CS 228: Computational Learning Theory

Eric K. Zhang
ekzhang@college.harvard.edu

Franklyn H. Wang
franklyn_wang@college.harvard.edu

Spring 2020

Abstract

These are notes for Harvard's *CS 228*, a graduate-level class on computational learning theory taught by Leslie Valiant[1] in Spring 2020. The textbook used in this class is *An Introduction to Computational Learning Theory* by Kearns and Vazirani (MIT Press, [KV94b]).

Course description: Possibilities of and limitations to performing learning by computational agents. Topics include computational models, polynomial time learnability, learning from examples and learning from queries to oracles. Applications to Boolean functions, automata and geometric functions.

Contents

[1] With teaching fellow: Benjamin Edelman

1 January 28

Today is the first lecture; we discuss broad goals for the course, as well as an example for why inductive learning is possible in the statistical sense.

1.1 Introduction

This is CS 228, computational learning theory.

There are four different aspects to this class. It can be thought of as philosophy, cognition, making predictions, and machine learning technology. To do well in this class, you have to have some reasonable ideas about how computers work and their underlying theory.

In philosophy, Aristotle believes that all belief comes from syllogism in induction. There's a phenomenon that we can see a few things that look like chairs, and then we see some chairs that we've never seen before. Millions of children learn words, and just like this, machine learning is induction. In fact, things that you've learned got there through evolution. You can phrase Darwinian evolution as learning in some sense. This implies that all information in living things got there by a learning process.

The science angle can be illustrated with the following real-world example. Old Faithful is a geyser in a Yellowstone National Park. It erupts once every 70 to 110 minutes, and the only thing you want to do there is to go out to see this eruption. There was a park ranger who predicted when the next eruption was. He used the nearest neighbor method. If you look at the interval between the last two eruptions, e.g. 80 minutes, and the length of the last eruption, e.g. 52 seconds, you can get a pretty good idea as to the state of the geyser. One can construct a large table with these numbers to determine predictions for the future. This is a very basic machine learning algorithm called k-nearest neighbors.

Now we have machine learning technology, which is a new paradigm for computers. Instead of programming them, you can have them learn instead. Here there are a few interesting problems.

1. Ultimate Limits: what can a machine learning model learn?
2. Sample Complexity: how many samples are needed to learn?
3. Time compexity: how much time does it take to learn?
4. Learning: Which kinds of algorithms work?
5. Interpretability: Explain why your prediction is made.

There's a paper, "Very Simple Classification Rules Perform Well on Most Commonly Used Datsets" by Holte (1993). The point of this paper is that very simple decision trees (only two or three leaves) are often close to optimal.

Even though the class is theoretical, you can still do experiments for the final project. **However, you should not do projects that have no relevance to the class**. Projects can be done in pairs, and no larger groups are allowed.

1.2 An Interesting Problem

For the rest of the class, we will do an example which has a learning theory flavor, which discusses whether induction is possible.

Consider a hypothesis that classifies numbers of the set $X = \{1, 2, \ldots, 10^{100}\}$ as positive or negative, where each hypothesis can be represented by 100 binary bits. Let $Y \subseteq X$ is a subset of

X with half of the size, so that $|Y| = |X|/2$. Now, we'll let c^* be functions that map X to $\{0, 1\}$. Let c^* be the indecator of Y, so that $c^*(x) = 1$ iff $x \in Y$ and 0 otherwise.

Example 1.1. Given a small set $Z \subseteq X$ (the training set) so that $|Z| = 300$, and the value of $c^*(x)$ for each $x \in Z$ we want to determine the value of $c^*(t)$ for any $t \in X$.

This problem is impossible as stated, so we need to make some assumptions. With no assumptions, there is nothing you can say, because there are still effectively $2^{10^{100}-300}$ choices of Y (this is not exactly right, but you still can't tell whether an element is more likely to be positive or negative).

Are there some minimal assumptions that make induction justified? There are two assumptions are needed to make this argument completely work:

1. There is a succinct program h^* (written in a programming language) that evaluates c^* (e.g. has 100 bits).

2. There is a natural distribution $\mathcal{D}$, and you are tested on that distribution, and the samples $x_1, \ldots, x_{300}$ are drawn independently from $\mathcal{D}$.

Theorem 1.2. *Now, we can show that any* 100 *bit program that predicts on all of* Z *correctly will be a usually reliable predictor for the other* $10^{100} - 300$ *different numbers correctly. Generally, on a random* $t \sim \mathcal{D}$*, the probability that* t *is misclassified is at most* 25%*.*

Proof. Assume that our algorithm is:

1. Take all programs which are perfect on the training data, say $\mathcal{P}$.

2. Choose a random element from $\mathcal{P}$.

Note that this will always yield a result, because at least one program will be perfect on the training data. Some hypotheses are bad, meaning that they classify examples correctly with at most 75% probability and have fewer than 100 bits of code. There are at most 2^{100} of these hypotheses. For each such bad hypothesis, the probability that it predicts on all of Z correctly is at most $(75\%)^{300} = 2^{-120}$. Thus, by union bound with probability at least $1 - 2^{-20}$ no bad hypothesis will classify all of Z correctly, so with probability at least $1 - 2^{-20}$ the hypothesis we end up choosing was good. □

The general principle that this shows is that statistical inductive learning can be justified essentially universally. However, it does not explain how a good hypothesis can be computed from the data. A hypothesis from a person is valuable (in that you would buy it) if that hypothesis does well on random points in the past, and if it's short (which intuitively means that the existence of such an algorithm is "rare").

The hard problem is to find an algorithm that finds c, meaning that it agrees with data. This is the entire problem of machine learning: finding algorithms that agree with data. In essence, machine learning is computation added to statistics.

1.3 Addendum

It's worth noting that in the above, some of the notation is somewhat unnecessary. Another way to think about it is that you have 2^{100} coins, and each of them has a probability p of landing on heads. One of the coins in particular has probability 1 of landing on heads. If you flip all the coins and discard them once they flip tails, the remaining coins are all probably going to have pretty high chances of landing on heads if you flip them again.

2 January 30

This is the second lecture of computational learning theory, where we move toward formalizing our model by introducing key examples of algorithms that will motivate PAC learnability.

Recall from last lecture that a hypothesis from a **small hypothesis** set that agrees with a **large enough** random set of training examples in $\mathcal{D}$ will with high probability predict well on future examples from $\mathcal{D}$. The hypothesis has to be small, and the previous hypothesis set must also be small enough.

This can be used to define a rigorous answer to why learning is possible. There are four questions that we will try to answer:

1. Computational complexity: how fast can we find a hypothesis that does well?
2. Sample complexity: how many examples are needed before an algorithm can be learned?
3. Noise: in some cases, it may be hard to be perfect.
4. Are there other arguments for generalization?

2.1 Compact Cars

Example 2.1 (Compact Cars)**.** Suppose that all cars are either *compact* or not, based entirely on one feature: length. Letting x be the length, the function we seek to learn is therefore

$$c(x) = \begin{cases} 1 & \text{iff } a \le x \le b \\ 0 & \text{otherwise} \end{cases}.$$

The ultimate goal is, given a new random car, how can we predict it well? Suppose that compact cars are drawn from a distribution $\mathcal{D}^+$ supported on $[a, b]$. Also, suppose that the other cars are drawn from a distribution $\mathcal{D}^-$ supported on $\mathbb{R} \setminus [a, b]$. First, we try to learn from only positive examples, by, say, renting a compact car.

The algorithm works as follows. Take m positive examples, $\ell_1, \ell_2, \dots \ell_m \sim \mathcal{D}^+$. Now let $\hat{a}, \hat{b}$ be the minimum and maximum lengths. Then the interval is $[\hat{a}, \hat{b}]$. One question is, does this learn, and how large can m be? This motivates the following theorem.

Theorem 2.2. *$\forall \epsilon, \delta > 0$, if $m \ge \left(\frac{2}{\epsilon}\right) \ln(2/\delta)$, then with probability at least $1 - \delta$ the hypothesis will be correct on a random $x \in \mathcal{D}$ with probability $1 - \epsilon$.*

Proof. This is a standard proof: note that there is a true distribution of positive examples. The idea here is that if we can get something in both the first $\epsilon/2$ tail and the last $\epsilon/2$ tail of the distribution, then we're good. For one example x, the probability that x misses the left $\epsilon/2$ tail is equal to $1 - \epsilon/2$. Thus, the probability that they all miss the left tail is at most $(1 - \epsilon/2)^m$. The probability that either tail is missed is, by a union bound, at most twice the probability that a given tail is missed. This probability is then

$$2\left(1 - \frac{\epsilon}{2}\right)^m \le 2e^{-m\epsilon/2} < \delta,$$

where we have used the fact that $1 - x < e^{-x}$ for all $x > 0$. and we may conclude. □

Note. If you want something that has error rate 10^{-6}, you need around 10^6 training examples to have an acceptable probability.

1. This is non-parametric; it works for any distribution $\mathcal{D}$, and does not assume that, say, $\mathcal{D}$ is normal.

2. It is possible to learn from the positive distribution only.

Our two parameters are ϵ, which is roughly the resolution of your distribution, and δ, which is a function of your lack of luck; it's possible that everything you sample was from a very unrepresentative portion of the training set.

2.2 Learning Boolean Conjunctions

Example 2.3. Now let's talk about learning Boolean conjunctions, like learning functions of $x_1, \dots, x_n$. Assume that the correct formula looks like $x_2 \wedge \overline{x}_4 \wedge x_7$.

Again, we might consider learning from positive examples only. If we see that there is a positive example where $x_1 = 1$ and one where $x_1 = 0$, we know that neither x_1 nor $\overline{x}_1$ can show up in the answer. However, if $x_2 = 0$ *in all positive examples*, you should have a pretty good deal of confidence that $\overline{x}_2$ is in the conjunction, *as long as* cases in which $x_2 = 1$ are not simply sampled very infrequently! Thus, consider the algorithm which takes every literal that has not eliminated from the presence of the data alone. For example, if there is a training example with $x_1 = 1$, obviously $\overline{x}_1$ is eliminated.

Again, this algorithm is correct on all negative samples, because the algorithm isn't conservative and is always aggressive. It finds the most aggressive hypothesis possible given the data. In particular, the solution found is always the true solution combined with other things.

Now, here's a theorem akin to the above one on learning compact cars.

Theorem 2.4. *$\forall \epsilon, \delta > 0$, if the algorithm is executed for $m = (2n/\epsilon)(\ln(2n) + \ln(1/\delta))$ samples for $\mathcal{D}^+$, then with probability at least $1 - \delta$ the first hypothesis will have error at most ϵ on $\mathcal{D}$.*

Proof. For each of the $2n$ literals $z \in \{x_1, \overline{x}_1, \dots x_n, \overline{x}_n\}$, let $p(z)$ be the probability that a random draw for $\mathcal{D}^+$ produces that element. Note that if $p(z) > 0$ then z cannot be in the true conjunction. Define z to be bad if $p(z) > \epsilon/2n$. If all bad literals have been eliminated then retention of those with $p(z) \leq \epsilon/2n$ will cause error at most $2n \cdot \epsilon/2n = \epsilon$ since there are at most $2n$ literals.

The only remaining question is, how many examples do we need so that all bad literals are hit with probability $1 - \delta$? For a fixed bad z, the probability that z is not eliminated in m examples is at most

$$\left(1 - \frac{\epsilon}{2n}\right)^m.$$

Thus, we need for

$$2n\left(1 - \frac{\epsilon}{2n}\right)^m < \delta,$$

which we can again show with the trick that $1 - x < e^{-x}$. □

It's important to note that in this problem, we have not assumed that the Booleans satisfy any independence conditions, or i.i.d.-ness, which would make the problem considerably easier.

3 February 4

Today we define various kinds of Boolean functions, and we finally formalize the notion of PAC learnability.

3.1 Types of Boolean Functions

We will be using Boolean operations in many of our learning problems. For two variables x, y, we will use their concatenation xy to notate AND (conjunction), and we use the addition symbol $x+y$ to notate OR (disjunction). Also, $\overline{x}$ represents the negation of x.

