[f] \to \mathbb{E}[f]$. Specifically,

1.6.1 Theorem

Let $\mathscr{F}$ be a family of functions $f : Z \to [0, 1]$ and let S be a collection of z_i where $z_i \sim D$. Then with probability $1 - \delta$, and for all $f \in \mathscr{F}$,

$$\mathbb{E}[f] \leq \hat{\mathbb{E}}_S[f] + 2 \cdot \Psi + \mathcal{O}\left(\sqrt{\frac{\ln(1/\delta)}{m}}\right)$$

where Ψ is either $\hat{R}_{\mathscr{F}}(S)$ or $R_{\mathscr{F}}(m)$.

Proof

Let $\Phi(S) = \sup_{f\in\mathscr{F}}(\mathbb{E}[f] = \mathbb{E}_S[f])$. Then first we show (via McDiarmid) that

$$\Phi(S) \leq \mathbb{E}_S[]\Phi(S)] + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

So then we have reduced the problem with finding some bound for $\mathbb{E}_S[\Phi(S)]$.

Then let S' to be a hypothetical second sample (called the "ghost sample"). Compute

$$\mathbb{E}_S[\Phi(S)] = \mathbb{E}_S[\sup_f(\mathbb{E}[f] - \hat{\mathbb{E}}_S[f])] \leq \mathbb{E}_{S,S'}[\sup(\hat{\mathbb{E}}_{S'}[f] - \hat{\mathbb{E}}_S[f])]$$

There is something to prove in the last line there but he didn't show us. He said there was a slide somewhere. :)

For the next step, go through the two samples S and S' one index at a time and for each i with equal probability either swap z_i and z'_i or do nothing. Call the resulting samples T and T'. But then the following two expressions have the same distribution:

$$\hat{\mathbb{E}}_{S'}[f] - \hat{\mathbb{E}}_S[f] \quad \text{and} \quad \hat{\mathbb{E}}_{T'}[f] - \hat{\mathbb{E}}_T[f]$$

and so if we let σ_i denote the random variable for whether we swapped at the i^{th} step ($+1$ if we swapped z_i and z'_i and -1 otherwise), the latter is

$$\frac{1}{m}\sum_i \sigma_i f(f(z'_i) - f(z_i))$$

so we get

$$\mathbb{E}_{S,S'}[\sup_f(\hat{\mathbb{E}}_{S'} - \hat{\mathbb{E}}_S[f])] = \mathbb{E}_{S,S'}\left[\sup_f\left(\frac{1}{m}\sum_i \sigma_i f(f(z'_i) - f(z_i))\right)\right]$$

Finally we can show (another slide) that this value is at most $2R_{\mathscr{F}}(m)$ and then use McDiarmid's once more to show that the expected Rademacher complexity is close enough to that of the empirical RC. ♠

Now the whole point was to show that $\text{err}(h) \approx \widehat{\text{err}}(h)$. One can show the empirical Rademacher complexity of the class of loss functions comprised of indicator functions $l(x, y) = \delta_{h(x)\neq y}$ is twice that of $\mathcal{H}$ itself. You can easily plug this in to get $\mathbb{E}[l_h]$. Then you can get what you want.

Now if $|\mathcal{H}| < \infty$, (careful, he made a notation error earlier so I am switching subscripts and arguments in Rademacher complexities–they are still the same)

$$\hat{R}_S(\mathcal{H}) \leq \sqrt{\frac{2\ln|\mathcal{H}|}{m}}.$$

If the space is infinite, then we choose a representative hypothesis for each behavior seen in applying all hypotheses to S. Call this space $\mathcal{H}'$ and notice that $|\mathcal{H}'| = |\Pi_{\mathcal{H}}(S)|$. Then one can show that the empirical Rademacher complexity of $\mathcal{H}'$ is the same as that of $\mathcal{H}$!

1.7 Weak Learning

At the end of the last lecture we discussed the ideal of weak vs. strong learning. In weak learning, we only require that there is an algorithm yielding a hypothesis h such that $\text{err}_R(h) \leq \frac{1}{2} - \gamma$ for some $\gamma > 0$. But surprise!

1.7.1 Theorem

A class is strongly learnable if and only fif it is weakly learnable.

Today we will discuss why. We want to convert any algorithm that weakly learns a class into one that strongly learns the class. The idea here is that a class $\mathcal{C}$ is either completely (PAC) learnable or else there is no algorithm that learns the concept any better than random chance.

1.7.1 The Ada Boost Algorithm

We will assume we are given a weak learning algorithm A and data $(x_1, y_1), \ldots, (x_m, y_m) \sim \mathscr{D}$. We want to find a hypothesis H whose error $\text{err}_{\mathscr{D}}(H)$ is arbitrarily small.

1.7.2 Definition: A theorem that can take a weak learning algorithm and convert it into an ε-good hypothesis is called a **boosting algorithm.**

The basic idea is what one would expect; run A a total of T times, extracting weak hypotheses $h_1, \ldots, h_T$ and somehow combine these in a way to get a strong hypothesis H. There is a problem with just feeding the same data in to A over and over (e.g. A may be deterministic). So instead we need to find a way of creating distributions $D_1, \ldots, D_T$ that "focus in" on the bad/difficult examples that the weak hypotheses fail on.

1.7.3 Remark: Notice that the definition of learnability means that there has to be an algorithm that works **for all true distributions**. So for instance, the fact that we can learn something about the weather, but can't get it ε-good. This is because our algorithms only work well (enough) on a particular distribution.

Let $\varepsilon_t = \text{err}_{D_t}(h_t) = \frac{1}{2} - \gamma_t$ and notice $\gamma_t \geq \gamma > 0$. Then define the following distributions: D_1 will be the uniform distribution, so the weight on the i^{th} training datum: $D(i) = \frac{1}{m}$. Then define the rest iteratively:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t}, & h_t(x_i) = y_i \\ e^{\alpha t}, & \text{otherwise} \end{cases} = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

where Z_t is a normalization constant and

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right) > 0$$

Then we output H which is defined as

$$H(x) = \text{sign} \left(\sum_1^T \alpha_t h_t(x) \right).$$

That is, we let a weighted sum (according to the error on the provided distribution) tells us what to select.

1.7.4 Theorem

$$\widehat{\text{err}}(h) \leq \prod_1^T 2\sqrt{\varepsilon_t(1 - \varepsilon_t)} = \prod \sqrt{1 - 4\gamma_t^2} \leq \exp\left(-2 \sum \gamma_t^2 \right)$$

So if $\gamma_t > \gamma$, we get

$$\widehat{\text{err}}(h) \leq e^{-2\gamma^2 T}$$

Proof
He went too fast. Slides are somewhere supposedly. ♠

So we get a bound on training error, great! Let's turn it into something about generalization error: if $h_t \in \mathcal{H}$ and $d = \text{VCdim}(\mathcal{H})$, then with probability $1 - \delta$ (see problem session)

$$\text{err}(H) \leq \widehat{\text{err}}(H) + \widetilde{\mathcal{O}}\left(\sqrt{\frac{Td + \ln 1/\delta}{m}}\right)$$

where $\widetilde{\mathcal{O}}$ means that we ignore log factors.

Then something that may be surprising is that as T grows, we may expect that eventually we will overfit our training data, but in practice you can often run this very far out and just keeping better results! in fact, your output H will continue to get better even after it achieves zero error on the training set (since the underlying h_t are still incorrect).

But this points to the issue: the test error doesn't tell the whole story. Instead we are interested in measuring the *confidence* the algorithm has in the combined classifier. So how do we do that? Recall that to determine H we held an "election" for the best output. What we are interested in here is the **margin** (the difference between the fraction of votes for and against an output): we call the margin the *weighted fraction of the h_t's correct minus the weighted fraction of the h_t's incorrect.*

Doing some math, you get

$$\text{margin} = y \cdot \frac{\sum \alpha_t h_t(x)}{\sum \alpha_t}$$

Since we are about out of time, we are sprinting to the finish line: Doing an analysis, we can show that this algorithm tends to push the margins towards the positive. Then you prove that this phenomenon tends to create better generalization performance. Note that we can actually remove the dependence on T by doing the following:

$$\text{err}(H) \leq S + \widetilde{\mathcal{O}}\left(\frac{d/\theta^2 + \ln 1.\delta}{m}\right)$$

where S is the fraction of training examples with margin $\leq \theta$.

2 Convex Optimization

This series was given by Sebastien Bubeck from Microsoft Research.

2.1 Fundamentals

Let $K \subseteq \mathbb{R}^n$ be a convex set: that is one such that the straight line segment between two points in K is contained entirely in K.

2.1.1 Definition: The map $f : K \to \mathbb{R}$ is called **a convex function** if

$$f((1-\gamma)x + \gamma y) \leq (1-\gamma)f(x) + \gamma f(y).$$

2.1.2 Remark: Then if f is differentiable, this implies

$$\frac{f(x+\gamma(y-x)) - f(x)}{\gamma} \leq f(y) - f(x)$$

and as $\gamma \to 0$, the LHS tends to $\nabla f(x) \cdot (y-x)$. Thus

$$f(y) \geq f(x) + \nabla f(x) \cdot (y-x).$$

Then the goal of this course is to "find" $\operatorname{argmin}_{x \in K} f(x)$. A question we ask ourselves: how are f and K specified? Mostly in these lectures we will be focusing on *simple* K and functions f such that $\nabla f(x)$ can be computed (for any x).