In general, Boolean formulas consist of some nested conjunctions, disjunctions, or negations of some finite number of variables. These can represent all Boolean circuits and Boolean functions. However, we would usually like to restrict our attention to certain special cases of formulas.

Definition 3.1. We define various common types of Boolean formulas.

1. A *conjunction* is a simple conjunction of one or more variables, of which some may be negated. Examples are x_1x_2 and $x_3\overline{x_5}x_1$.

2. A *disjunction* is a simple disjunction of one or more variables, of which some may be negated. Examples are $x_1 + x_2$ and $x_3 + \overline{x_5} + x_1$.

3. A *disjunctive normal form (DNF)* is the conjunction of one or more disjunctions. An example is $x_1x_3x_7 + x_2\overline{x_4}$.

4. A *disjunctive normal form (DNF)* is the disjunction of one or more conjunctions called *clauses*. A simple example with 2 terms is $x_1x_3x_7 + x_2\overline{x_4}$.

5. A *conjunctive normal form (CNF)* is the conjunction of one or more disjunctions called *clauses*. An example is $(x_1 + x_3 + x_7)(x_2 + \overline{x_4})$.

6. A *k-term* CNF or DNF is a formula as defined above, but with at most k clauses.

7. A k-CNF or k-DNF is a CNF or DNF where each clause has at most k literals.

Another type of Boolean function is the decision tree, where the appropriate path from every node is conditioned on some literal. A decision tree can be expressed in disjunctive normal form, because one can mark each path to the root with a conjunction, so taking the disjunction of these gives us a DNF formula.

A stricter class of hypotheses than a decision tree is a k-decision list, which requires that for each node, at least one of the children is a leaf. However, each node of the list can be the conjunction of up to k literals. Analogously to decision trees, k-decision lists can be written in k-DNF form.

3.2 Boolean Duality

Definition 3.2 (Dual)**.** The *dual* of a Boolean formula is one in which all the ORs are replaced with ANDs, and vice versa. The dual of a function applied to inverted inputs yields inverted outputs; this is known as *De Morgan's law.*

For an example of duality, note that

$$(\overline{x}_1 + x_3)(x_4 + \overline{x}_3)(\overline{x}_7 + x_8) = \neg(x_1\overline{x}_3 + x_3\overline{x}_4 + x_7\overline{x}_8).$$

While above we described many types of function classes, it is now time for us to discover that some of these classes are in fact contained within others. Observe that

$$(x_1\overline{x}_2 + \overline{x}_3x_4) = (x_1 + \overline{x}_3)(x_1 + x_4)(\overline{x}_3 + \overline{x}_2)(\overline{x}_2 + x_4).$$

This shows us that k-term DNF is a subset of k-CNF.

3.3 Defining PAC Learning

We will now define PAC learning. To do this, we will need a decent amount of technical definitions. First, we have a *concept class* $\mathcal{C}$ of possible functions to learn, which contains many *concepts* $c : X \to \{0,1\}$. We will also have a *hypothesis class* $\mathcal{H}$ of possible hypotheses h that our learning algorithm can output. It will be useful to define the concept class and hypothesis class as a sequence of inclusions, so that

$$\mathcal{C} = \bigcup_{i\geq 1} \mathcal{C}_i \text{ and } \mathcal{H} = \bigcup_{i\geq 1} \mathcal{H}_i.$$

In essence, we have a concept class $\mathcal{C}$ and a hypothesis class $\mathcal{C} \subseteq \mathcal{H}$. We will also need an oracle $EX(c, \mathcal{D})$ which draws examples from a probability distribution $\mathcal{D}$ supported on X. The goal is to uses these examples to learn $c \in \mathcal{C}$ by finding a hypothesis $h \in \mathcal{H}$ so that h and c do not differ by much. This is why we want our learning algorithm to be *approximately correct*; we want to bound our *error* given by

$$\text{error}_c(h) = \Pr_{x\sim\mathcal{D}}(c(x) \neq h(x)).$$

However, in general it will be asking too much for us to always have a finite bounded error. If are very unlucky, then our extracted samples could be unrepresentative of the actual distribution, and we could have arbitrarily high error. This motivates the following definition of *probably approximately correct* (PAC) learning.

Definition 3.3 (Efficient PAC learning)**.** Let $\mathcal{C}_n$ be a concept class over either $\{0,1\}^n$ or $\mathbb{R}^n$. We say that $\mathcal{C}$ is *PAC learnable* if there exists an algorithm L with the following property: for every concept $c \in \mathcal{C}_n$, for every distribution $\mathcal{D}$ on X, and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $EX(c, \mathcal{D})$ and inputs ϵ and δ, then with probability $1 - \delta$, L outputs a hypothesis concept $h \in \mathcal{C}$ satisfying $\text{error}(h) < \epsilon$. L must also run in time polynomial in $1/\epsilon$, $1/\delta$, n, and the size of the concept c.

In many cases, it will be the case that the size of the concept c is already polynomial in n. In these cases, we only need our algorithm L to be polynomial in $1/\epsilon, 1/\delta$, and n.

4 February 6

Today we discuss reductions between various learning problems on Boolean functions, which are useful for showing equivalences and intractability.

4.1 Learning k-CNF and k-DNF

Last class we talked about various types of boolean formulas and how easy they are to learn. It can be shown that k-DNF and k-CNF expressions are equivalent in terms of learnability. To discover why this is the case, read on!

The biggest class of Boolean functions that you can hope to learn is the set of all Boolean circuits. It can be shown that learning a Boolean cicruit is very difficult. In particular, being able to learn arbitrary Boolean circuits would allow one to break very strong crpytographic systems.

We have also discussed 3-CNF and 3-term CNF. Today we will think about learning these! Recall that every 3-term DNF can be expressed in 3-CNF form. Learning 3-term DNFs from 3-term DNFs is NP-hard. However, 3-CNFs are PAC-learnable, even though 3-CNFs are a superset of 3-term DNFs. A trivial consequence of this fact is the following:

Proposition 4.1. *The concept class of* 3*-term DNFs is PAC learnable using the hypothesis class of* 3*-CNFs.*

This should feel **very, very strange**! To recap, it is very difficult to find an algorithm that learns a specific 3-term DNF from samples. However, it is tractable to find a 3-CNF which approximates this. Intuitively, what's going on here is that being restricted to a 3-CNF in the hypothesis space serves as a straitjacket that prevents us from finding an approximation, but the function is more-easily learnable by a richer representation.

Proposition 4.2. *If $\mathcal{C}$ is learnable by $\mathcal{H}$, then $\mathcal{C}$ is also PAC-learnable by any class $\mathcal{H}' \supseteq \mathcal{H}$.*

4.2 Reductions among Learning Problems

Now we can go over reductions among learning problems which show that one problem is solvable given that another one is solvable. There are two simple general techniques.

1. **Recoding:** A conjunction is 1-CNF, e.g.

   $$x_1\overline{x}_3x_8$$

   We can learn 1-CNF, as we showed in Lecture 2.

   Can we learn 3-CNF? Recall that these expressions look like

   $$(x_1 + \overline{x}_2 + x_7)(\overline{x}_2 + x_4 + x_6)\dots$$

   The miracle is that this turns out to be pretty easy! Every formula in 3-CNF is simply a collection of $8n^3$ of these clauses, so we can simply use the elimination algorithm defined earlier. It is relatively straightforward to show that this is PAC learnability.

2. **Duality** Recall that by replacing each of the terms with their inverse and by replacing each of the ors with ands and vice versa flips the entire function. If class A is learnable from class B, then the dual of A is also learnable from the dual of B. The construction is exactly how

you might expect it to look. This lets us show that 3-CNFs are learnable from 3-CNFs iff 3-DNFs are learnable from 3-DNFs. In this case, it shows that after showing 3-CNFs are learnable from 3-CNFs, 3-DNFs are learnable from 3-DNFs.

The last part of this is to show that it is NP-hard to learn 3-term DNF from 3-term DNF.

Claim. If 3-term DNF is PAC learnable by 3-term DNF, then $NP = RP$, where RP is the complexity class of polynomial-time randomized algorithms with one-sided error.

To do this, we can show that this is as hard as 3-coloring graphs. In next lecture, we will see that there is a polynomial-time transformation f that takes n-node graphs to a set S of $O(n^2)$ labeled examples.

If we have such a construction f, then consider a uniform distribution $\mathcal{D}$ over the examples. Now, apply any supposed PAC-learning algorithm L to S, with epsilon equal to $1/2|S|$. By Markov's inequality, this function will be correct on all points with probability at least $1/2$. If $|S|$ is consistent with some 3-term DNF, then with high probability L will output such a formula.

Second, if S is not consistent then L outputs something that can be verified to be wrong. Therefore, if L PAC-learns 3-term DNF using 3-term DNF, it must implicitly determine whether G is 3-colorable with high probability.

5 February 11

Today we continue discussing lower bounds for learnability, i.e., methods for showing that some intractable problems that are not PAC learnable. We also introduce Occam learning.

5.1 Reducing Graph Coloring to 3-Term DNF

First we prove a result about 3-term DNFs, or Boolean expressions in disjunctive normal form that have at most 3 disjunctions:

$$(x_1 \wedge x_3 \wedge x_5 \wedge \neg x_4) \ \vee \ (-) \ \vee \ (-).$$

Proposition 5.1. *If $RP \neq NP$, 3-term DNFs are not PAC learnable from sets of labeled examples.*

Proof. We will prove this by reduction from graph 3-colorability. Define a function f that takes an n-node graph to a set $S = S^+ \cup S^-$ of labeled examples over $\{x_1, \dots, x_n\}$, such that G is three-coverable if and only there exists a 3-term DNF which is consistent with S.

Let $G = (V, E)$ such that $n = |V|$. Then, f will take G to a set of n positive examples and $|E|$ negative examples, of the following form:

$$S^+ = \{v(i) = \langle 11 \dots 101 \dots 1 \rangle \mid 1 \leq i \leq n\},$$

$$S^- = \{v(i,j) = \langle 11 \dots 10110111 \rangle \mid (i,j) \in E\},$$

where in the above expression for S^+, all variables are "1" except the bit x_i in the i-th position, and in the definition of S^-, all variables are "1" except the two bits x_i, x_j in the i-th and j-th positions.

To prove the simpler first direction, we will show that if G is 3-colorable, then there exists a 3-term DNF consistent with $f(G) = (S^+, S^-)$. Consider any 3-coloring of G with colored sets of vertices Y, G, B. Then we can write a corresponding 3-term DNF $T_Y + T_O + T_B$, where T_Y is the conjunction of all variables that correspond to vertices that are not in Y. Likewise, we use T_O to refer to the conjunction of all variables that are not in O, and T_B to refer to the conjunction of variables not in B.

Clearly, $T_Y + T_O + T_B$ is satisfied for any example $v(i)$ in our set S^+, which has x_i missing. If vertex i is colored X, then the clause T_X will be satisfied. Also, the three-term DNF is not satisfied for any example $v(i,j)$ in our set S^-, as i and j are of two distinct colors since they are connected by an edge, so at least one of these two vertices is present in each of the clauses T_Y, T_O, and T_B. Thus, if G is 3-colorable, then we have constructed a 3-term DNF that is consistent with $f(G) = (S^+, S^-)$.

Now, we prove the reverse direction. Consider a graph G and a 3-term DNF that satisfied for all $v(i) \in S^+$, and not satisfied for all $v(i,j) \in S^-$. Let this 3-term DNF be written as

$$T_Y + T_O + T_B.$$

Then, we can define a coloring of G as follows:

- The color of node i is X if $v(i)$ satisfies T_X.
- If $v(i)$ satisfies more than one T_X, then pick one arbitrarily.

Since every $v(i) \in S^+$ is satisfied, each node i can be assigned a color with the above method. Furthermore, we argue that this is a valid 3-coloring of G. Suppose for the sake of contradiction that there exists some edge $(i, j) \in E$ such that vertices i and j get the same color X, even though they are connected by an edge. Then, the example $v(i, j) \in S^-$ would be satisfied because it accepts the clause T_X, which is a contradiction. □

Note. This proof shows, as a side-effect, that three-term DNFs with negations are equivalent to three-term DNFs that do not have negated variables.

Note. Blum and Rivest show a similar learning hardness result on neural networks in their paper "Training a 3-Node Neural Network is NP-Complete" [BR92].

5.2 Occam Algorithms

Now, we discuss a notion similar to what we saw in the first lecture, that will show that learning algorithms with a small enough hypothesis size will generalize well to the full distribution if they perform well on a random sample of training data.

In general, there are several models of learning that will equivalently imply PAC learnability and offer practical routes to such algorithms:

1. Occam learning — short hypotheses (e.g., to learn DNF it is enough to find a hypothesis of size $m^{.99}$ that is consistent with m examples).

2. Mistake bounded — perceptron.

3. Weak learning — adaptive boosting.

To formalize a particular example of this, we will prove that a cardinality version of Occam learning implies PAC learnability. Let L be a learning algorithm, and let $\mathcal{H}_{n,m}$ be the set of hypotheses L can output for all concepts $c \in \mathcal{C}_n$, where $|S| = m$.

Theorem 5.2 (Occam, cardinality version)**.** *Suppose that for all m, n, for all concepts $c \in \mathcal{C}_n$, and for all sets S of m labelled examples of c, L outputs a hypothesis $h \in \mathcal{H}_{n,m}$ consistent with c. Then there exists a constant $B > 0$ such that for all $\epsilon < \frac{1}{2}$ and $\delta \leq 1$, given any distribution D on X_n and concept $c \in \mathcal{C}_n$, if L is given a set of m random examples from D and*

$$m \geq \frac{1}{B\epsilon}\left(\log_2 |\mathcal{H}_{n,m}| + \log_2\left(\frac{1}{\delta}\right)\right),$$

then with probability $1 - \delta$, h will have error less than ϵ on D.

Proof. Let h be bad if $\text{error}(h) > \epsilon$. For a fixed h, the probability that h is consistent with S is $(1 - \epsilon)^m$. Now we just note that the probability that at least one bad $h \in \mathcal{H}_{n,m}$ is consistent with S is at most $|\mathcal{H}_{n,m}|(1 - \epsilon)^m$. We want this to be at most δ, which is direct through some algebra and application of Bernoulli's inequality. □

5.3 An Application of Occam Learning

Consider the concept class $\mathcal{C}$ of conjunctions on n Boolean variables, where each variable is either present, present as negation, or not present in the expression. For example, one example of a concept for $n = 7$ could be $c = x_1\overline{x_3}x_7$. If our hypothesis class is the same, then the size of our hypothesis class $H_{n,m} = 3^n$, so by Theorem 5.2, if our number of labeled examples is at least

$$m \geq \frac{1}{B\epsilon}(n + \log(1/\delta)),$$

then with probability $1 - \delta$ our learning algorithm will have error less than ϵ on D.

Note. A second application of Occam learning will be k-decision lists, covered next lecture.

6 February 13

Today we continue from last week, using the Occam learning model to show that k-decision lists are PAC learnable. We also introduce mistake-bounded learning.

6.1 k-Decision Lists

Recall that Occam learning says that a short hypothesis which is consistent with data should believed. First, we'll use Occam learning to show that 1-decision lists are PAC-learnable.

Definition 6.1 (k-decision list)**.** A *k-decision list* is a concept that consists of a chain of conditions with fall-through. Each condition consists of a conjunction of up to k variables or their negations. To classify an example as true or false, the decision list iterates through each condition in order and finds the first one that evaluates to true, then outputs either 0 or 1 depending on the output corresponding to that condition.

An example of a 1-decision list is shown below. Notice how in this example, each of the conditions in the list are either single variables or their negations. In a 2-decision list, a condition might look like x_2x_3, and in a 3-decision list, you might have a condition like $x_5\overline{x_2}x_4$.

$$\begin{tikzcd} \overline{x_3} \arrow[r] \arrow[d] & x_2 \arrow[r] \arrow[d] & \overline{x_2} \arrow[r] \arrow[d] & 0 \\ 0 & 1 & 1 & \end{tikzcd}$$

We can notice immediately that k-decision lists are a stronger concept class than k-DNF, which are represented by the special cases of decision lists where all intermediate outputs are 1.

Example 6.2. We present the following algorithm for learning 1-decision lists.

1. Initialize three sets: S, the set of labeled examples, L, an empty decision list, and $Z = \emptyset$, a set of literals.
2. Enumerate all $\ell \in \{x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, \overline{x_n}\} \setminus Z$ until we find a literal ℓ such that:
   (a) $S_\ell = \{x \in S \mid x[\ell] = 1\}$ is nonempty, and
   (b) S_ℓ is either all true or all false.
3. Add decision ℓ to the decision list such that the "yes" branch has value 0 or 1 according to the value of all of the elements in S_ℓ. Then, update $S \leftarrow S \setminus S_\ell$.
4. Iterate until $S = \emptyset$, at which point our decision list is consistent with the starting S.

Note. This learning algorithm for 1-decision lists can be generalized to learn k-decision lists very easily. Rather than starting with a set of single-variable literals, we can enumerate over all $2^k \cdot \binom{n}{k}$ tuples of literals when finding the next condition.

Proposition 6.3. *If there exists a 1-decision list consistent with the starting examples S, the algorithm described in Example 6.2 will not fail.*

Proof. It suffices to show that as long as $S \neq \emptyset$, some literal ℓ making S_ℓ either all true or all false can be found. Call the original decision list DL*. Then, this comes directly from iterating over each condition $\ell \in$ DL* in order. We continue this until we reach the first ℓ such that $S_\ell \neq \emptyset$, which must satisfy the requirement, as desired. □

Thus, if we have data that is consistent with at least one decision list, our algorithm will always learn a new decision list that is consistent with the data. This allows us now to use the Occam learning hypothesis to analyze our algorithm.

Proposition 6.4. 1-*decision lists are Occam learnable.*

Proof. We first compute the size of the hypothesis class $\mathcal{H}_{n,m}$. First, we can take $n!$ choices for the ordering of the variables, then 2^n choices for whether each variable is negated, and 2^n choices for the intermediate outputs. The total size of the hypothesis class is

$$|\mathcal{H}_{n,m}| = n! \cdot 2^n \cdot 2^n.$$

Then, $\log(\mathcal{H}_{n,m}) = O(n \log n)$, so by the Occam bound (see Theorem 5.2), the number of examples we need is, for some constant k,

$$m \geq k\left(\frac{n}{\epsilon}\log n + \frac{1}{\epsilon}\log\frac{1}{\delta}\right).$$

□

Note. In all the examples of Occam learning that we've seen so far, the size of the hypothesis class $\mathcal{H}_{n,m}$ has not depended on m. However, we'll later on that m can be useful if we have a learning algorithm that classifies almost all, except a few of the training samples, and our hypotheses may contain special hard-coded cases. This would make the hypothesis size depend on m.

6.2 Mistake Bounded Learning

We will now move on to our second model of learning that will imply PAC learnability, known as the *mistake bounded* model. A classic example of this is the perceptron algorithm, which we will review.

Definition 6.5 (Online)**.** Suppose that we have a domain X_n, and a concept class $\mathcal{C}_n$. An algorithm is *online* if for any sequence $\underline{x}^1, \underline{x}^2, \ldots \in X_n$, the algorithm makes a prediction on each $\underline{x}^i$ and immediately is told $c(\underline{x}^i)$, before having seen $\underline{x}^{i+1}$, where $c \in \mathcal{C}_n$.

Definition 6.6 (Mistake bounded)**.** A online learning algorithm L is called $m(n)$-*mistake bounded* if for any, possibly infinite, sequence of samples given to the algorithm, it will have at most a finite number $m(n)$ of mistakes.

Example 6.7. Learning the concept class of intervals in $\mathbb{R}$ (see Example 2.1) cannot have any finite mistake bound, as there can always be an infinite number of unknown labels after any number of examples.

Example 6.8. The elimination learning algorithm for disjunctions (1-DNF) from negative examples has an $(n+1)$-mistake bound.

Proof. This is easy to show. The first negative example will remove at least n terms from the original disjunction with $2n$ terms. After this, each additional mistake will remove at least 1 term from the disjunction, so the mistake bound is $m(n) = n + 1$. □

Now, we cover a classic online mistake bounded algorithm called Littlestone's winner algorithm. Consider the concept class $\mathcal{C}_n$ of monotone disjunctions on n variables, using at most k literals. We present an algorithm that will learn this concept class with at most $2k\lceil \log_2 n \rceil + 1$ mistakes, if such a disjunction exists. This bound is useful when k is smaller than n.

The key insight in Littlestone's algorithm is the hypothesis class used in learning, where hypotheses are given by a linear inequality h of the form

$$\sum_{i=1}^{n} w_i x_i \geq n,$$

where the left-hand side has a 0-1 dependence on the value of each variable x_i. Then, for each sample $\underline{x}^j \in \{0,1\}^n$ that we classify incorrectly, we do one of the following updates to our hypothesis:

1. (Elimination). If $h(\underline{x}^j) = 1$ and $c(\underline{x}^j) = 0$, then we set $w_i = 0$ for all variables $x_i = 1$ in $\underline{x}^j$.

2. (Promotion). If $h(\underline{x}^j) = 0$ and $c(\underline{x}^j) = 1$, then we set $w_i \leftarrow 2w_i$ for all variables $x_i = 1$ in $\underline{x}^j$.

In the next lecture, we'll see a really nice, short proof of why Littlestone's algorithm has such a nice mistake bound.

7 February 18

Today we discuss Littlestone's winnow algorithm, and we prove general theorems about mistake-bounded learning.

7.1 Littlestone's Algorithm

Here we show that Littlestone's algorithm makes at most $2k\lceil \log_2 n \rceil + 1$ mistakes while learning the space of monotone disjunctions of at most k variables out of n.

Proposition 7.1. *Consider the concept class $\mathcal{C}_{n,k}$ of monotone disjunctions on $x_1, \dots, x_n$, given by Boolean formulas of the form*

$$x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k},$$

where the indices i_j may or may not be distinct. The number of mistakes made by Littlestone's online algorithm on $\mathcal{C}_{n,k}$ is is at most $2k\lceil \log_2 n \rceil + 1$.

Proof. First, we make some observations about the algorithm:

1. In each promotion, at least one of the relevant w_i is promoted.

2. No w_i is promoted once $w_i \geq n$, because it will always be recognized as a positive example.

3. A variable can be eliminated at most once, before its weight is set to 0.

Claim. The total number of promotion steps is at most $k \cdot \lceil \log_2 n \rceil$.

Proof. Note that each w_i can be promoted at most $\lceil \log_2 n \rceil$ times by observation 2. Then, the product of the weights of relevant variables cannot be greater than $(2^{\lceil \log_2 n \rceil})^k$. Each time a promotion step happens, this product at least doubles, so the claim follows. □

Claim. The total number of elimination steps is at most the number of promotion steps plus 1.

Proof. Note that each elimination reduces the sum of the weights by at least n. Also, each promotion only increases the sum of the weights by at most n. Since the initial sum of weights is n, and the weights are always nonnegative, we have the result. □

Finally, each mistake made by Littlestone's algorithm is either a promotion or an elimination, so the number of mistakes is bounded by $2k\lceil \log_2 n \rceil + 1$. □

Now consider the following variant of Littlestone's algorithm applied to a multiplicative perceptron model. Instead of simply doubling, we pick some $\alpha > 1$ and set $w_i \leftarrow w_i \cdot \alpha$ on promotion and $w_i \leftarrow w_i/\alpha$ on elimination (demotion). Suppose that there exists a classifier with double margin $\delta, \theta > 0$, i.e., there exist weights $\mu_1, \dots, \mu_n \geq \theta$ such that

$$\sum \mu_i x_i \geq 1$$

for all samples where $c = 1$ and

$$\sum \mu_i x_i \leq 1 - \delta$$

for all samples where $c = 0$. The number of mistakes is then bounded by at most

$$\frac{8}{\delta^2}\frac{n}{\theta} + \left(\frac{5}{\delta} + \frac{14\log\delta}{\delta^2}\right)\sum \mu_i.$$

7.2 General Theorems on Mistake Boundedness

Definition 7.2 (Conservative algorithm)**.** A mistake bounded learning algorithm is called *conservative* if it changes its hypothesis only after a mistake.