2.2 Examples in machine learning

2.2.1 Regression

Here we have a dataset: $(a_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$ are the elements. Then we want to find a rule that assigns y to $\mathbf{a}$. We focus on *linear* rules. That is, we are looking for a linear space in $\mathbb{R}^{n+1}$ that approximates the "true" values.

Tentatively we let our function be $a \mapsto x \cdot a$, parameterized by $x \in \mathbb{R}^n$. Then for each x we can evaluate how the tentative function fits our data. More specifically:

$$\frac{1}{m} \sum_{1}^{m} l(x \cdot a_i, y_i)$$

where l is some loss function.

There are several loss functions one might consider:
Least Squares:

$$l(u, v) = (u - v)^2$$

The upshot here, however, is that the evaluation function above in this case is convex! The idea here is that we are applying a linear function to a convex function. The other nice thing is that if you make a modeling assumption about the distribution underlying the label distribution, nice things happen.

Specifically, assume that $y \sim \mathcal{N}(a \cdot \underline{x}, \sigma^2)$ where $\underline{x}$ is the *true value*. Notice we haven't put any assumption on the draw distribution, just the relationship between the (fixed) a_i and y given the truth x. One can compute

$$\frac{1}{2\pi\sigma^2} \exp(-\frac{1}{2\sigma}(y_i - a_i \cdot x)^2).$$

Thus the likelihood of the entire data set is the product:

$$\frac{1}{(2\pi\sigma^2)^{m/2}}\exp(-\frac{1}{2\sigma^2}\sum_1^m(y_i - a_i\cdot x)^2)$$

and so we recover the least squares loss function as the objective we want to minimize. That is, the maximum likelihood estimator minimizes least squares fit.

2.2.2 Classification

Now restrict $y \in \{\pm 1\}$. A natural loss is

$$l(u,v) = \delta_{\text{sign}(u)\neq\text{sign}(v)}.$$

This is a problem for this class since it is non-convex.

So instead we define **support vector machines** using the loss

$$l(u,v) = \max(0, 1-uv)$$

which is convex

Another example is the **logistic loss**

$$l(u,v) = \log(1-e^{-uv})$$

which you can see is a smooth upper bound on the 0-1 loss we started with. Minimizing this function is equivalent to minimizing the MLE for the *logistic model:*

Denote $p(a) = \mathbb{P}(+1|a)$. Then we assume that the "scale" of our probability is given by a linear function:

$$\log\left(\frac{p(a)}{1-p(a)}\right) = \underline{x}\cdot a \Leftrightarrow p(a) = \frac{1}{1+e^{-\underline{x}\cdot a}}$$

and then the **logistic loss** is the negative log-likelihood of the logistic model.

2.2.3 Graphical Models

Given $a_1,\ldots,a_m \in \mathbb{R}^n$, we want to infer the correlation structure of the variables (how are they all related).

Let's start from a (Gaussian) model: we assume we are drawing IID samples from $\mathcal{N}(\mu,\underline{\Sigma})$. A fact: if $(\Sigma^{-1})_{ij} = 0$, then x_i and x_j are independent, conditioned on the rest. Thus the goal is to estimate Σ^{-1}. So if the truth is Σ, the density of $a_1,\ldots,a_n$ is

$$\frac{1}{\sqrt{(2\pi)^n\det(\Sigma)}}\exp(-\frac{1}{2}(a_i-\mu)^T\Sigma^{-1}(a_i-\mu))$$

and by taking the negative log likelihood:

$$c + \frac{m}{2}\log\det\Sigma + \frac{1}{2}\sum_1^m(a_i-\mu)^T\Sigma^{-1}(a_i-\mu) = C + \frac{m}{2}\left(-\log\det\Sigma^{-1} + \text{tr}(\Sigma^{-1}S)\right)$$

where c is some constant and S is the sample covariance matrix. Then to estimate Σ^{-1} via MLE, we want to solve

$$\text{argmin}_{X>0} - \log\det(X) + \text{tr}(XS)$$

where this is convex in X (this is not obvious but we will see it in the problem session).

If you know that the true matrix is sparse, you may add a penalty term $\lambda\|X\|_1$. Similarly in regression you may add either $\|x\|_1$ or $\|x\|_2^2$. In either case, these add curvature to your objective function. This may speed up the optimization process.

2.2.4 Unsupervised Learning

Finally we want to talk about *clustering*. Notice that we have been slowly changing our dataset. First we dropped labels and now we are going to represent our dataset as a graph $G = (V, E)$ of interactions. We would like to infer some partitioning of the vertices.

As a modelling assumption, we use the stochastic block model. There is a hidden partitioning $\sigma \in \{-1, 1\}^{|V|}$. G is generated from the distribution

$$(i,j) \in E \text{ with probability } \begin{cases} p & \sigma_i = \sigma j \\ q, \text{otherwise} \end{cases}$$

where we assume $q < p$. Furthermore we assume all connections are independent.

The the likelihood of $x \in \{-1, 1\}^n$ is given as follows: let $A = (a_{ij})$ be the adjacency matrix of G. Then the likelihood is

$$\prod_{ij}\left[\frac{1+x_ix_j}{2}\left(pa_{ij} + (1-p)(1-a_{ij}) + \frac{1-x_ix_j}{2}(qa_{ij}(1-q)(1-a_{ij}))\right)\right]$$

this is non-convex! But we can look at a convex upper bound and we're good! Next time we'll talk about gradient descent.

2.3 Gradient Descent

This is the most simple algorithm we will study (also one of the most used ones). It is an iterative method where we define

$$x_{t+1} = x_t - \eta\nabla f(x_t)$$

which dates back to the mid 1800's (likely by Cauchy). Notice that this is similar to Newton's method, but the latter requires inverting an $n \times n$ matrix and this was an attempt to avoid that difficult computation.

Why does this work? By Taylor's theorem, $f(y) \approx f(x) + \nabla f(x) \cdot (y - x)$. The idea here is that the negative $\nabla f(x)$ direction is in the direction of the minimum, so we just follow that.

Now if we have a convex function, where $f(x) - f(y) \leq \nabla f(x) \cdot (x - y)$, we can set $y = x^*$ and define

$$\delta(x) := f(x) = f(x^*) \leq \nabla f(x) \cdot (x^* - x) = -\nabla f(x) \cdot (x^* - x)$$

then the idea is that progress in the $x^* - x$ direction (that is towards the target) is bounded below by $\delta(x)$. That is to say: *the rate of decrease of the instance to OPT is lower bounded by the suboptimal gap.*

There is a small problem, namely that we also take a (hopefully small) step away from the direct-line path from x to x^*. To fix this, we use continuous time analysis:

$$\frac{d}{dt}x(t) = -\nabla f(x(t))$$

which we call the **gradient flow.** Then

$$\frac{d}{dt}\frac{1}{2}\|x(t) - x^*\|^2 = \left(\frac{d}{dt}(x(t) - x^*)\right) \cdot (x(t) = x^*) = -\nabla f(x(t)) \cdot (x(t) - x^*) \leq -\delta(x(t))$$

and integrating over time:

$$\int_0^\tau \frac{1}{2}\|x(t) - x^*\|^2 \mathrm{d}t \leq \int_0^\tau \delta(x(t))\mathrm{d}t$$

and using the FTOC and dropping the τ term (and normalizing):

$$\frac{1}{\tau}\int_0^\tau \delta(x(t))\mathrm{d}t \leq \frac{1}{2\tau}\|x(0) - x^*\|^2 := \frac{r_0^2}{2\tau}$$

and so finally

$$\delta\left(\frac{1}{\tau}\int_0^\tau x(t)\mathrm{d}t\right) \leq \frac{r_0^2}{2\tau}.$$

Of course this isn't a discrete process, so it's useless for coding! So we want to run gradient descent with step size η for T steps and approximate this integral with a finite sum from 1 to T. Then if we further impose that f is Lipshitz ($\|\nabla f(x)\| \leq L$) then the error in this approximation is at worst ηL^2. So if we are trying to measure the total error of our discretized algorithm from the true value, we get an error bound

$$\frac{r_0^2}{T\eta} + \eta L^2 \leq \frac{r_0 L}{\sqrt{T}}$$

for a "good" choice of $\eta = \frac{r_0}{L\sqrt{T}}$.

2.3.1 Discrete Time Analysis

2.3.1 Theorem

Let $g_1, \ldots, g_T$ be arbitrary vectors in $\mathbb{R}^n$ and that $x_{t+1} = x_t - \eta g_t$. Then for any $x \in \mathbb{R}^n$,

$$\sum_1^T g_t(x_t - x)le\frac{\|x_1 - x\|^2}{2\eta} + \eta\sum_1^T \|g_T\|^2$$

2.3.2 Corollary
If $g_t = \nabla f(x_t)$ (f convex) and $\|nablaf(x_t)\| \leq L$, then

$$f\left(\frac{1}{T}\sum_1^T x_t\right) - f(x^*) \leq \frac{1}{T}\sum_1^T (f(x_t) - f(x^*)) \leq \frac{1}{T}\sum \nabla f(x_t)(x_t - x^*) \leq \frac{r_n^2}{2\eta T} + \eta L^2$$

The proof here is really just one line of computation but I missed it. Interestingly, the proof gives us *equality* rather than inequality. I suppose that it's nice to use it as if it were a bound.

2.3.2 Projective Gradient Descent

A generalization is optimization with some constraint. So if we have a bounded convex set K, we just follow each step in gradient descent with a projection onto the closest point in K (should we leave it). This is only better for us since

2.3.3 Lemma
$\|P_K(x) - z\| \leq \|x - z\}$ for any $z \in K$.

2.3.4 Remark: Notice that we are assuming here that both K and f are convex. Without convexity we definitely don't have this.