In other words, a conservative algorithm ignores inputs that it classifies correctly in the process of online learning, and it doesn't update any state based on this.

Proposition 7.3. *If L is a mistake bounded learning algorithm for learning $\mathcal{C}$, then there exists a conservative mistake bounded algorithm L^* that also learns $\mathcal{C}$ with the same mistake bound.*

Proof. Let L^* mimic L, except that after each correct prediction, it does not change h. Suppose that L^* makes m^* mistakes on an input sequence $\underline{x}^1, \ldots, \underline{x}^n$, and let the subsequence on which the m^* mistakes are made be $\underline{x}^{i_1}, \ldots, \underline{x}^{i_{m^*}}$. Then L^* run on this subsequence will still make m^* mistakes, since it is conservative.

However, L will make the same mistakes as L^* when run on this subsequence, as the two algorithms behave the same after they make a mistake. Thus, for each input sequence, if L^* makes m^* mistakes on this input sequence, then there exists a sequence such that L also makes m^* mistakes, so L^* has the same mistake bound as L. □

Finally, we connect mistake boundedness back to the main topic. The following proposition shows that if there exists a mistake bounded algorithm for learning a concept class $\mathcal{C}$, then $\mathcal{C}$ is also PAC learnable.

Proposition 7.4. *If L is an m-mistake bounded conservative algorithm for a given concept class and hypothesis class, then there exists a PAC-learning algorithm for the given concept class and hypothesis class.*

Proof. The following proof is from http://www.cs.cmu.edu/~ninamf/ML11/lect0908.pdf.

Consider the following algorithm: Run L on the data seen, halting if any hypothesis h survives for $\frac{1}{\epsilon}\ln(\frac{M}{\delta})$ subsequent examples. Return h as the output of the algorithm.

Note that this algorithm will terminate after $\frac{M}{\epsilon}\ln(\frac{M}{\delta})$ examples, because the number of mistakes is bounded by M. The probability that the algorithm accepts a hypothesis with error greater than ϵ is at most

$$M(1-\epsilon)^{\frac{1}{\epsilon}\ln(\frac{M}{\delta})} < M\frac{\delta}{M} = \delta.$$

Thus, the probability that the algorithm accepts a good hypothesis (of error at most ϵ) is at least $1-\delta$, so this algorithm PAC-learns the concept class. □

Proposition 7.5. *If L is an m-mistake bounded conservative algorithm, then if it has seen*

$$t = m + \left(\frac{m+1}{\epsilon}\right)\left(\log\left(\frac{1}{\delta}\right) + \log(m+1)\right)$$

examples from $\mathcal{D}$, then the hypothesis that is unchanged for the longest has error at most ϵ with probability at least $1-\delta$.

Proof. By the pigeonhole principle, the longest surviving hypothesis must be consistent with at least $\frac{t-m}{m+1}$ examples. Let X_i be the probability that the hypothesis that starts surviving after the i-th mistake has error greater than ϵ and survives for at least $\frac{t-m}{m+1}$ examples. The probability of this is at most

$$X_i \le (1-\epsilon)^{(t-m)/(m+1)} \le (1-\epsilon)^{\frac{1}{\epsilon}(\log(1/\delta)+\log(m+1))} \le \frac{\delta}{m+1}$$

Thus, taking a union bound over each of the $m + 1$ possible starting locations for the longest surviving hypothesis, the probability that it has large error is at most δ, so we have the result. $\square$

8 February 20

We were not present for this lecture. Topics covered included Vapnik–Chervonenkis dimension and the perceptron, which is a mistake bounded algorithm for learning linear separators between points in $\mathbb{R}^n$. For a discussion of VC dimension, see Section 9.3.

9 February 25

Today we analyze linear models in more detail, including kernel perceptrons and support vector machines.

9.1 Kernel Methods

Say that we have input variables given by $(x_1, \dots, x_n) \in \mathbb{R}^n$. Then we can fit a perceptron with various features. For example, the standard feature transformation gives us a hypothesis class parameterized by weights w_i, with linear hypothesis

$$\sum_{i=1}^{n} w_i x_i \geq c.$$

Through third-degree polynomial combinations of three features, we can create a more general hypothesis class parameterized by real numbers h_{ijk}, where hypotheses are of the form

$$\sum_{1 \leq i,j,k \leq n} h_{ijk} x_i x_j x_k \geq c.$$

The perceptron algorithm can be restated so that the only operations on examples are inner products on pairs of them. Recall that in the classical perceptron algorithm, in order to learn

$$\sum w_i x_i \geq 0,$$

we have a promotion step where we take $w_i \leftarrow w_i + x_i$, and a demotion step where $w_i \leftarrow w_i - x_i$. Let the sequence of inputs that the perceptron algorithm is given be $\underline{x}^1, \underline{x}^2, \dots \underline{x}^m$. Then, we can define two sets of indices

$$P = \{i : \underline{x}^i \text{ causes a promotion}\}, \; D = \{i : \underline{x}^i \text{ causes a demotion}\}.$$

This results in a weight vector given by

$$\underline{w}^m = \sum_{i \in P} \underline{x}^i - \sum_{i \in D} \underline{x}^i$$

Then, note that when testing $\underline{x}^{m+1}$, we are a computation to check if $\underline{x}^{m+1} \cdot \underline{w}^m \geq 0$, or equivalently,

$$\sum_{i \in P} \underline{x}^{m+1} \cdot \underline{x}^i - \sum_{i \in D} \underline{x}^{m+1} \cdot \underline{x}^i \geq 0$$

which only uses dot products of examples. We can see that in order to train a perceptron in $\mathbb{R}^n$ on a sequence of m examples, with this method that remembers sets of incorrect classifications, we must take m^2 inner products of pairs of examples.

Now, suppose that we want to learn a slightly more sophisticated linear separator on $\{z_{ij} \mid z_{ij} = x_i x_j\}$, which is a second-order polynomial basis. To train a perceptron using the above algorithm we need to do inner products on vectors of the form $\underline{z}'$ and $\underline{z}''$, where $\underline{z}'_{ij} = x'_i x'_j$ and $\underline{z}''_{ij} = x''_i x''_j$.

We can make a few nice algebraic manipulations in this form, to see that

$$\underline{z}' \cdot \underline{z}'' = \sum_{ij} z'_{ij} z''_{ij} = \sum_{i,j} x'_i x'_j x''_i x''_j = \left(\sum_i x'_i x''_i \right) \left(\sum_j x'_j x''_j \right) = (\underline{x}' \cdot \underline{x}'')^2.$$

This is equivalent to the original, but is much simpler to compute, as only one inner product is needed over a vector in $\mathbb{R}^n$, which takes $O(n)$ time to compute, versus $O(n^2)$ time.

In general, with m inputs and kernel basis of degree k polynomials, we take m^2 inner products, and each inner product costs n, for a total of $O(m^2 n)$ training time and final hypothesis size $O(mn)$. A standard perceptron algorithm (on the transformed feature space) would require $O(mn^k)$ running time and hypothesis size $O(n^k)$.

Note. This kernel perceptron algorithm exactly simulates the normal perceptron algorithm with a polynomial basis. One cannot recover the weights this way (in fact, there are $O(n^k)$ weights), but one can make predictions using this algorithm because we remember the sets P and D, and the hypothesis depends solely on inner products with elements of these sets.

Note. Since the kernel perceptron algorithm is exactly equivalent to finding a separating hyperplane on a higher dimensional space, the sample complexity is the same as if there were a high number of features (in this case, scales with m^k).

9.2 Support Vector Machines

Support vector machines are motivated by two main ideas.

1. There are other algorithms that only need inner products.
2. Let's find a linear separator with maximum margin.

Suppose that we had a set S of m positive and negative examples $\underline{x}_i$ in $\mathbb{R}^n$, each with labels $y_i \in \{-1, +1\}$. In other words,

$$S = \{(\underline{x}_i, y_i) \mid 1 \leq i \leq m,\ \underline{x}_i \in \mathbb{R}^n,\ y_i \in \{-1, +1\}\}.$$

A linear separator for S exists if and only if there exists a weight vector $\underline{v} \in \mathbb{R}^n$ such that

$$\forall i : y_i(\underline{v} \cdot \underline{x}_i) > 0.$$

Definition 9.1 (Margin)**.** Given a linearly separable dataset S, the *margin* of S is defined as

$$\max_{|\underline{v}|=1} \left\{ \min_i \{y_i(\underline{v} \cdot \underline{x}_i)\} \right\}.$$

We can broadly distinguish between three different algorithms for finding linear separators:

1. Perceptrons find some particular linear separator.
2. Linear programming finds some linear separator.
3. Support vector machines find the separator with the *largest margin.*

For more on support vector machines, see any introductory machine learning textbook. We won't cover the algorithm itself in too much depth here, but note that similar kernel methods can be used to generalize SVMs.

9.3 VC Dimension

Now let's once again discuss the VC dimension. This is the analog for infinite concept classes of $\log|H|$, which roughly defines the "complexity" of a class and lets us put bounds on number of samples needed to learn these classes.

Given an algorithm that exactly classifies training samples of m elements of a concept class $\mathcal{C}_n$ using hypotheses from $\mathcal{H}_{n,m}$, recall from Theorem 5.2 that we need about

$$\frac{1}{\epsilon}\left(\log|\mathcal{H}_{n,m}| + \log\left(\frac{1}{\delta}\right)\right)$$

training samples to achieve error at most ϵ on the full set with confidence $1-\delta$.

Let X be the domain, and consider $c : X \to \{0,1\}$, where c belongs to a concept class $\mathcal{C}$. Define the intersection of a concept c with a subset $S \subset X$ by

$$c \cap S = \{\underline{x} \mid c(\underline{x}) = 1,\ \underline{x} \in S\}.$$

Now, define the set of *dichotomies* of S in $\mathcal{C}$ to be

$$\Pi_{\mathcal{C}}(S) = \{c \cap S \mid c \in \mathcal{C}\}.$$

Likewise, we can obtain a rough measure of complexity by defining

$$\Pi_{\mathcal{C}}(m) = \text{max number of dichotomies of sets of size } m = \max_{|S|=m}\{|\Pi_{\mathcal{C}}(S)|\}$$

Definition 9.2 (Shattering)**.** If $\Pi_{\mathcal{C}}(S) = 2^{|S|}$, we say that $\mathcal{C}$ *shatters* S.

Definition 9.3 (Vapnik–Chervonenkis dimension)**.** The *VC dimension* of a concept class $\mathcal{C}$ over a domain X is the maximum size of a subset $S \subset X$ that is shattered by $\mathcal{C}$. In other words, this is the maximum m such that $\Pi_{\mathcal{C}}(m) = 2^m$.

This gives rise to two regimes of growth for the number of dichotomies, which have behavior described in the claim below.

Proposition 9.4 (Sauer-Shelah lemma)**.** *For a domain X and concept class $\mathcal{C}$ with VC dimension d, consider all sets of S of m examples from X. The maximum number of dichotomies of any set S is bounded by*

$$\Pi_{\mathcal{C}}(m) \le \sum_{i=0}^{d}\binom{m}{i} \le \left(\frac{me}{d}\right)^d.$$

In other words, the number of dichotomies initially grows exponentially in the size of S up to the VC dimension of $\mathcal{C}$. However, for sets S larger than the VC dimension, the number of dichotomies grows at most polynomially in d.

Example 9.5. Consider the domain $X = \mathbb{R}^2$, the Euclidean plane, with concept class $\mathcal{C}$ being all linear separators. The VC dimension of $\mathcal{C}$ in X is 3, as lines can shatter up to 3 points, but not 4. Thus, the number of dichotomies initially grows exponentially, with $\Pi_{\mathcal{C}}(1) = 2$, $\Pi_{\mathcal{C}}(2) = 4$, and $\Pi_{\mathcal{C}}(3) = 8$. However, with larger sets of points, the bound tells us that $\Pi_{\mathcal{C}}(m) = O(m^3)$, and in fact this is not tight, as $\Pi_{\mathcal{C}}(m) = O(m^2)$.