2.4 Stochastic Gradient Descent

We begin with a corollary of the theorem we just proved for standard GD:

2.4.1 Corollary
Say that g_t is a random variable such that $\mathbb{E}[g_t|x_t] = \nabla x_t$ and that $\mathbb{E}[\|g_t\|^2] < B^2$. Then

$$\mathbb{E}[f\left(\frac{1}{T}\sum_1^T x_t\right) - f(x^*)] \leq \frac{r_1^2}{2\eta} + \frac{\eta}{2}TB^2 \leq \frac{r_1 B}{\sqrt{T}}$$

given an optimal choice of η.

So then we can use this result to define SGD, Stochastic Gradient Descent. Let $f(x) = \mathbb{E}_{(a,y)}l(x,(a,y))$. The idea here is that we don't have the actual gradient of the loss function, but we have access (under the statistical learning framework) to a continuous stream of IID data. So we define the **stochastic gradient**:

$$g_t = \nabla_x l(x_t, (a, t, y_t))$$

that is, compute the x gradient of $l(x,(a_t,y_t))$ and evaluate at x_t. Then basically this pos right into the above corollary.

2.4.2 Remark: here the notation varies from statistical learning, so think of x as our hypothesis (the thing we're optimizing over) and (a,y) denotes a sample from the dataset.

Note that here you only get one pass on the data (although here you can directly control the generalization error as opposed to bounding it with testing error in the other class).

Instead, we can do multi-pass SGD where at each step we sample uniformly data points from your dataset (possibly having much more steps than there are data points) but then your function converges to an unbiased estimator for the test error rather than the generalization error.

2.5 Optimality

Recall that we had two kind of bounds on expectation: $\frac{r_0 L}{\sqrt{T}}$ and $\frac{r_0 B}{\sqrt{T}}$ where $r_0 = \|x_0 - x^*\|_2$, $L = \sup_x \|\nabla f(x)\|$ and $B \geq \mathbb{E}[\|g_t\|^2]$ where g_t is an unbiased estimator for $\nabla f(x_t)$, which we think of as a random oracle.

2.5.1 Basic minimax principle

We want to show that for all (possible randomized) algorithms such that there exists a function/oracle such that the error rate of gradient descent is optimal. It suffice to find a **random** function such that any deterministic algorithm on this function has error rate bigger than $\frac{\square}{\sqrt{T}}$ in expectation.

This is a consequence of the fact that $\min\max \geq \max\min$ (these are read in the opposite order than you'd think).

2.5.2 Random oracle lower bound

Let $K = [-1, 1]$ and $f(x) = \xi x$ where $\xi = \pm\varepsilon$ randomly with equal probability. Then our random oracle is $g_t \sim \mathcal{N}(\xi, 1)$ (equivalently $g_t = \xi + Z_t$ for Z_i IID $\mathcal{N}(0, 1)$). Intuitively, after T queries the best estimator for ξ is the empirical mean $\frac{1}{T}\sum_1^T g_t \sim \mathcal{N}(\xi, \frac{1}{T}) = \xi + \frac{1}{\sqrt{T}} Z$ where Z is $\mathcal{N}(0, 1)$.

So if $\varepsilon < \frac{1}{\sqrt{T}}$, there is a constant probability to incorrectly estimate the sign of ξ. Formally,

2.5.1 Lemma
For any $\Psi : \mathbb{R}^T \to [-1, 1]$,

$$\mathbb{P}(\text{sign}(\Psi(g_1, \ldots, g_T)) = \text{sign}(\xi)) \leq \frac{1}{2} + \sqrt{T\varepsilon^2}$$

2.5.3 Deterministic oracle model

let $v_1, \ldots, v_{T+1} \in \mathbb{R}^n$ be a random orthonormal set. Let $f(x) = \max_{i \in [T+1]} v_i \cdot x$. We also let $K = \{x \in \mathbb{R}^n : \|x\|_2 = 1\}$. Then $\nabla f(x) = v_i$ for $i = \text{argmax}_j\, v_j \cdot x$ (except for a set of measure zero, where we can use subgradients).

After T queries, one of the v_i remains unknown, say $T + 1$. So for any $x, v_1, \ldots, v_T$,

$$\mathbb{E}_f[f(x)|v_1, \ldots, v_T] \geq \mathbb{E}_f[v_{T+1} \cdot x]|v_1, \ldots, v_T] = 0$$

And so $f^* = \min_{x \in B} f(x) \leq -\frac{1}{\sqrt{T+1}}$.

2.5.2 Remark: It is very important that $T < n$, since we are looking for an orthonormal set! But the bound is not dependent on the dimension! So although we got a similar lower bound on T, we still may be able to do better when we have lower dimension.

To see this, we need

2.5.3 Proposition (Grünbaum's Inequality)
Let K be a convex body. For any half-space H that contains the center of gravity of K. Then

$$\text{vol}(K] \cap M) \geq \frac{1}{e}\,\text{vol}(K).$$

How do we get half spaces by making gradient queries? Well by convexity, $f(x) \geq f(x_t) + \nabla f(x_t)(x - x_t) \geq f(x_t)$ for any $x \in H_t = \{x : \nabla(x_t) \cdot (x - x_t) \geq 0\}$.

Then we have the **center of gravity algorithm**: let $K_0 = K$ and for any $T \geq 0$, $x_t = cg(K_t)$ and set $K_{t+1} = K_t \setminus H_t$. Then by Grünbaum, $\text{vol}(K_t) \leq (1 - \frac{1}{e})\,\text{vol}(K_{t-1}) \leq (1 - \frac{1}{e})^t\,\text{vol}(K)$

2.5.4 Theorem
Say $f : K \to [-1, 1]$. After $\mathcal{O}(n \log(1/\varepsilon))$ steps, $f(x_t) - f(x^*) \leq \varepsilon$.

Proof
After this many steps, $\text{vol}(K_t) \leq \varepsilon^n\,\text{vol}(K)$. We claim $\exists \Omega \subseteq K$ such that $\forall x \in \Omega$, $f(x) - f(x^*) \leq \varepsilon$ and $\text{vol}(\Omega) \geq \varepsilon^n$ ♠

2.5.4 Beyond $\frac{1}{\sqrt{T}}$ in high dimensions

The best way to get around the lower bounds we've cooked up for ourselves is to put some further restrictions on our functions:

2.5.5 Definition: A function f is **α-strongly convex** if $f(x) = g(x) + \frac{\alpha}{2}\|x\|_2^2$, where g is convex.

2.5.6 Remark: This property is equivalent with the fact that

$$f(x) - f(y) \leq \nabla f(x) \cdot (x - y) - \frac{\alpha}{2}\|x - y\|_2^2.$$

So we have that f curves away from its tangent at that point like a parabola with coefficient α.

Let's do the continuous-time analysis of the gradient flow $\frac{d}{dt}x(t) = -\nabla f(x(t))$. Let $r^2(t) = \|x(t) - x^*\|^2$ and $\delta(t) = f(x(t)) - f(x^*)$. Then the calculation we get is

$$\frac{d}{dt}r^2(t) = -\nabla f(x(t)) \cdot (x(t) - x^*)$$

$$\leq -\delta(t) - \frac{\alpha}{2}\|x(t) - x^*\|_2^2.$$

Thus $\frac{d}{dt}r^2(t) \leq -\alpha r^2(t)$ and so $r^2(t) \leq r^2(0)e^{-\alpha t}$. All this to say that gradient flow gets to OPT in time $\mathcal{O}(\frac{1}{\alpha}\log(r_0^2/\varepsilon))$. By comparison with out strong convexity, we had that

$$\frac{1}{\tau}\int_0^\tau \delta(t)\mathrm{d}t \leq \frac{r_0^2(0)}{\tau}$$

and so our time is $\mathcal{O}(r_0(0)/\varepsilon)$.

By switching from gradient flow to gradient descent, we pick up a discretization error! let ηL^2 be the error from gradient flow at $\tau = T\eta$. Then through some analysis, we get a $\widetilde{\mathcal{O}}(L^2/\alpha T)$ rate.

Next we can talk about **smoothness**.

2.5.7 Definition: A function f is β-smooth if ∇f is β-Lipshitz.

Intuitively, we are looking to improve the discretization error. Before we could overshoot our maximum easily because the gradient could change sharply. Now, we are forcing it to tend (loosely) to zero as you get closer to the optimum. Notice

$$\|\nabla f(x + \eta\nabla f(x)) - \nabla f(x)\| \leq \beta\eta\|\nabla f(x)\|$$

which suggests that $\eta \propto \frac{1}{\beta}$ is fine for the discretization. In this case we get convergence like $r^2(0)\beta/T$ (roughly $1/\varepsilon$) and if we combine both α-strong convexity and β-smoothness, you get a parameter

$$\kappa = \frac{\beta}{\alpha}$$

called the **Condition number** and we get growth like $\kappa\log(1/\varepsilon)$

2.5.5 A formal inequality

2.5.8 Proposition
Let $x^+ = x - \frac{1}{\beta}\nabla f(x)$. Then

$$f(x^+) - f(x) \leq \frac{-1}{2\beta}\|\nabla f(x)\|_2^2.$$

Proof
We begin by claiming

$$f(x) - f(y) \leq \nabla f(y)\cdot(x - y) + \frac{\beta}{2}\|x - y\|_2^2$$

which implies that

$$f(x^+) - f(x) \leq \nabla f(x) \cdot (x^+ - x) + \frac{\beta}{2}\|x^+ - x\|_2^2$$

and unfolding this

$$f(x^+) - f(x) \leq -\frac{1}{2\beta}\|\nabla f(x)\|^2.$$

♠

2.5.9 Definition: A point x is called **ε-stationary point of f** if $\|\nabla f(x)\| \leq \varepsilon$.

2.5.10 Remark: The above inequality shows that gradient descent on a bounded, smooth, non-convex function finds an ε-stationary point in $1/\varepsilon^2$ steps.

Example 2.1
Let $f(x) = \frac{1}{2m}\|Ax - b\|^2$ where A is $m \times m$. Then the Hessian is

$$\nabla^2 f(x) = \frac{1}{m}A^T A = \frac{1}{m}\sum_1^m a_i a_i^T$$

where the a_i should be thought of as feature vectors. Then the above quantity is the sample covariance matrix of the features.

In this case, $\alpha = \lambda_{min}(\frac{1}{m}A^T A$ and $\beta = \lambda_{max}(\frac{1}{m}A^T A)$, so if $\|a_i\|_2 \leq 1$ then $\beta \leq 1$.

2.5.11 Remark: It is natural to ask how we can get a better value of β. One way is to convolve your function f with some noise z: $\hat{f}(x) = f(x + z)$.

Sometimes f is nonsmooth, but can be written as $f = g + r$ where g is smooth and r is "simple". Typically one takes $r(x) = \|x\|_1$.

2.6 Acceleration

It turns out that $1/\sqrt{\varepsilon}$ (for smooth) and $\sqrt{\kappa}\log(1/\varepsilon)$ for smooth and strongly convex functions are achievable!

What could be better than gradient descent? Well here's a crazy idea:

$$x_{t+1} = \underset{\text{span}(\nabla f(x_0),\ldots,\nabla f(x_t))}{\text{argmin}} [f(x)]$$

where $[f(x)] = \sum_{s=1}^t \lambda_s \nabla f(x_s)$. THen if f is quadratic, it turns out that λ_s has a closed form formula!

2.6.1 Nemiraski's Acceleration (1982)

Let

$$x_{t+1} = \operatorname*{argmin}_{x \in P_t} f(x)$$

where $P_t = \text{span}(x_t^+, \sum_1^t \lambda_s \nabla f(x_s))$.

Then doing some analysis, notice

$$\sum_1^T \lambda_s \delta_s \leq \sum_1^T \lambda_s \nabla f(x_s) \cdot (x_s - x^*) = \sum_1^T \lambda_s \nabla f(x_s) \cdot (-x^*)$$

where the last equality follows since $x_s \in P_{s-1}$ implies that $\nabla f(x_s) \in P_{s-2}^\perp$. Thus

$$\|x^*\| \cdot \sqrt{\sum_1^t \lambda_s^2 \|\nabla f(x_s)\|^2}$$

since $\nabla f(x_{T+1}) \perp \sum_1^T \lambda_s \nabla f(x_s)$. But then by smoothness,

$$\frac{1}{2beta} \|\nabla f(x_s)\|^2 \leq \delta_s - \delta_{s+1}$$

and so

$$\sum_1^R \lambda_s \delta_s \leq \|x^*\| \cdot \sqrt{2\beta \sum_1^T \lambda_s^2 (\delta_s - \delta_{s+1})}.$$

Now pick some λ_s such that $2\beta(\lambda_s^2 - \lambda_{s-1}^2) = \lambda_s$ and after some juggling things,

$$\delta_T \leq \frac{\beta \|x^*\|^2}{T^2}$$

and so we get an algorithm that runs in time like $1/\sqrt{\varepsilon}$.

2.6.2 Nesterov's AGD (1983)

The key breakthrough in this algorithm is that all that matters about the past is the momentum.

2.6.1 Definition: The **momentum** if an iterative algorithm is

$$d_t := \gamma_t (x_t - x_{t-1}).$$

The began with Polyak's *heavy ball method.* Let $x_{t+1} = x_t^+ + d_t$. This method alone doesn't accelerate (can be seen on simple quadratic functions). Instead, Nestrov's uses

$$x_{t+1} = (x_t + d_t)^+.$$

Now for some of the analysis. Unfortuately this is notoriously "magical", so hold onto your pants. Denote by $g_t = -\frac{1}{\beta}\nabla f(x_t + d_t)$. Thus $x_{t+1} = x_t + d_t + g_t$. Then

$$\delta_{t+1} - \delta_t = f(x_t + d_t)^+) - f(x_t) \leq f(x_t + d_t) = \frac{1}{2\beta}\|\nabla f(x_t + d_t)\|_2^2 - f(x_t)$$

by the smoothness of f. Then by convexity,

$$\delta_{t+1} - \delta_t \leq \nabla f(x_t + d_t) \cdot d_t - \frac{1}{2\beta}\|\nabla f(x_t + d_t)\|_2^2 = -\beta g_t \cdot d_t - \frac{\beta}{2}\|g_t\|_2^2.$$

But notice that

$$\delta_{t+1} = f((x_t + d_t)^+) - f(x^*) \leq= \beta g_t \cdot (x_t + d_t - x^*) - \frac{\beta}{2}\|g_t\|_2^2.$$

And the plan is to take a combination of these two statements to get a weighted sum of the δ_i on the left and a telescopic sum on the right. It wasn't that bad. Basically just put in some parameters and then choose them to be the "right ones". THe punchline is approximate convergence like $\delta_{T+1} \leq \frac{\beta\|x_1 - x^*\|^2}{T^2}$.

2.7 Some variance reduction techniques

2.7.1 The finite sum problem

Let's say we have a function $f(x) = \frac{1}{m}\sum_i f_i(x)$ where f_i is β-smooth (note that in particular this implies f is β-smooth) and f is α-strongly convex. Then stochastic gradient descent does $x_{t+1} = x_t - \eta\nabla f_{i_t}(x_t)$ which runs like $1/\alpha\varepsilon$ (the number of "data operations").

In regular gradient descent, $x_{t+1} = x_t - \frac{\eta}{m}\sum_1^m \nabla f_i(x_t)$, which runs with $\kappa\log(1/\varepsilon)$ data operations. The differnce here is that SGD has a lot of variance in its estimators, so to be "safe" you have to take smaller steps. On the other hand, GD needs much more data but can run with better accuracy. So the idea here is to somehow control the variance.

The key idea is that we, from time, to time, compute a full gradient at some point y.

$$g_t := \nabla f_{i_t}(x_t) - \nabla f_{i_t} + \nabla f(y)$$

We can see that g_t is un unbiased estimator of $\nabla f_{i_t}(x_t)$ with the added bonus that the variance $\mathbb{E}[\|g_t\|^2]$ gets smaller when close to the optimum.

2.7.1 Theorem

During an epoch T, let $x_1 \leftarrow y$ and for $t \geq 2$, $x_{t+1} = x_t - \eta g_t$. Then set y to be $\frac{1}{T}\sum_1^T x_t$ and recompute $\nabla f(y)$. Then by using $\eta = \frac{1}{10\beta}$,

$$f(\text{new } y) - f(x^*) \leq 0.9(f(y) - f(x^*)).$$

2.8 Further Topics

- We talked earlier about decomposing functions into smooth and non-smooth (simple) parts.
- Mini-batch SGD: write $x_{t+1} = x_t - \frac{\eta}{|B|}\sum_B \nabla f_i(x)$ where B is a batch. Then we get similar results for convergence up to $b = \sqrt{m}$.
- Bandit convex optimization
- Interior point method: tells you how to handle complicated K.
- Constant step size least squares: no $1/\alpha$ in the bound.
- Mirror descent: the equation $x - \nabla f(x)$ doesn't actually make sense (one is a column, one is a row). Instead we pick a function ϕ and look at

$$\nabla\phi^{-1}(\nabla\phi(x) - \nabla f(x)).$$

3 Bandits

This series was given by Kevin Jamieson, a professor in the CSE department here at UW.

3.1 Introduction

The core idea here (motivated very well by his description of the WSJ front page (ads, content, etc) and google maps (that does bandit-like testing to find new good routes) that balances the exploration (data-gathering) phase and the application phase and the transiton between them. It more resembles a conversation than a one-time transaction.

The name comes from slot machines, obviously. :) We are thinking of these problems as having a slot machine with n "arms", which can be pulled to yield some data. So the input to the algorithm is the number n and for $t = 1, 2, \ldots$, the algorithm pulls some arm $I_t \in [n]$ and nature reveals a corresponding reward $r_t \sim P_{I_t}$ where $\mathbb{E}[r_t|I_t] = \theta^*_{I_t}$.

3.1.1 Regret Minimization

We define the **regret** as follows:

3.1.1 Definition: After T timesteps, the regret is

$$R_T = \max_i \theta_i^* T - \mathbb{E}\left[\sum_1^T r_t\right]$$

Then the goal is to achieve "sublinear regret", that is get that $R_T \in \mathbf{o}(T)$.

3.1.2 Lemma

$$\begin{aligned} R_T &= \max_i \theta_i^* T - \mathbb{E}[\sum_1^T \sum_1^n r_t \delta_{I_t=i}] \\ &= \max \theta_i^* T - \sum_1^n \mathbb{E}[\sum +1^T \theta_i^* \delta_{I_t=i}] \\ &= \max \theta_i^* T - \sum_1^n \theta_i^* \mathbb{E}[T_i] \\ &= \sum_2^n (\theta_1^* - \theta_i^*) \mathbb{E}[T_i] \\ &= \sum_2^n \Delta_i \mathbb{E}[T_i]. \end{aligned}$$

3.2 Another game

This leads us to the concept of **best-arm identification**: we fix some $\delta \in (0,1)$ and we identify $\operatorname{argmax}_i \theta_i^*$ with probability $1 = \delta$ with as few as possible total pulls. This time we don't put an upper bound on the time you're given, and instead your job is to identify what the best strategy is.