Recall from last class that we showed that the mistake bound for any online learning algorithm cannot be better than the VC dimension.

Theorem 9.6 (Lower bound on sample complexity)**.** *If the VC dimension of a concept class $\mathcal{C}$ is d, then to learn $\mathcal{C}$ to error ϵ with confidence at least* 13/14*, we need at least* $\frac{1}{32}\frac{d}{\epsilon}$ *examples.*

Proof. Since this is a proof of PAC-learnability, we want to show the example for any distribution over the inputs. There's a worst case that's actually pretty easy to guess here.

Let S be a shattered set of $\underline{x}^1, \ldots, \underline{x}^d$ for $\mathcal{C}$. Define $\mathcal{D}$ to have weight $8\epsilon/d$ for $\underline{x}^1, \ldots \underline{x}^{d-1}$. and weight $1-(d-1)8\epsilon/d$ on x^d. Choose a concept c randomly by choosing uniformly each shattering of the dataset. This distribution requires many samples to learn, as to distinguish between the concepts requires sampling each of the relevant shattered points. The rest follows from a Chernoff bound, as it is very rare that we collect all elements $\underline{x}^1, \underline{x}^2, \ldots \underline{x}^{d-1}$ in less than $o(d/\epsilon)$ samples. □

10 February 27

Today we prove the Sauer-Shelah lemma.

10.1 Proof of the Sauer-Shelah Lemma

Last class, we used gave the statement of an upper bound on the number of dichotomies of a set of m points, which is polynomial in the VC dimension of the domain. Here we present a proof of this statement, Proposition 9.4. Define the integer function $\Phi_d(m)$ recursively as follows:

$$\Phi_d(m) = \begin{cases} 1 & \text{if } d = 0 \text{ or } m = 0, \\ \Phi_d(m-1) + \Phi_{d-1}(m-1) & \text{otherwise.} \end{cases}$$

We can show by induction that a non-recursive form of $\Phi_d(m)$ is given by

$$\Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}.$$

This easily follows from Pascal's identity because

$$\sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d-1} \binom{m-1}{i} = \sum_{i=0}^{d} \left[\binom{m-1}{i} + \binom{m-1}{i-1} \right] = \sum_{i=0}^{d} \binom{m}{i}.$$

Then, there are two key statements involved with Sauer-Shelah:

1. $\Phi_d(m) \leq \left(\frac{me}{d}\right)^d$ if $m > d$.
2. If d is the VC dimension of $\mathcal{C}$, then for all m, $\Pi_{\mathcal{C}}(m) \leq \Phi_d(m)$.

Taken together, these imply Proposition 9.4. We will prove both statements below. Statement 2 is the important key observation, while Statement 1 is a standard counting bound.

Lemma 10.1 (Statement 2)**.** *Given a concept class $\mathcal{C}$ with VC dimension d over a domain X, then for all m, the maximum number of dichotomies of a subset of cardinality m from X is bounded by:*

$$\Pi_{\mathcal{C}}(m) \leq \Phi_d(m).$$

Proof. We proceed by strong induction on $m + d$.

- **Base Case:** If $m = 0$, there is only one set of points, the empty set, so $\Pi_{\mathcal{C}}(0) = 1 = \Phi_d(0)$.
- **Base Case:** If $d = 0$, no set of points is shattered, so $\Pi_{\mathcal{C}}(m) = 1$ because there is only one way of classifying any given point.

The main part of the proof is the inductive step. Our inductive hypothesis is that for all pairs of integers x, y such that $x + y \leq m + d - 1$, and concept classes $\mathcal{C}$ with VC dimension x, assume that

$$\Pi_{\mathcal{C}}(y) \leq \Phi_x(y)$$

Consider a set S of size m. Fix u to be any element of S. Note that

$$|\Pi_{\mathcal{C}}(S \setminus \{u\})| \leq \Pi_{\mathcal{C}}(m-1) \leq \Phi_d(m-1).$$

However, the size of the actual set of dichotomies $\Pi_{\mathcal{C}}(S)$ exceeds this by some amount Q, which equals the number of pairs of sets in $\Pi_{\mathcal{C}}(S)$ that differ only in whether u is a member. Let

$$\mathcal{C}' = \{c \in \Pi_{\mathcal{C}}(S) \mid u \notin c, \text{ and } c \cup \{u\} \in \Pi_{\mathcal{C}}(S)\}\}.$$

The missing quantity Q is exactly equal to $|\mathcal{C}'|$. But $\mathcal{C}'$ can now be treated as a concept class over the smaller set S! Furthermore, as a concept class, the VC dimension of $\mathcal{C}'$ is bounded by $d-1$, as any set $T \subset S$ shattered by $\mathcal{C}'$ has a corresponding set with one more element, $T \cup \{u\}$, which is shattered by $\mathcal{C}$. Thus, we conclude by observing that

$$|\Pi_{\mathcal{C}}(S)| \leq |\Pi_{\mathcal{C}}(S \setminus \{u\})| + |\Phi_{\mathcal{C}'}(S \setminus \{u\})| \leq \Phi_d(m-1) + \Phi_{d-1}(m-1) = \Phi_d(m).$$

□

Lemma 10.2 (Statement 1)**.** *For all $m > d$ integers, $\Phi_d(m) \leq \left(\frac{me}{d}\right)^d$.*

Proof. This will involve some bounding. Observe that because $d/m < 1$,

$$\begin{aligned} \left(\frac{d}{m}\right)^d \Phi_d(m) &= \sum_{i=0}^{d} \binom{m}{i} \left(\frac{d}{m}\right)^d \\ &\leq \sum_{i=0}^{d} \binom{m}{i} \left(\frac{d}{m}\right)^i \\ &\leq \sum_{i=0}^{m} \binom{m}{i} \left(\frac{d}{m}\right)^i \\ &= \left(1 + \frac{d}{m}\right)^m \leq e^d. \end{aligned}$$

The bound follows from dividing both sides of the inequality by $(d/m)^d$. □

Note. Today we proved upper bounds on the number of dichotomies (or lower bounds on the VC dimension), but it still remains to find bounds the reverse direction. This is also an interesting problem; we will discuss it in the future!

11 March 3

Today we state and prove a major result that shows that concept classes with finite VC dimension are PAC learnable, linking these notions together.

11.1 Finite VC Dimension Implies PAC Learnability

Let's now finish what we can say about the VC dimension.

Theorem 11.1 ([BEHW89])**.** *If $\mathcal{C}$ has VC dimension d (and obeys some measurability conditions) and L is any algorithm that for any sample S labeled according to some $c \in \mathcal{C}$ produces a hypothesis $h \in \mathcal{C}$ consistent with S, then L is a PAC-learning algorithm if applied to*

$$m \geq B\left(\frac{1}{\epsilon}\log\left(\frac{1}{\delta}\right) + \frac{d}{\epsilon}\log\left(\frac{1}{\epsilon}\right)\right)$$

samples, where B is some absolute constant.

Recall that Theorem 9.6 provides a lower bound on the sample complexity of learning from a Boolean function in terms of VC dimension. Intuitively, the hardest case comes when we have a distribution is supported on a shattered set of d objects, and where all but one of the samples is very rare (total probability about ϵ), so we must take many samples in order to college all labels of the set. There, the lower bound was asymptotically $\Omega(d/\epsilon)$.

This suggests that Theorem 11.1 will be nontrivial to prove, because it provides an upper bound that is asymptotically fairly close to the lower bound of d/ϵ samples. We provide a proof below, which will use some new machinery and combinatorial techniques.

11.2 Difference Regions and ϵ-Nets

Define the distance of $c_1, c_2 \in \mathcal{C}$ as $c_1 \,\Delta\, c_2 = \{x \in X \mid c_1(x) \neq c_2(x)\}$. This characterizes the regions on which two concepts disagree. Then we can define

$$\Delta_c = \{c \,\Delta\, c' \mid c' \in \mathcal{C}\},$$

which is the set of difference regions with respect to some concept $c \in \mathcal{C}$.

Lemma 11.2. *For all concepts $c \in \mathcal{C}$, the VC dimension of Δ_c is equal to that of $\mathcal{C}$.*

Proof. For all $S \subset X$, we can establish a bijection between dichotomies created by $\mathcal{C}$ in S and dichotomies created by Δ_c in S, which simply consists of taking the symmetric difference of the relevant dichotomy with $c \cap S$. This means that S is shattered by $\mathcal{C}$ if and only if S is shattered by Δ_c, and we are done. □

Next, define a function

$$\Delta_\epsilon(c) = \left\{h \,\Delta\, c \mid h \in \mathcal{C},\ \Pr_{x \sim \mathcal{D}}(x \in h \,\Delta\, c) \geq \epsilon\right\}.$$

The set $\Delta_\epsilon(c)$ contains the difference regions of hypotheses h that are at least ϵ far away from c. Now the idea is to consider *ϵ-nets*. A subset $S \subset X$ is called an ϵ-net for Δ_c if every region in $\Delta_\epsilon(c)$ is "hit" by a point in S. In other words, for all $r \in \Delta_\epsilon(c)$,

$$r \cap S \neq \emptyset.$$

The idea is that S provides a set of examples that shows whether a hypothesis is good or not. As long as h is very different from c, some sample in the ϵ-net of $\mathcal{C}$ will show this. In particular, if S is an ϵ-net for $\mathcal{C}$, and a learning algorithm outputs a hypothesis $h \in \mathcal{C}$ consistent with S, then $h \,\Delta\, c < \epsilon$ by definition.

Example 11.3. If $\mathcal{C}$ is the class of closed intervals in $[0, 1]$ and $\mathcal{D}$ is uniform, then the set $S = \{x = k\epsilon/2 \mid k \in \mathbb{Z}\}$ is an ϵ-net, because $c_1 \,\Delta\, c_2$ for $c_1, c_2 \in \mathcal{C}$ is a union of two intervals.

The key step to finish this proof is to show that we can obtain an ϵ-net of Δ_c with a small number of samples. We want to find a function $m(\mathrm{VC}(c), \epsilon, \delta)$, such that $\forall c \in \mathcal{C}$ and $S \subset X$ of size m randomly sampled according to $\mathcal{D}$, we will have an ϵ-net for Δ_c with confidence at least $1 - \delta$.

Consider any $r \in \Delta_\epsilon(c)$ where $r = c \,\Delta\, c'$ for some $c' \in \mathcal{C}$. Suppose $r \in \Delta_\epsilon(c)$ is fixed. We conduct the experiment of drawing a sequence of $2m$ examples $S_1 \cup S_2$, where S_1 is the set of the first m examples, and S_2 is the set of the last m examples.

Let A_r be the event that S_1 misses r, and let B_r be the event that A_r occurs and S_2 hits at least $\epsilon m/2$ times. Let A be the event that A_r occurs for some r, and let B be the event that B_r occurs for some r.

The hard part, then, is to show that we can hit every set S of size m. For any S we can discuss regions $r \in \Pi_{\Delta_\epsilon(c)}(S)$, i.e., dichotomies of S rather than X. Now, we can let U_r be the probability that $S = S_1 \cup S_2$ is drawn in some order and S_1 missed r, but S_2 hit r at least $\epsilon m/2$ times.

Consider $2m$ balls, colored so that ℓ are red, $2m - \ell$ are blue. If we split them up randomly into two urns having m balls each, then the probability that the red balls all end up in the second urn is bounded by

$$\Pr(\text{red balls all in second urn}) \le \frac{\binom{m}{\ell}}{\binom{2m}{\ell}} = \frac{m(m-1)\cdots(m-\ell+1)}{2m(2m-1)\cdots(2m-\ell+1)} \le 2^{-\ell}.$$

Therefore, U_r is at most $2^{-\epsilon m/2}$. We are now attempting to estimate the probability that B happens. The probability of B is at most the number of choices of r multiplied by U_r, which is at most

$$\Pi_{\mathcal{C}}(2m) \cdot 2^{-\epsilon m/2} \le \left(\frac{2em}{d}\right)^d \cdot 2^{-\epsilon m/2}.$$

The m stated makes this lower than δ, and we are done after some playing around with some algebra involving conditional probability.