3.2.1 Definition: A random variable X is **R-sub-Gaussian** if

$$\mathbb{E}[\exp(\lambda(x))] \leq \exp(\lambda R^2/2)$$

3.2.2 Remark: Then by Hoeffding, if $X \in [a,b]$ and letting $R^2 = \frac{(b-a)^2}{8}$ and notice when $X \sim \mathcal{N}(0,1)$, we get $R^2 = 1$.

Then we can develop the Chernoff bound: Suppose X_i are IID and 1-sub-Gaussian. Then

$$\begin{aligned} \mathbb{P}(\sum_1^n X_i > \varepsilon) &= \mathbb{P}(\exp(\lambda \sum X_i) > \exp(\lambda\varepsilon)) \\ &\leq e^{-\lambda\varepsilon} \mathbb{E}[\exp(\lambda \sum X_i)] \\ &= e^{-\lambda\varepsilon} \prod \mathbb{E}[e^{\lambda X_i}] \\ &\leq e^{-\lambda\varepsilon} (e^{\lambda^2/2})^n \end{aligned}$$

$$\leq e^{-n\varepsilon^2/2}$$

where the last step "comes from calculus". So by letting $\delta = e^{-n\varepsilon^2/2}$, one can solve for epsilon and get that with probability $1-\delta$, $\frac{1}{n}\sum X_i \leq \sqrt{\frac{2\ln 1/\delta}{n}}$.

If $n=2$, say $\theta_1^*\theta_2^*$ and $\Delta = \theta_1^* - \theta_2^*$. If both arms are pulled τ times, let the event

$$\mathcal{E}_i = \left\{ \left| \theta_i^* - \frac{1}{\tau}\sum_{t=1}^{\tau} X_{i,t} \right| \leq \sqrt{\frac{2\ln(4/\delta)}{\tau}} \right\}$$

for $i = 1, 2$ (these are the "good events"), then $\mathbb{P}(\mathcal{E}_i^C) \leq \delta/2$, so

$$\mathbb{P}(\mathcal{E}_1^C \cup \mathcal{E}_2^C) \leq \sum \mathbb{P}(\mathcal{E}_i^C) \leq \delta.$$

Then if $\tau = 8\Delta^{-2}\ln(4/\delta)$, we can say

$$\hat{\theta}_2 - \hat{\theta}_2 = \hat{\theta}^1 - \theta_1^* - \hat{\theta}_2 + \theta_2^* + \theta_1^* - \theta_2^* > 0$$

with probability $1-\delta$. This of course means that we have accurately determined which arm to pull.

Now consider the regret:

$$R_T = \theta_1^* t = \mathbb{E}[\theta_1^* T_1 + \theta_2^* T_2] = (\theta_1^* - \theta_2^*)\mathbb{E}[T_2] = \Delta\mathbb{E}[T_2]$$

from which we can compute

$$\Delta\mathbb{E}[T_2] = \Delta\mathbb{E}[T_2\delta_{\mathcal{E}_1\cap\mathcal{E}_2} + T_2\delta_{\mathcal{E}_1^C\cup\mathcal{E}_2^C}] \leq \Delta\tau + \Delta T\mathbb{P}[\mathcal{E}_1^C \cup \mathcal{E}_2^C]$$

and so if we set $\delta = \frac{1}{T}$,

$$R_T \leq 8\Delta^{-1}\log(8T) + \Delta$$

But notice that if we just always pick the bad arm, the worst regret we can get is $R_T \leq \Delta T$. So

$$R_T \leq 1 + \min 8\Delta^{-1}\log(8T), \Delta T \leq 1 + \sqrt{8T\ln(8T)}$$

If you were to graph the minimum function we had there, you'd notice that the regret tends towards zero near zero and towards infinity. The idea here is that if there is only a small difference between the optimal and suboptimal choice, then guessing randomly will only incur a tiny regret. Likewise, if the means are very far apart, you will quickly detect this through testing. There is a spike at about $1/\sqrt{T}$ that is the place where both ideas miss the mark.

3.3 Extending to more Arms

We are going to start by writing down an algorithm and then trying to analyze it:

3.3.1 Algorithm

We input n arms and confidence δ. Initialize $\hat{X}_1 = \{1, \dots, n\}$. Then for $l = 1, 2, \dots$ we let $\mathcal{E}_l = 2^{-l}$ and

$$\tau_l = \left\lceil 2\mathcal{E}_l^{-2} \ln \frac{4l^2 n}{\delta} \right\rceil$$

Then pull each arm in $\hat{X}_l$ τ_l times and compute $\hat{\theta}_{i,l}$. Then define

$$\hat{X}_{l+1} = \hat{X}_l \setminus \left\{ i \in \hat{X}_l : \max_{j \in \hat{X}_l} \hat{\theta}_{j,l} - \hat{\theta}_{i,l} > 2\varepsilon_l \right\}$$

3.3.2 Analysis

Define $\mathcal{E}_{i,l} = \{|\hat{\theta}_{i,l} - \theta_i^*| \leq \varepsilon_l\}$ and let $\mathcal{E} = \cap_{i,l} \mathcal{E}_{i,l}$. But then

$$\mathbb{P}(\mathcal{E}_{i,l}^C) \leq 2\mathbb{P}(\hat{\theta}_{i,l} - \theta_i^* > \varepsilon_l) \leq 2 \exp -\tau_l \varepsilon_l^2/2) \leq \frac{\delta}{2l^2 n}$$

and so $\mathbb{P}(\mathcal{E}^c) \leq \delta$.

3.3.1 Definition: Let $\Delta_i = |\theta_i^* - \theta_1^*|$ for all $i > 1$ and define $\Delta_1 = \Delta_2$ (the minimum (nonzero) gap), where we assume for notational simplicity that $1 \geq \theta_1^* > \theta_2^* \geq \cdots \geq \theta_n^* \geq 0$.

3.3.2 Lemma
With probability greater or equal to $1 - \delta$, we have $1 \in \hat{X}_l$ and

$$\max_{i \in \hat{X}_l} \Delta_i \leq 8\varepsilon_l$$

for all l.

3.3.3 Remark: The key thing to notice in the proof here is the fact that for any $j \in \hat{X}_l$,

$$\hat{\theta}_{j,l} - \hat{\theta}_{1,l} = (\hat{\theta}_{j,l} - \theta_j^*) - (\hat{\theta}_{1,l} - \theta_1^*) - \Delta_j \leq 2\varepsilon_l - \Delta_j \leq 2\varepsilon_l$$

Now let's talk about regret:

$$R_T = \mathbb{E}\left[\sum_1^n \Delta_i T_i \right]$$

and so for any $\nu > 0$,

$$\sum_1^n \Delta_i T_i \leq \nu T + \sum_{l : 8\varepsilon_l \geq \nu} 8\varepsilon_l \tau_l |\hat{X}_l| \leq \nu T + \sum_{l-1}^{\lceil \lg \max(\Delta, \nu)^{-1} \rceil} 8\varepsilon_l \tau_l \delta_{\Delta_i \leq 8\varepsilon_l}$$

and then doing some more massaging, we get

$$\sum \Delta_i T_i \leq \nu T + \sum_1^n c(\max(\Delta_i, \nu)^{-1}) \log\left(\frac{n \log(\max(\Delta_i, \nu)^{-1})}{\delta} \right)$$

3.3.4 Remark: Notice there are some serious limitations here. In particular, it requires prior knowledge of T, which we don't always have.

The previous algorithm we described (which one?) we will denote AE1, and we ended up with regret

$$R_T \leq \sqrt{cnT\log(nT)}$$

and

$$R_T \leq c\sum_2^n \Delta_i^{-1}\log(T)$$

One can show that with probability $\geq 1-\delta$, AE1 give us that $\hat{X}_l = \{1\}$ after at most

$$\tau = \sum_2^n c\Delta_i^{-2}\log\left(\frac{n\log(\Delta_i^{-2})}{\delta}\right)$$

pulls.

3.4 Theoretical Bounds

Then the question one can ask is whether the bounds given are *tight*. A information-theoretic result we will see in the problem session.

3.4.1 Proposition
Suppose n samples are taken from $\mathcal{N}(\mu,1)$ where $\mu \in \{0,\Delta\}$. To identify if $\mu = 0$ or Δ with probability $1-\delta$, one requires $n \geq 2\Delta^{-2}\log(1/4\delta)$.

This really hints how difficult the multi-arm bandit problem can be.

3.4.2 Theorem
Any δ-PAC on $\{P : P_i = \mathcal{N}(\theta_i,1), \theta \in [0,1]^n\}$ satisfies

$$\mathbb{E}[\tau|\theta^*] \geq 2\log(1/2\delta)\sum_2^n \Delta_i^{-2}$$

3.4.3 Remark: This is a result on expectation rather than a high-probability bound, obviously, but under these conditions one can turn this into a lower-bound in the high-probability case as well (after losing a big constant factor). Going in the other direction is tricky, because there are bad cases in which this algorithm never terminates or outputs the right thing, so the expectation goes up a bit.

3.4.4 Remark: If we take the partial sums $S_T = \sum_{t=1}^T Z_t$ of your random variables, you get the *law of the iterated logarithm*, which says

$$\limsup_T \frac{S_T}{\sqrt{2T\log\log T}} = 1$$

so the $\log\log$ factor in the bound we found is actually tight.

We also get:

3.4.5 Theorem

Any algorithm satisfying $\mathbb{E}[T_i] \leq \mathbf{o}(T^\alpha)$ for all $i \neq 1$ (suboptimal samples) any $\alpha \in (0,1)$ suffers

$$\lim_{T\to\infty} \frac{R_T}{\log(T)} = \sum_2^n \frac{2}{\Delta_i}$$

and a worst-case result:

3.4.6 Theorem

For any n and T, there exists an instance (θ^*) such that

$$R_T \geq \sqrt{(n-1)T}/27.$$

3.5 Upper Confidence Bound

An algorithm that achieves the bound in thm 3.4.5 is as follows. For $t = 1, 2, \ldots,$ pull $\operatorname{argmax}_i \hat{\theta}_{c,T_i(t)} + \sqrt{\frac{2\log(t)}{T_i(t)}}$ where $T_i(t)$ is the number of pulls up to t.

This leads to a minimization of regret, but notice that this algorithm is suboptimal on the problem of identifying the best arm. But Kevin created an alteration called "Lil' UCB" that proceeds similarly but adds in an extra $\frac{1}{\delta}$ factor (as well as some constants) to achieve optimal best arm identification.

These algorithms have this phenomenon where, as time goes on, the likelihood of being selected gradually increases, but then each pull reduces it somewhat. Also notice that this algorithm is very much not optimal for an overall ranking since it will stop pulling arms that have a bad run.

3.6 Linear Bandits

Notice that so far we still have a dependence on n, but if you are running (e.g.) a music recommendation software, you likely have millions of data points to sample. Furthermore, songs have tons of features to compare, so we would like to be able to exploit the fact that a person is likely to enjoy music that is "close" under some sort of metric (using genre, BPM, etc).

So we develop a **linear bandits:** For $t = 1, 2, \ldots,$ we pull arm I_t and nature reveals $\langle x_{I_t}, \theta^* \rangle + \xi_t$. Here we say $\xi_t \sim \mathcal{N}(0,1)$. We take as known $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ and $\theta^* \in \mathbb{R}^d$ is unknown. We define the best arm to be $x^* = \operatorname{argmax}_{x\in X} \langle x, \theta^* \rangle$.

Say we observe some data $\{(x_i, y_i)\}_1^\tau$ where $y_i = \langle x_i, \theta^* \rangle + \xi_i$. Then setting

$$\hat{\theta} = \operatorname*{argmin}_{\theta\in\mathbb{R}^d} \sum_1^\tau (\langle x_i, \theta \rangle - y_i)^2$$

and then if $X = (x_i^T)^T$ and $Y = (y_i) = X\theta^* + \xi$, then

$$\hat{\theta} = (X^T X)^{-1} X^T Y.$$

Then unfolding Y, we get

$$\hat{\theta} = \theta^* + (X^T X)^{-1} X^T \xi$$

thus $\mathbb{E}[\hat{\theta}] = \theta^*$. Computing the standard deviation:

$$\mathbb{E}[(\hat{\theta} - \theta^*)(\hat{\theta} - \theta^*)] = (X^T X)^{-1}$$

so $\hat{\theta} - \theta^* \sim \mathcal{N}(0, (X^T X)^{-1})$.

Importantly, notice that the variance of $\hat{\theta} - \theta^*$ depends only on X so we can "plan ahead" to control how good we want our estimator $\hat{\theta}$ to be!

Now notice that for any $z \in \mathbb{R}^d$, we have $\langle z, \hat{\theta} - \theta^* \rangle \sim \mathcal{N}(0, z^T (X^T X)^{-1} z)$. Then if we just call $\sigma^2 = z^T (X^T X)^{-1} z$, then

$$\mathbb{P}\left(\langle z, \hat{\theta} - \theta^* \rangle > \sqrt{2\sigma^2 \log(1/\delta)} \right) \leq \delta.$$

Writing this another way, with probability $1 - \delta$,

$$|\langle z, \hat{\theta} - \theta^* \rangle| \leq \sqrt{z^T (X^T X)^{-1} z} \cdot \sqrt{2 \log(2/\delta)} = \|z\|_{(X^T X)^{-1}} \sqrt{2 \log(2/\delta)}$$

where, as a matter of notation, we write $x^T A x = \|x\|_A^2$.

Somehow the idea here is that you may have data lining up along a certain line, in which case your confidence interval along the line in question goes down.

For all X, there exists $\lambda \in \Delta_x$ such that

$$\tau \sum_{x \in X} \lambda_x x x^T = X^T X$$

and so if we set $A_\lambda = \sum \lambda_x x x^T$, we get a bunch of optimization problems:

- E-optimal
$$\min_\lambda \max_{\|u\|_2 \leq 1} u^T A_\lambda^{-1} u$$
- A-optimal
$$\min_\lambda \operatorname{Tr}(A_\lambda^{-1})$$
- G-optimal
$$\min_\lambda \max_{x \in X} x^T A_\lambda^{-1} x$$
- D-optimal
$$\max_\lambda \log \det(A_\lambda)$$

3.6.1 Theorem (Kiefer-Wolfowitz)
For any $X \in \mathbb{R}^d$ (that spans), there is $\lambda^* \in \Delta_X$ such that

(a) $\max_\lambda g_D(\lambda) = g_D(\lambda^*)$

(b) $\min_\lambda f_G(\lambda) = f_G(\lambda^*)$

(c) $\text{support}(\lambda^*) \leq \frac{(d+1)d}{2}$

(d) $f_G(\lambda^*) = g_D(\lambda^*) = d$

where $g_D(\lambda) = \log\det(A_\lambda)$ and $f_G(\lambda) = \max_{x\in X} x^T A_\lambda^{-1} x$.

3.6.2 Proposition
If λ^* is the $\frac{(d+1)d}{2}$-sparse G-optimal solution and x is pulled $\lceil \lambda_x \tau \rceil$ and we compute $\hat{\theta}$ (using at most $\frac{(d+1)d}{2} + \tau$ pulls) then with probability $\geq 1-\delta$, for any fixed $x \in X$,

$$\langle x, \hat{\theta} - \theta^* \rangle \leq \sqrt{\frac{2d\log(1/\delta)}{\tau}}$$

3.7 AE revisited

Recall AE1: for $t = 1, \ldots,$ $\hat{\lambda}_l$ is G-optimal for $\hat{X}_l$ and $\varepsilon_l = 2^{-l}$ and $\tau_l = 2d\varepsilon_l^{-2}\log\left(\frac{4l^2|x|}{\delta}\right)$. Then pull arm x $\lceil \tau_l \hat{\lambda}_l \rceil$ times and construct $\hat{\theta}_l$. Then let

$$\hat{X}_{l+1} = \hat{X}_l \setminus \{x \in \hat{X}_l : \max_{x'\in X} \langle x' - x, \hat{\theta}_l \rangle > 2\varepsilon_l\}$$

then $\mathbb{P}(\cup_l \cup_{x\in X} \mathcal{E}_{x,l}^c) \leq \delta$ where $\mathcal{E}_{x,l}$ is the event that $\langle x, \hat{\theta}_l - \theta^* \rangle \leq \varepsilon_l$.

Going through basically the entire argument from before,

$$R_T \leq c\sqrt{dT\log(|X|T)}.$$

3.7.1 Remark: Notice that part of what we wanted in all this was to get rid of dependence on the sides of in the inputs, $|X|$. The upshot is, in many (all?) cases, you can just a kinds of ε-cover of your space, which means we can replace it with a factor of d.

3.8 Contextual Bandits

Another game we can play: for $t = 1, \ldots$ Nature reveals context $c_t \sim D$ (IID). Then the player chooses $x_t \in X$ and nature reveals the reward (with noise)

$$r_t = v(v_t, x_t) + \xi_t$$

where X is known and we know $\mathbb{E}[\xi] = 0$.

3.8.1 Linear Contextual Bandits

One can actually capture this in the linear bandit framework. Assume there exists a feater map $\Phi : X \times X \to \mathbb{R}^d$ and $v(c,x) = (\Phi(c,x), \theta^*)$ for some unknown θ^*. For example, if $x \in \mathbb{R}^p$ and $c \in \mathbb{R}^q$, we could have $\Phi(c,x) = \text{vec}(xc^T) \in \mathbb{R}^{pq}$.

3.8.1 Remark: Often one uses as Φ a deep network. One can train a deep network and then take out the last layer and use it to assign features to your data.

3.8.2 General Policies

A policy is a map $\pi : \mathcal{C} \to X$, that is it assigns an action to each observed context. Then the **value** $V(\pi)$ is

$$V(\pi) = \mathbb{E}_{c\sim D}[v(c, \pi(c)) + \xi].$$

To define regret, we say

$$R_T = T \cdot v(\pi^*) - \mathbb{E}\left[\sum_1^T v(\pi_t)\right]$$

where $\pi^* = \text{argmax}_{\pi\in\Pi} v(\pi)$.