12 March 5

Today we will cover a slightly different topic: multiplicative updates, or boosting. We got a taste for the flavor of this approach when discussing the winnow algorithm.

12.1 Multiplicative Updates – Best Expert

Example 12.1 (Best expert model)**.** Suppose that we have N experts, and we are given as input a sequence of examples $\underline{x}^1, \dots, \underline{x}^T$. On each example, each expert makes a prediction. We want a method of combining predictions so that for any sequence of examples, our combined prediction cannot be much worse than the best expert.

Formally, suppose that an expert at time t suffers an loss $\ell_i^t \in [0,1]$, e.g., $\ell_i \in \{0,1\}$. We do not know ℓ_i^t until after making predictions for timestep t. Our online algorithm should output a *combiner*, which is a probability $p_i^t \geq 0$ for each expert i at each time t such that for all $1 \leq t \leq T$,

$$\sum_{i=1}^{n} p_i^t = 1.$$

The total loss of the combiner at time t is

$$L^t = \sum_{i=1}^{N} p_i^t \ell_i^t.$$

Our goal is to minimize the cumulative loss of our combiner $L_A = \sum_{t=1}^{T} L^t$, relative to the minimum of the losses $L_i = \sum_{t=1}^{T} \ell_i^t$ for all experts i.

Proposition 12.2 ([FS97])**.** *There is an online algorithm outputting a combiner* Hedge *with parameter $\beta \in (0,1)$ such that for all experts i,*

$$L_{\text{Hedge}(\beta)} \leq \frac{1}{1-\beta}\left(L_i \ln\left(\frac{1}{\beta}\right) + \ln N\right).$$

Proof. We initialize weights $w_1^1, \dots, w_N^1 = \frac{1}{N}$, where $w_i^t \in [0,1]$ represents the weight of the i-th expert at time t. Then for each value of t in $1, 2, \dots, T$, we do the following:

1. Output combiner prediction weights

$$p_i^t = \frac{w_i^t}{\sum_{i=1}^{N} w_i^t}.$$

2. Obtain information that the experts suffer losses $\ell_1^t, \dots, \ell_N^t$ from the environment during this timestep, so our combiner suffers loss $\underline{p}^t \cdot \underline{\ell}^t$.

3. Update weights multiplicatively for the next timestep by setting $w_i^{t+1} = w_i^t \beta^{\ell_i^t}$.

Observe that at time T, we will have

$$w_i^{T+1} = w_i^1 \beta^{\sum_{t=1}^{T} \ell_i^t}.$$

Our first step will be to upper bound $L_{\text{Hedge}(\beta)}$ in terms of the sum of the final weights. We will prove this with a couple of lemmas.

Lemma 12.3. *For all $\gamma \in [0,1]$ and $\alpha \in [0,1]$,*

$$\alpha^\gamma \le 1 - (1-\alpha)\gamma.$$

Proof. This is a quick consequence of Jensen's inequality on $f(x) = \alpha^x$. □

Lemma 12.4. *The loss of the hedge algorithm is bounded by*

$$L_{\text{Hedge}(\beta)} \le \frac{1}{1-\beta}\left[-\ln\left(\sum_{i=1}^N w_i^{T+1}\right)\right].$$

Intuitively, this means that if the experts perform sufficiently poorly, the sum of final weights will be small, and so the loss of our algorithm can be large.

Proof. This is technical and involves some algebra. Note that by Lemma 12.3

$$\begin{aligned}
\sum_{i=1}^N w_i^{t+1} &= \sum_{i=1}^N w_i^t \beta^{\ell_i^t} \\
&\le \sum_{i=1}^N w_i^t (1 - (1-\beta)\ell_i^t) \\
&= \left[\sum_{i=1}^N w_i^t\right] - (1-\beta)\left[\sum_{i=1}^N w_i^t \ell_i^t\right] \\
&= \left[\sum_{i=1}^N w_i^t\right] - (1-\beta)\left[\sum_{i=1}^N p_i^t \ell_i^t \left(\sum_{j=1}^N w_j^t\right)\right] \\
&= \left(\sum_{i=1}^N w_i^t\right)\big(1 - (1-\beta)(\underline{p}^t \cdot \underline{\ell}^t)\big).
\end{aligned}$$

Iterating this inequality for $t = 1, \dots, T$ gives

$$\begin{aligned}
\sum_{i=1}^N w_i^{T+1} &\le \left(\sum_{i=1}^N w_i^t\right) \cdot \prod_{t=1}^T \big(1 - (1-\beta)(\underline{p}^t \cdot \underline{\ell}^t)\big) \\
&\le \prod_{t=1}^T e^{-(1-\beta)(\underline{p}^t \cdot \underline{\ell}^t)} \\
&= e^{-(1-\beta)\sum_{t=1}^T (\underline{p}^t \cdot \underline{\ell}^t)} \\
&= e^{-(1-\beta) L_{\text{Hedge}(\beta)}}.
\end{aligned}$$

The lemma immediately follows from rearranging this last inequality. □

Finally, applying Lemma 12.4, we see that

$$\begin{aligned}
L_{\text{Hedge}(\beta)} &\le \frac{1}{1-\beta}\left[-\ln\left(\sum_{i=1}^N w_i^1 \beta^{L_i}\right)\right] \\
&= \frac{1}{1-\beta}\left[-\ln\left(\frac{1}{N}\right) - \ln\left(\sum_{i=1}^N \beta^{L_i}\right)\right].
\end{aligned}$$

Since $\ln(x)$ is an increasing function, we can replace the sum of all β^{L_i} above with simply the maximum of all β^{L_i}. This means that for any i,

$$\begin{aligned} L_{\text{Hedge}(\beta)} &\leq \frac{1}{1-\beta}\left(-\ln\left(\frac{1}{N}\right) - \ln\left(\beta^{L_i}\right)\right) \\ &\leq \frac{1}{1-\beta}\left(\ln N + L_i \ln\left(\frac{1}{\beta}\right)\right). \end{aligned}$$

The result follows directly. □

Note. Although this algorithm is algebraically nice, it turns out that it isn't very useful in practice. We will need to do a little more to actually obtain a practical algorithm.

12.2 Weak Learning and Boosting

Now we introduce a slightly more useful application in a similar algorithm, which is called *boosting*.

Definition 12.5 (Weak PAC learning)**.** We call a concept class $\mathcal{C}$ *weakly PAC-learnable* by a hypothesis class $\mathcal{H}$ if there exists an algorithm L that for fixed polynomials p, q, and access to labeled examples from $\mathcal{D}$, outputs in time $p(n, \text{size}(c))$ a hypothesis $h \in \mathcal{H}$ that with probability at least $q(n, \text{size}(c))^{-1}$ satisfies

$$\text{error}(h) \leq \frac{1}{2} - \frac{1}{p(n, \text{size}(c))} \leq \epsilon.$$

Proposition 12.6 ([Sch90])**.** *If $\mathcal{C}$ is weakly PAC learnable by a polynomially evaluatable hypothesis class $\mathcal{H}$, then $\mathcal{C}$ is PAC learnable by the same $\mathcal{H}$.*

Intuitively, this proof strongly uses the idea that we can find a weak learner for ANY distribution $\mathcal{D}$ on X. The idea is to initially train one weak learner, then train more weak learners iteratively on the samples that we do not answer correct.

13 March 10

Today we cover a boosting algorithm named AdaBoost (short for adaptive boosting). In future classes, our plan is to use Zoom, as the Harvard campus will be closed due to COVID-19. Topics will include learning in the presence of malicious errors, noise, queries, crypto hardness, deep learning, and project presentations.

13.1 AdaBoost

We now cover an algorithm that converts a weak learning algorithm into a fully PAC algorithm. This will be a slight modification from our best experts model from last lecture (Proposition 12.2), but it instead combines weak learners into a strong learner.

Suppose that we have a sequence S of N labeled examples $(\underline{x}_i, y_i) \in X \times \{0,1\}$, for each sample index $i = 1, \ldots, N$. First initialize weights $w_i^1 = \mathcal{D}(i)$ according to some arbitrary distribution $\mathcal{D}$ supported on S. Given an algorithm **WeakLearn** and a parameter T representing the number of times the weak algorithm should be applied, AdaBoost applies the following procedure for each iteration $t = 1, \ldots, T$.

1. Set $p_i^t = w_i^t \Big/ \sum_{j=1}^N w_j^t$.
2. Call **WeakLearn** with distribution $\underline{p}^t$ over S, and retrieve a weak hypothesis $h_t : X \to \{0,1\}$.
3. Calculate the error of h_t, which is given by
$$\epsilon_t = \sum_{i=1}^n p_i^t \cdot |h_t(x_i) - y_i|.$$
4. Set $\beta_t = \dfrac{\epsilon_t}{1-\epsilon_t}$.
5. Update weights so that
$$w_i^{t+1} = w_i^t \cdot \beta_t^{1-|h_t(x_i)-y_i|}.$$

Finally, the output of the boosting algorithm is a hypothesis h given by

$$h(x) = \begin{cases} 1 & \text{if } \displaystyle\sum_{t=1}^T \left[\log_2\left(\frac{1}{\beta_t}\right)\right] h_t(x) \geq \frac{1}{2}\sum_{t=1}^T \log_2\left(\frac{1}{\beta_t}\right), \\ 0 & \text{otherwise.} \end{cases}$$

The intuition behind this math is that β_t represents a weighting for the importance of each hypothesis, while $\underline{w}^t$ is decreased for specific samples that a weak hypothesis classifies correctly. This means that the samples that are consistently classified incorrectly have higher weight, while those that are classified correctly have exponentially decreasing weight.

13.2 Analyzing AdaBoost

We will prove a series of claims to analyze the error complexity of AdaBoost. The final result will imply Proposition 12.6.

Lemma 13.1. *The hypothesis outputs $h(x) = 1$ if and only if*

$$\prod_{h_t(x)=1} \left(\frac{1}{\beta_t}\right) \geq \left(\prod_{t=1}^{T} \frac{1}{\beta_t}\right)^{1/2}.$$

Equivalently, for all i, if $h(x_i)$ is a mistake, then

$$\prod_{t=1}^{T} \beta_t^{-|h_t(x_i)-y_i|} \geq \prod_{t=1}^{T} \left(\frac{1}{\beta_t}\right)^{1/2}.$$

Proof. This follows from the definition of the output hypothesis as a simple restatement. □

Lemma 13.2 ([FS97])**.** *After running AdaBoost, the final weights satisfy*

$$\sum_{i=1}^{N} w_i^{T+1} \leq \prod_{i=1}^{T} [1 - (1 - \epsilon_t)(1 - \beta_t)].$$

Proof. Technical proof, omitted. See the paper. This is useful for proving the next theorem. □

Theorem 13.3 ([FS97])**.** *If successive calls of* ***WeakLearn*** *generate weak hypotheses with errors $\epsilon_1, \ldots, \epsilon_T$, then the error of h on the original distribution $\mathcal{D}$ is at most*

$$2^T \prod_{i=1}^{T} \sqrt{\epsilon_t(1 - \epsilon_t)} = \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2},$$

where $\epsilon_t = \frac{1}{2} - \gamma_t$.

For example, if $\gamma = 1/n$ as in a standard weak learner error model, then $\epsilon \leq \left(1 - \frac{4}{n^2}\right)^{T/2}$. Morally, the key idea that makes AdaBoost useful is that our error decreases exponentially in T, while the VC dimension grows linearly, so we will have relatively fast convergence.

Note. Why is boosting so effective in practice? Unlike other algorithms we have discussed so far (such as the best experts model), boosting works even if we do not know if h is a weak or strong learner, and it continues to improve even after h is perfect. If you graph generalization error compared to the number of parameters, most learning algorithms have an initial regime with few parameters where they sharply decrease in error (bias error), but after adding too many parameters, the error once again rises (variance). However, boosting does not have this second regime and continues to improve after many iterations.