3.8.3 Inverse Propensity Scores

We define an estimator for $v(\pi)$ no matter what π we have. Suppose we have a distribution $\mu(x|c) \in \mathbb{S}^{|X|}$. Then we play action x_t distributed as $\mu(x|c_t)$ for all t. Build the estimator $\hat{v}(c_t, x) = r_t \frac{\delta_{x_t=x}}{\mu(x|c_t)}$. Then we define the estimator

$$\hat{v}_t(\pi) = \frac{1}{t}\sum_{s=1}^t \hat{v}(c_s, \pi(c_s)).$$

Notice that

$$\begin{aligned}
\mathbb{E}[\hat{v}_t(c_t,x)|c_t] &= \sum_{x'\in X} \mu(x'|c_t)\mathbb{E}\left[r_t \frac{\delta_{x_t=x}}{\mu(x|c_t)}\middle| c_t, x_t = x'\right] \\
&= \sum_{x'\in X} \mu(x'|c)v(c_t,x')\frac{\delta_{x_t=x}}{\mu(x|c_t)} \\
&= v(c_t,x)
\end{aligned}$$

So we get that $\hat{v}(c_t|x)$ is unbiased, but could suffer from very large variance. For instance it is zero unless $x = x_t$! Then one can show $\mathbb{E}[\hat{v}_t(\pi)] = v(\pi)$.

3.8.2 Remark: To follow this process, you need to record as data $\{(c_t, x_t, r_t, \mu(c_t, x_t))\}$. Most important is the distribution μ! This is a common mistake in practice but is useless without this data.

To compute the variance:

$$\mathbb{E}[(\hat{v}_t(c_x, x) - \mathbb{E}[\cdot])^2 | c_t] \leq \frac{1}{\mu(x, c_t)}.$$

Now let $Q \in \mathbb{S}^\pi$ be a distribution over policies. Then we are going to let

$$\mu(x|c) = \sum_{\pi \in \Pi : \pi(c) = x} Q(\pi).$$

Then at round l, we choose

$$\hat{Q}_l = \operatorname*{argmin}_{Q \in \mathbb{S}^{|\hat{\Pi}_l|}} \max_{\pi \in \hat{\Pi}_l} \mathbb{E}[(\hat{v}_t(\pi) - v(\pi))^2]$$

Using this scheme, we can get a regret bound that looks like

$$R_T \leq \sqrt{T|x| \log(|\pi|T)}$$

3.9 Introduction to Adversarial Bandits

In this scheme, an adversary chooses $x \in [0,1]^{n \times T}$. Then for $t = 1, 2, \ldots,$ the player chooses $I_t \in [n]$, and receives reward $x_{I_t,t}$. Then we set the regret to be

$$R_T = \max_{i \in [n]} \sum_1^T x_{i,t} - \mathbb{E}\left[\sum_1^t x_{I_t,t}\right]$$

The twist here is that your adversary will be able to choose x *with full knowledge of your algorithm.* Notice that any deterministic algorithm can be tricked into achieving linear regret. So we need something stochastic.

Let $\hat{X}_{i,t} = \frac{\delta_{I_t=i}}{P_{i,t}} x_{I_t,t}$ where $I_t \sim P_t \in \mathbb{S}^n$. Then $\mathbb{E}[\hat{X}_{i,t}] = x_{i,t}$.

The algorithm we need will be able to deal with the high variance of this estimator. It is called EXP3 (exponential something for exploration and exploitation). Playing this game, you can get

$$R_T \leq \sqrt{nt \log n}.$$

Some textbooks: Seb has one. You can find a preprint on the arXiv. Also check out banditalgs.org.

4 Deep Learning

This talk was given by Joan Bruna from NYU's Courant Institute.

4.1 The Puzzle

We are primarily going to be interested in what is called *supervised learning*. Notice that this specifically omits some hot ideas that are currently being developed. Throughout the talks, we will be considering $\mathcal{X}$, a *high-dimensional* space. For instance, if we were working in the domain of computer vision, $\mathcal{X}$ may be the space of all $d \times d$ images, giving us a huge space, potentially.

The next ingredient we have is the **data distribution** ν on $\mathcal{X}$. Importantly, U is *unknown to us*, so we are not allowed to directly access any of its properties. Next, we have a **target function** $f^* : \mathcal{X} \to \mathbb{R}$ which we are trying to learn. We also need a loss functional $L(f) = \mathbb{E}_\nu[l(f(x), f^*(x))]$.

4.1.1 Remark: In this class, we will almost always restrict to the case when L is given by

$$\|f - f^*\|^2_{L^2(\mathcal{X},\nu)}$$

Then our goal is to predict f^* from a finite IID sample $\{(x_i, f(x_i)\}$, where $x_i \sim \nu$. Notice that this is basically the same as a problem we have all done at some point: fit a curve to a finite number of points. But now the dimensionality has exploded.

4.1.1 Empirical Risk Minimization

Consider a set $\mathcal{F} \subseteq \{f : \mathcal{X} \to \mathbb{R}\}$, which we call the **hypothesis class**. This space of functions comes along with a notion of *complexity*. We have a value $\gamma(f)$ for each $f \in \mathcal{F}$, which we call the **complexity of f.** One should think of this like a norm. The upshot here is this gives us a way to "organize" $\mathcal{F}$ in a nice way.

A natural way to proceed is to begin with the norm zero piece and gradually increase to consider more candidate functions.

4.1.2 Remark: Careful! Notice that assuming that γ is honestly a norm means that such "balls" are already convex! In general this will not be true.

4.1.3 Definition: The **empirical risk** of a function f is

$$\hat{L}(f) = \frac{1}{n}\sum_1^n |f(x_i) - f^*(x_i)|^2.$$

Now if one wants to minimize $\hat{L}$, there are a couple different ways to do this. The first idea is consider the **constrained form**:

$$\min_{f \in \mathcal{F}^\delta} \hat{L}(f)$$

where $\mathcal{F}^\delta = \{f \in \mathcal{F} : \lambda(f) \leq \delta\}$. This requires a bit more prior knowledge than we generally have, so we sometimes use instead a **penalized form**:

$$\min_{f\in\mathcal{F}} \hat{L}(f) + \lambda \cdot \gamma(f).$$

Another idea is called the **interpolant**:

$$\min_{f\in\mathcal{F}} \gamma(f) \text{ such that } \hat{L}(f) = 0$$

4.1.2 The Fundamental Theorem of Machine Learning

Let us assume for this formulation that we are focusing on the constrained form. This was found in the seminal paper from 2008 by Bolton and Bousquet.

4.1.4 Theorem (FtoML)
Assume that we have $\hat{f} \in \mathcal{F}^\delta$ such that $\hat{L}(\hat{f}) \leq \varepsilon + \inf_{f\in\mathcal{F}^\delta} \hat{L}(f)$. Then

$$L(\hat{f}) - \inf_{f\in\mathcal{F}} L(f) \leq \inf_{f\in\mathcal{F}^\delta} L(f) - \inf_{f\in\mathcal{F}} L(f) + 2\sup_{f\in\mathcal{F}^\delta} |L(f) - \hat{L}(f)| + \varepsilon.$$

4.1.5 Remark: Here the first two terms on the right constitute an approximation error, the third term is a statistical error, and the ε comes from our optimization error.

Notice that we have a free parameter δ. If δ is small, then our approximation error gets very bad and the statistical error drops (we are working on a "smaller" set in terms of complexity). On the other hand, if δ is very big, the statistical error blows up.

4.2 The Big Questions

We have a series of questions we want to answer with this theory:

- [**Approx**] How can we design "good" spaces $\mathcal{F}$ to approximate f^* in higher dimensions?
- [**Optimization**] How to design algorithms to solve the problem of empirical risk management in general (gradient descent).

4.2.1 The Curse of Dimensionality

To begin to convince ourselves that dimensionality is troubling, consider the question: How many samples do I need to estimate f^* depending on its regularity assumptions?

If we assume that f^* is *linear*, then $f^*(x) = \langle x, \theta^*\rangle$ for some $\theta^* \in \mathbb{R}^d$. Then $\mathcal{F}$ is the set of all such functions and we need d samples to get enough equations to solve the corresponding linear equations.

Now assume that f^* is "locally linear", that is *Lipschitz*. That is, there exists a (minimal) β such that

$$|f^*(x) - f^*(y)| \leq \beta\|x - y\|.$$

Then $\mathcal{F}$ is the space of all Lipschitz functions. Then $\gamma(f)$ is determined by the Lipschitz constant as well as $|f|_\infty$. Now for all $\varepsilon > 0$, we want to find $f \in \mathcal{F}$ such that $\|f - f^*\| \leq \varepsilon$ from n IID samples. Then in this case, the sample complexity blows up! To get an ε-error, we need $n \sim \varepsilon^{-d}$ samples!

Let's build the upper bound here: given a set of points $(x_i, f^*(x_i))$, we want to find

$$\hat{f} = \operatorname*{argmin}_f \operatorname{Lip}(f) \text{ such that } f(x_i) = f^*(x_i)$$

where $\operatorname{Lip}(f) = \min_\beta |f(x) - f(y)| \leq \beta \|x - y\|$. To do this, we can, for any x, pick the closest data point x_k to x and notice

$$|\hat{f}(x) - f^*(x)| \leq |\hat{f}(x) - \hat{f}(x_k)| + |\hat{f}(x_k) - f^*(x_k)| + |f^*(x_k) - f^*(x)| \leq 2 \operatorname{Lip}(f^*) \|x - x_k\|.$$

But now

$$\mathbb{E}_{x\sim\nu} |\hat{f}(x) - f^*(x)|^2 \leq \mathbb{E}_{x\sim\nu} \|x - x_k\|^2 \approx \varepsilon^2.$$

4.2.1 Definition: The **maximum discrepancy** of a function class $\mathcal{F}$ is

$$\widetilde{R}_n(\mathcal{F}) = \sup_{f\in\mathcal{F}} \frac{1}{2n} \sum_1^{n/2} f(x_i) - \sum_{n/2+1}^{n} f(x_i))$$

5 Reinforcement Learning

This series of lectures was given by Emma Brunskill from Stanford's CS department.