Note. Boosting tends to work well in a lot of cases and is very data efficient, which makes it hard to beat in practice. However, it can sometimes be confused by applying the right noise.

14 March 24

Today we are discussing error models in PAC learning. Error models are important because practical learning algorithms must be able to deal with errors in the labeled input. This material is covered in Chapter 5 of [KV94b].

14.1 Models of Error in PAC Learning

We consider a few modes of errors, each with a "badness" parameter $0 < \eta < 1$:

1. *Malicious Noise:* With probability $1 - \eta$, the oracle outputs a correct sample $(\underline{x}, \ell)$, where $\underline{x}$ is an example drawn from $\mathcal{D}$, and $\ell = f(\underline{x})$. With probability η, it outputs $(\underline{x}, \ell)$, where $\underline{x}, \ell$ are chosen by an adversary with knowledge of $f, \mathcal{D}$, current state of learner, etc.

2. *Malicious Classification Noise:* This model is the same as above, except $\underline{x}$ is taken from $\mathcal{D}$ even in the error case, where the adversary can only choose ℓ.

3. *Random Classification Noise:* Outputs $(\underline{x}, \ell)$ where $\underline{x}$ is produced by the example oracle, and with probability $1 - \eta$, we have the correct class $\ell = f(x)$, while with probability η we have $\ell \neq f(x)$.

4. *Instance Noise:* This is the noise model which has received attention in recent times. In this case, we add noise to $\underline{x}$, not ℓ. For example, we can imagine photos that are taken of dogs, training a machine learning model on those, and then testing the model on pictures of dogs with a few pixels changed.

To talk about noise, we have a couple of general facts about learning in the presence of errors.

1. Random classification noise up to $\eta = \frac{1}{2} - \epsilon$ can be overcome for a certain complexity class SQ that we will discuss soon, which is a subset of PAC.

2. Random classification noise outside SQ may introduce conjectured intractability, as seen in the problem [Reg09] (2018 Gödel Prize). For example, linear forms modulo 2 are clearly PAC learnable, by taking Gaussian elimination in $\mathbb{F}_2$. However, in the presence of errors, this problem is the basis of cryptography schemes that are resistant to quantum computation.

Proposition 14.1. *For the concept class $\mathcal{C}$ of intervals, consider the scenario where positive examples are returned from $\mathcal{D}^+$ with probability $1 - \eta$, while arbitrary adversarial examples are chosen with probability η. Then,*

$$E^{\epsilon,\delta}_{\text{MAL},+} = \Omega(\epsilon).$$

Proof. Let the error rate be $\eta = \epsilon/4$. Take $m = 1/\eta = 4/\epsilon$ examples. Then with probability $1/e$, there are no errors in the examples. Then, you can simply use take the minimum and the maximum as usual and hope that nothing malicious has been sent. This implies that the maximum tolerable error is at least $\Omega(\epsilon)$, as desired. □

14.2 Statistical Query Leraning

Another learning model (weaker than PAC learning) is *statistical query learning*, a model from Kearns, where we can ask questions of the form $\chi : X \times \{0, 1\} \to \{0, 1\}$ which is a *criterion* on labelled examples. Then the oracle returns an estimate of the probability P_χ that $\chi = 1$. For example one query criterion could be $X(\underline{x}, \ell) = 1 \iff x_3 = 0 \wedge \ell = 1$.

Definition 14.2 (Statistical query)**.** A *statistical query* is an oracle $\mathrm{STAT}(c, \mathcal{D})$ that accepts queries (χ, τ), where χ is a criterion on labelled examples on which the probability of occurrence in $\mathcal{D}$ is sought, and τ $(0 < \tau < 1)$ is the additive error tolerance. Then, the oracle returns some probability p, such that

$$\left| p - \Pr_{x \sim \mathcal{D}}[\chi(\underline{x}, c(\underline{x})) = 1] \right| = |p - P_\chi| < \tau.$$

What is an SQ learning algorithm? Before we present the definition, here is a motivating example for the class of monotone conjunctions. A criterion is of the form $\chi_i(\underline{x}, \ell) = 1 \iff x_i = 0 \wedge \ell = 1$ for $i = 1, \ldots, n$ (these are a set of queries). Each query that we make is then of the form $(\chi_i, \epsilon/2n)$ to the oracle STAT.

Proposition 14.3. *To learn monotone conjunctions, there is a learning algorithm that outputs a hypothesis after n statistical queries the oracle. Written as a formula, the hypothesis h is*

$$h = \bigwedge_{i=1}^{n} \{x_i \mid (\chi_i, \epsilon/2n) \text{ returns } p < \epsilon/2n\}.$$

Proof. We present a proof that when $\tau < \frac{\epsilon}{2n}$, this hypothesis always has error less than ϵ.

(i) If x_i is in the monomial, you will never have $\chi_i(\underline{x}, \ell) = 1$, because it is impossible for $\ell = 1$ when $x_i = 0$. Hence STAT, which is guaranteed to have error less than $\epsilon/2n$ will return a value $p < \epsilon/2n$, and the variable will be included.

(ii) For x_i not in the monomial, then if it is included by mistake in the hypothesis h, we may have false negatives. Our error τ implies that the probability of $x_i = 0$ while $\ell = 1$ is less than ϵ/n. Then, by a union bound, the total error probability is at most

$$\sum_{i=1}^{n} \Pr(x_i = 0 \wedge \ell = 1) < \epsilon.$$

□

Note that the Perceptron algorithm, which we have covered previously, is not SQ learnable. This is because the Perceptron algorithm relies on very specific individual examples that modify the weight parameters, giving a conservative mistake bound.

Definition 14.4 (SQ learning)**.** Let $\mathcal{C}$ be a concept class and $\mathcal{H}$ be a representation class over X. Then $\mathcal{C}$ is learnable from statistical queries using $\mathcal{H}$ if there exists a learning algorithm L and polynomials p, q, and r so that $\forall c \in \mathcal{C}$, for all $\mathcal{D}$ over X, and for all $0 < \epsilon < 1/2$, if L is given access to $\mathrm{STAT}(c, \mathcal{D})$ and the value of ϵ, then:

- For every query (χ, τ) made by L, χ can be evaluated in time $q(1/\epsilon, n, \mathrm{size}(c))$, and $1/\tau$ is upper bounded by $r(1/\epsilon, n, \mathrm{size}(c))$.
- L will halt in time bounded by $p(1/\epsilon, n, \mathrm{size}(c))$.
- L will output a hypothesis $h \in \mathcal{H}$ with $\mathrm{error}(h) < \epsilon$.

Note that this definition is “simpler” than PAC learning in the sense that there is no confidence parameter δ.

Proposition 14.5. *If a concept class $\mathcal{C}$ is SQ learnable using $\mathcal{H}$, then it is PAC learnable by $\mathcal{H}$.*

Proof. In the PAC learning scenario, we can simulate calls of STAT for the SQ learning algorithm by taking enough calls of the oracle $\mathrm{EX}(c, \mathcal{D})$ so that estimates of P_χ are accurate to within tolerance bounds τ, with high enough probability. This requires $\log(1/\tau)$ examples by a Chernoff bound, which is polynomial (in fact, log-polynomial). □

Proposition 14.6. *If a concept class $\mathcal{C}$ is SQ learnable using $\mathcal{H}$, then it is PAC learnable with random classification noise when $\eta < \frac{1}{2} - 1/\operatorname{poly}(1/\epsilon, 1/\delta, n, \mathrm{size}(c))$.*

Proof. The key idea is to consider what we can deduce from a noisy source of labelled examples. Suppose we consider a criterion such as $\chi(\underline{x}, \ell) = 1$ if $\ell = 0$ and $x_3 = 1$. In the case where $x_3 = 1$, we have that $\chi(\underline{x}, 0) \neq \chi(\underline{x}, 1)$. In the case where $x_3 = 0$, we have that $\chi(\underline{x}, 0) = \chi(\underline{x}, 1)$.

Given any criterion such as χ, we can estimate the probability p_1 such that $\chi(\underline{x}, 0) \neq \chi(\underline{x}, 1)$, as well as $p_2 = 1 - p_1$, despite the noise η from the oracle $\mathrm{EX}(c, \mathcal{D})$. Once we can estimate p_1 and p_2, this is enough to learn a PAC hypothesis, as discussed in more detail in the next lecture. □

Note. Although the definition of statistical queries in *SQ* seemed unmotivated at first, we can now understand why. The reason why we defined the criterion to be $\chi : X \times \{0, 1\} \to \{0, 1\}$ for inputs $(\underline{x}, \ell)$, not just for $(\underline{x}, c(\underline{x}))$, is because we need to support noisy examples. We do not always know for certain the classification $c(\underline{x})$ of the input.

15 March 26

Today we continue discussing statistical query learning.

15.1 Statistical Query Learning (cont.)

Last lecture, we introduced a useful framework of learning by statistical queries, where we give a criterion whose probability (according to some distribution of examples) is estimated by an oracle. This class SQ can represent useful problems such as stochastic gradient descent and backpropagation, but we also saw its limitations [Reg09].

A more elementary way to see the limitations of SQ learning is as follows. A theorem of Kearns shows that parity is not SQ learnable. One intuition: if the hidden function is, say, $x_1 \oplus x_5 \oplus x_7$, and we make a wrong guess, the correlation will be 50% either way. Contrast this to Proposition 14.3, which shows that the class of monotone conjunctions is in SQ.

The motivating question for SQ-learning is: from a noisy source of labeled examples of $c \in \mathcal{C}$, what can we deduce? We can try to answer this statistical query given the value of the things we have. The following provides more details related to Proposition 14.6.

For some values we have $\chi(\underline{x}, 0) \neq \chi(\underline{x}, 1)$, and in some cases we have $\chi(\underline{x}, 0) = \chi(\underline{x}, 1)$. The criterion is defined $\chi : X \times \{0,1\} \to \{0,1\}$, not just for $x \in X : (\underline{x}, c(x)$ which would be sufficient for noise free examples, that for all $(\underline{x}, \ell)$, $\ell \in \{0,1\}$, as is needed for noisy examples.

Suppose that we have available a source $EX^{\eta}(c, \mathcal{D})$ of examples distributed according to $\mathcal{D}$ with a rate η of random classification noise, which we assume is known for now. We define $P_r^{\eta}(\chi = 1)$ to be equal to the quantity $P_r(\chi(\underline{x}, \ell) = 1)$ when the examples come from the noisy source. We also define probabilities $p_1 = \Pr(\chi(\underline{x}, 0) \neq \chi(\underline{x}, 1))$, and $p_2 = 1 - p_1$.

Lemma 15.1. *For any η,*

$$\Pr_{\mathcal{D}}(\chi = 1) = p_1 \cdot (\Pr^{\eta}_{\mathcal{D}_1}(\chi = 1) - \eta)/(1 - 2\eta) + \Pr^{\eta}_{\mathcal{D}}(\chi = 1 \wedge \underline{x} \in X_2),$$

where $\mathcal{D}_1$ is the conditional distribution of $\mathcal{D}$ where $\chi(\underline{x}, 0) \neq \chi(\underline{x}, 1)$ supported on the set X_1, while $\mathcal{D}_2$ is the conditional distribution where $\chi(\underline{x}, 0) = \chi(\underline{x}, 1)$ supported on the set X_2.