5.1 Introduction

There is a wide set of backgrounds in this class will start broad and get more focused towards the end. One can think of reinforcement learning as being at the intersection of supervised learning and bandits (and beyond). In a sentence, reinforcement learning is "learning to make good sequences of decisions under uncertainty."

5.1.1 Examples of things you can do

The main applications arise in a couple different domains: learning to play games (e.g. Atari games), teaching robots tasks, and in healthcare. There are a plethora of applications we will hear about more.

5.1.2 Tasks

There are four main aspects:

- Optimization
- Generalization
- Exploration
- Delayed consequences

In general **AI learning** uses all of these except exploration. Although RL was used in solving go, it wasn't needed.

Deep learning only exploits optimization and generalization. There is also a sense of **imitation learning**, which also omits exploration.

We will be following the following (rough) plan:

(a) Markov decision processes

(b) Q-learning

(c) RL and generalization

(d) Exploration in sequential domains

(e) Batch/counterfactual reinforcement learning

5.2 Markov Decision Processes

The big idea here is that two entities: the agent and the world. The agent takes an action and the world offers a reward. In the case of bandits, this is where the scenario stops, but here in RL, we continue the feedback loop. Our goal here is to maximize the expected discounted sum of rewards.

Some ideas that demonstrate the delayed consequences are as follows: imagine teaching a robot to empty the dishwasher—it only receives a positive reward once its task is complete.

Now one of the hardest aspects here is coming up with a **reward function**. That is, a real-valued map $r(s, a)$ given a state and action. Why is this hard? Well, imagine you are trying to teach students math and the fact of the matter is that teaching addition is easier than subtraction, and we design our reward to be +1 if the student is correct and -1 if incorrect. Then the machine is naturally going to give only additon questions (and infinitely many of them). This is an example of *reward hacking.*

A problem follows **Markov dynamics** if

$$\mathbb{P}(s_t|s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \cdots) = \mathbb{P}(s_t|s_{t-1}, a_{t-1}).$$

5.2.1 Definition: A **Markov decision process** is a set $\langle S, A, R, T, \gamma \rangle$ where

- S is a set of states,
- A is a discrete action space,
- R is a set of rules of the form $r(s,a)$ or $r(s)$ or $r(s,a,s')$,
- T is a collection of distributions $\mathbb{P}(s'|s,a)$ and $\gamma \in (0,1)$ is a parameter.

Then we are trying to learn a **policy** $\pi : S \to A$. Note this could also be stochastic.

There are a couple ways to do this: first, is the **model-based approach**. Here we directly estimate and use R and T. Another is **value-based:** here we consider the value function

$$V^\pi = \mathbb{E}_{s' \sim pi}\left[\sum_0^\infty \gamma^t r_t | s_{t=0} = s_0\right].$$

Finally we can perform a **policy-based approach** by computing $\text{argmax}_{\pi \in \Pi} V^\pi$.

Another parameter we have is H, the **time horizon** on the process. This tells us the number of steps that we may take before the system is reset to s_0.

Then this process will return

$$G^\pi = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^{H-1} r_H.$$

Notice that in general this is going to be a random variable! That;'s why $V^\pi(s_0) = \mathbb{E}[G^\pi | s_0]$.

5.2.1 Estimation

Now say we know all the parameters of our MDP. Then we can use S as a lookup table to approximate V^π by "running simulations." That is, we essentially have a black box that outputs specific measurements of G. We don't need the Markov assumption here since we aren't touching the distributions directly.

Using the Markov assumption yields additional structure, however. Consider that

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s|s, \pi(s)) V^\pi(s')$$

giving us our immediate and discounted future rewards.

Now let $V^* = \max_\pi V^\pi$ and $\pi^* = \text{argmax}_\pi V^\pi$. But then using the above equation,

$$V^*(s) = \max_a r(s,a) + \gamma_{s'} p(s'|s,a) V^*(s')$$

which induces a kind of dynamical programming structure on the problem. This is called the **Bellman equation.**

Then we proceed by a process called **value iteration**: let $V_0(s) = 0$ for all s. Then for all k, set

$$V_k(s) = \max_a r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_{k-1}(s')$$

Doing this is called a "Bellman backup." Then we break once you converge: $\|V_k - V_{k-1}\|_\infty \leq \varepsilon$.

5.2.2 Lemma
Assuming all rewards are valued in $[0,1]$, the Bellman backup is a contraction operator as long as the state and action spaces are discrete (i.e. the above process converges).

Proof
Throughout we will be considering $\|V - V'\|_\infty = \max_s |V(s) - V'(s)|$. So we compute

$$\begin{aligned}
\|BV_k - BV_j\| &= \| \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_{k-1}(s')) - \max_{a'} (r(s,a') \\
&\qquad + \gamma \sum_{s'} p(s'|a',s) V_{j-1} s) \| \\
&\leq \| \max_a \gamma \sum p(s'|s,a) \| V_k - V_j \| \\
&\leq \gamma \| V_k - V_j \|
\end{aligned}$$

so as long as $\gamma < 1$, our functions have gotten closer. ♠

5.2.3 Remark: Notice this also implies that V^* must be unique.

5.2.2 MDP Policy Iteration

Value iteration is great, but we have to consider the set of functions A^S, exponential in the set of states. Instead we can do **Policy Iteration**: Set $i = 0$ and initialize $\pi_0(s)$ randomly for all states s. Then while $i = 0$, or $\|\pi_i - \pi_{i-1}\|_1$ 0, set V^{π_i}, the MDP value function policy evaluation of π_i and let π_{i+1} be the policy improvement.

Note that we can find an analytic solution whenever you have a finite set of states and actions, otherwise you have to do the dynamic programming-type argument using the Bellman operator.

5.2.4 Definition: The Q value of a policy is

$$Q^\pi(s,a) = r(s,a) + \gamma + \sum_{s'} p(s'|s,a) V^\pi(S')$$

so now that we have computed V^{π_i} using analytical or dynamic programming methods, this is a relatively simple step. Then we just pick

$$\pi_{i+1}(s) = \operatorname*{argmax}_a Q^{\pi_i}(s,a).$$

5.2.5 Remark: It ends up that $V^{\pi_{i+1}} \geq V^{\pi_i}$ with strict inequality whenever π_i is sub-optimal. Notice that this is not trivial! We are getting better optimization over all time, but since we are switching policies greedily at each step, there is something to prove.

5.3 Reinforcement Learning and Q-learning

Now we are performing reinforcement learning in a contex where R and T are unknown, but we are still interested in finding V^* and π^*.

5.3.1 Policy Evaluation

Here we are given π and want to compute V^π and Q^π.

If **our horizon H is finite**, then we can do **Monte Carlo Evaluation**. If we let (again)

$$G^i = r_1 + \gamma r^2 + \cdots$$

where $s, r \sim \pi$ starting with s_0. Then the value is

$$V^\pi(s_0) = \mathbb{E}_{s\sim\pi}[G|s_0]$$

where the above does not require the Markov assumption and doesn't use the "bootstrapping" of MDP value iteration (where make incremental improvements to an imperfect hypothesis). Then the average of all runs acts as our estimator. This is unbiased but has high variance (although it converges by the law of large numbers).

5.3.2 Temporal Difference Learning

This is called "model free" in the sense that we are not directly estimating the dynamics and reward models. This combined ideas in Monte carlo and dynamic programming. It is episodic (finite horizon) or infinite H. Then each tuple (s,a,r,s') can update $\hat{V}^\pi$ (instead of having to wait for an entire run).

The MC estimator updates at each s_0:

$$\hat{V}(s_0) + \alpha(G^i - \hat{V}^*(s_0))$$

where $\alpha \in (0,1)$ is a learning rate. Then the idea we do here is

$$\hat{V}_{k+1}(s) = r(s,\pi(s)) + \gamma \hat{V}_k(s')$$

Notice that this uses bootstrapping of $\hat{V}$ as well as doing a sampling MC to get the data to update with. This new estimator will be biased, but wil have lower variance of what we did before.

So overall the TD learning we are doing looks like what follows:

$$\hat{V}^\pi(s) = \hat{V}^\pi(s) + \alpha\left[r + \gamma\hat{V}^\pi(s') - \hat{V}^\pi(s)\right]$$

where $r + \gamma\hat{V}^\pi(s')$ is called the **TD target** and this quantity minus $\hat{V}^\pi(s)$ is called the **TD error.**

For mean estimation, let $X \in [0,1]$ is a random variable and let

5.3.1 Theorem
Assume that $\alpha_n \in [0,1]$ and $\sum \alpha_i = \infty$ and $\sum \alpha_i^2 < \infty$, and let $X_0, x_1, \dots$ be iid samples of X. THen define μ_n for each $n \in \mathbb{N}$ such that

$$\mu_0 = X_0, \quad \mu_{m+1} = (1-\alpha_m)\mu_m + \alpha_m x_m$$

then μ_m almost surely approaches $\mathbb{E}[X]$.

5.3.2 Remark: I can't really read and my probability isn't strong enough to figure out what she meant.

5.3.3 Control

Compute π^* and V^*.

5.4 Reinforcement Learning and Generalization

5.5 Exploration in Sequential Domains

5.6 Batch/Counterfactual RL