Proof. Observe that

$$\Pr_{\mathcal{D}}(\chi = 1) = p_1 \cdot \Pr_{\mathcal{D}_1}(\chi = 1) + \Pr^{\eta}_{\mathcal{D}_2}(\chi = 1 \wedge \underline{x} \in X_2).$$

However, note that

$$\begin{aligned}\Pr^{2}_{\mathcal{D}_1}(\chi = 1) &= (1 - \eta) \Pr_{\mathcal{D}_1}(\chi = 1) + \eta(1 - \Pr_{\mathcal{D}_1}(\chi = 1)) \\ &= \eta + (1 - 2\eta) \Pr_{\mathcal{D}_1}(\chi = 1).\end{aligned}$$

Solving this equation for $\Pr_{\mathcal{D}_1}(\chi = 1)$ and plugging back in yields the result. □

Lemma 15.2. *The three quantities needed in Lemma 15.1 can all be estimated from noisy examples in $EX^{\eta}(c, \mathcal{D})$, when $\eta < \frac{1}{2}$.*

Proof. We describe how to estimate each quantity:

- p_1. This only depends on $\underline{x}$ and not on $c(\underline{x})$, so we can obtain an unbiased Monte Carlo estimate for it. This is because the noisy samples from $EX^{\eta}(c, \mathcal{D})$ still provide us with the correct distribution of $\underline{x} \sim \mathcal{D}$.

- $\Pr^{\eta}_{\mathcal{D}}(\chi = 1 \wedge \underline{x} \in X_2)$. This can be estimated immediately by sampling from $\mathcal{D}$ with a slightly modified criterion.
- $\Pr^{\eta}_{\mathcal{D}_1}(\chi = 1)$. This can be estimated from samples from $\mathcal{D}$ with a slightly modified criterion and dividing by p_1, since

$$\Pr^{\eta}_{\mathcal{D}_1}(\chi = 1) = \frac{1}{p_1} \cdot \Pr^{\eta}_{\mathcal{D}}(\chi = 1 \wedge \underline{x} \in X_1).$$

□

This concludes the justification for our claim that problems in SQ are also PAC learnable with random classification errors.

15.2 Darwinian Evolution as Stastical Query Learning

One application of learning with errors can be seen in Darwinian evolution, which we can see as a real-life example of statistical query learning. Each example $\underline{x} \in X$, is a different situation that can occur, such as in the concentration of the proteins in a cell, while $\mathcal{D}$ is a distribution over them. Suppose your DNA makes you do $d : X \to \{0, 1\}$, an on/off switch for each of the situations.

There is an *ideal function* $c : X \to \{0, 1\}$ that determines which set of actions is better to do (e.g. whether to express a gene). Consider the criterion $\chi(\underline{x}, \ell) = 1$ when $c(\underline{x}) = d(\underline{x})$, where $d(\underline{x})$ is the function realized by an evolving entity, such as a species. Then, the statistical query oracle $\text{STAT}(d, \mathcal{D})$ is effectively equivalent to the "fitness" of an organism, or how often it survives in its environment.

Thus, statistical queries such as this one can be seen as the objective function for an evolutionary algorithm running in a noisy environment. For more on this interpretation, see [Mad16].

15.3 Learning using MAT Oracles

Yet another general learning model is learning with oracles. We still want to learn an unknown concept $c \in \mathcal{C}$, but the learner obtains information by asking questions of the two following forms:

- (Membership). The learner chooses an $\underline{x} \in X$. The oracle answers the query by responding with whether $c(\underline{x}) = 0$ or $c(\underline{x}) = 1$.
- (Equivalence). The learner chooses a hypothesis $h \in H$. The oracle either answers yes if the hypothesis is the correct, and in the other case outputs a mistake $\underline{x} \in X$ such that $c(\underline{x}) \neq h(\underline{x})$.

Note. It's hard to delineate which questions are reasonable: e.g. what is the i-th bit of the description of h? There are other ways of giving the learner power, e.g., active learning, where the learner cannot suggest an $\underline{x}$, but given a sample $S \subseteq X$ chosen according to $\mathcal{D}$, it can choose a subset $S' \subseteq S$ to label.

Definition 15.3 (Minimally adequate teacher)**.** A *minimally adequate teacher (MAT)* is an efficient learning algorithm to which membership and equivalence are available. There are no random examples, or a distribution. If the algorithm is deterministic, it learns with certainty.

Just like "Twenty Questions" where people ask queries dynamically, oracle learning is stronger than static question-answering because each each question can depend online on the previous questions and responses.

Proposition 15.4 ([AS94]). *Monotone DNF is learnable with the MAT model.*

Proof. The idea is to consider the “prime implicants” of the DNF expression, which are maximal elements of the poset of subsets generated by c. We will cover this in more detail next lecture. □

Note that monotone DNF is not known to be PAC learnable, so this is a stronger result.

16 March 27

The following notes are from a virtual talk by Moritz Hardt (UC Berkeley), hosted by the Harvard ML Theory group, titled "Performative Prediction."

> When predictions support decisions they may influence the outcome they aim to predict. We call such predictions performative; the prediction influences the target. Performativity is a well-studied phenomenon in policy-making that has so far been neglected in supervised learning. When ignored, performativity surfaces as undesirable distribution shift, routinely addressed with retraining. We develop a risk minimization framework for performative prediction bringing together concepts from statistics, game theory, and causality. A conceptual novelty is an equilibrium notion we call performative stability. Performative stability implies that the predictions are calibrated not against past outcomes, but against the future outcomes that manifest from acting on the prediction. Our main results are necessary and sufficient conditions for the convergence of retraining to a performatively stable point of nearly minimal loss. In full generality, performative prediction strictly subsumes the setting known as strategic classification. We thus also give the first sufficient conditions for retraining to overcome strategic feedback effects. Joint work with Juan C. Perdomo, Tijana Zrnic, and Celestine Mendler-Dünner. Arxiv link: https://arxiv.org/abs/2002.06673

16.1 History of Statistical Decision-Making

Suppose that given a list of people in each age group, we are given a distribution of total lifespans. Edmond Halley did exactly this in the 18th century by building a "life table," representing the number of people at age i. This was the birth of statistical decision making.

An annuity is a life annuity. Knowing a life table is used to price life annuities. The price of the annuity at age x is the *expected* sum of discounts fixed annual payment for the rest of a person's life. For example, you want

$$p_x = \sum_i p[\text{death at age } x + i] 0.95^i$$

This was Halley's statistical model. It's been 333 years of consequential decisions from data. Halley built a statistical model for decision making. This has been used for centuries with varying degrees of rigor. In the 20th century, statistics formalized and vastly extended it. The current ML / AI wave pushes statistical learning into an ever-increasing range of domains: health, finance, insurance, employment, and education.

The standard view of learning and decision-making is that data is used to build a model, which then makes decisions. We sometimes forget the origin of the the model.

> "Technologies are developed and used within a particular social, economic, and political context. They arise out of a social structure.
>
> "Context is not a passive medium but a dynamic counterpart. The responses of people, individually and collectively, and the responses of people, individually, and collectively, and the responses of nature are often underrated in the formulations of plans and predictions."

In 1696, England's King William III seeks to tax wealth, but he first needs a way to know one's wealth. King William III introduced a tax based on the number of windows. The idea spreads to France, Spain, and Scotland. France's window tax was only removed in the early 20th century.

One row of houses in Edinburgh features no bedroom windows at all. People learned to adapt and use less windows, so tax revenue fell. This is an example of *Goodhart's law*, which states that "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes" (Charles Goodhart, 1975).

Definition 16.1 (Performative)**.** When making a model, often some observed input data is used to trigger an action, which then modifies the pattern of the observed data. This phenomenon is called *performativity.*[2]

Let's think about performativity. The hypothesis is that *prediction is performative.* The prediction triggers an action that changes the distribution of the outcome variable that we're trying to predict. Performtivity is a well-known concept in the social sciences, economics ("performativity hypothesis"), and so on.

Let's give an example of credit default prediction. It's possible that the bank estimates "high" default risk, and sets a high interest rate. As a result, the customer's default risk further increases.

This is called a "self-fulfilling prophecy." An example is in traffic prediction: there are two routes from San Francisco to Mountain View, and Google Maps can help commuters by recommending the shortest route given current traffic. However, this is a self-negating prophecy, where making predictions influences the result. Similarly, crime location prediction influences police allocation and may deter crime. Also, economics often has self-fulfilling prophecies, as in a "bank run," where predicting a crisis can cause a crisis.

16.2 Performative Supervised Learning

Definition 16.2 (Risk)**.** The *risk* of a model with parameters θ is

$$\mathrm{Risk}(\theta, \mathcal{D}) = \mathrm{E}_{(x,y)\in\mathcal{D}}[\ell(f_\theta(x), y)].$$

In the performative model, we replace θ with a distribution over models $D(\theta)$ that depends on the specific parameters. The performative risk of θ is

$$\mathrm{Risk}(\theta, D(\theta)).$$

We cannot compute the value of $D(\theta)$, but we can make inferences about it.

Definition 16.3 (Performative optimality)**.** A model θ^* is *performatively optimal* if its performative risk is at least as small as that of any other model θ, i.e.,

$$\mathrm{Risk}(\theta^*, D(\theta^*)) \le \mathrm{Risk}(\theta, D(\theta)).$$

Definition 16.4 (Performative stability)**.** Another natural notion is *performative stability*, where

$$\mathrm{Risk}(\theta^*, D(\theta^*)) \le \mathrm{Risk}(\theta, D(\theta^*)).$$

The intuition for this is that while there may exist more globally optimal models, you have no incentive to change your current model given that you act according to it. It is in some sense, a model that is *optimal for its given distribution.*

An interesting heuristic is that *there are only two kinds of random variables* in the world, up to a first approximation. One is a coin toss that is almost constant with $1-\epsilon$ probability, and one is a coin toss that is almost unbiased with $\frac{1}{2}+\epsilon$. [3]

[2] "When a metric becomes a target, it ceases to be a metric."

[3] Similarly, "every graph is either an expander or a cycle."

We are going to consider a linear model with a squared loss. A simple linear model for $x \in \{-1, 1\}$ has parameter θ, where $f_\theta(x) = 1/2 + \theta x$. The squared loss equals $(y - f_\theta(x))^2$. The *Bayes score* is then the expected value of y conditioned on x, which is $1/2 + \mu x$. We can take $\theta^* = \mu$. In the performative setting, we can take $y = B(1/2 + \mu X + \epsilon \theta X)$. The $\epsilon \theta X$ term makes it so that when you predict something, it becomes more likely.

As it turns out, the stable point is $\mu/(1 - \epsilon)$, while the optimal point is $\mu/(1 - 2\epsilon)$. How can we find performatively stable points or optimal points? The brute force approach is as follows:

1. Create ϵ-net over parameter space.

2. Search over all parameters in ϵ-net.

Another approach is *repeated risk minimization.* First, find a model θ that minimizes loss on data you have. Deploy the model and collect new data from $D(\theta)$. Repeat. Stable points correspond to the fixed points of repeated risk minimization. Assuming strong convexity of the loss and Lipschitzness of $D(\theta)$, repeated (empirical) risk minimization finds a performative stable point, and that performative stable point is close to the performative optimum.

Similar to repeated ERM, one can also train models in the performative setting by taking repeated gradient descent steps. This approach differs from the standard online learning setup because in online learning you compare yourself to the best baseline, but here, the input distribution is constantly changing based on the model.

17 March 31

Today we discuss the Minimally Adequate Teacher (MAT) learning model from [AS94] and applications to various problems that are difficult to solve without membership queries.

17.1 Learning Monotone DNF with a MAT

Recall that a MAT is a learning algorithm to which we make available Membership and Equivalence queries. There are no random examples, and this is an entirely deterministic system. This makes the model easier to analyze for statements like intractibility by reducing to cryptography problems.

We go back to the problem of learning monotone DNF (which is not PAC learnable) with a MAT. First, some terminology about Boolean functions.

Example 17.1 (Prime implicant)**.** For a Boolean function f, t is called a *prime implicant* if $t \implies f$, but no subset of Boolean values $t' \subset t$ satisfies $t' \implies f$.

This can be thought of as a minimal value in the partially ordered set of subsets.

Lemma 17.2. *In a monotone Boolean function f, the number of prime implicants of f is at most the minimum number of terms in any DNF expression equivalent to f.*

Using this lemma, we can describe a MAT algorithm for learning monotone DNF by obtaining prime implicants one-by-one using Membership and Equivalence queries. The algorithm maintains one-sided error (false negatives) and works as follows:

1. Initialize a hypothesis ϕ that always returns false.
2. Repeat the following steps, forever:
   - Run EQUIV(ϕ). If the answer is "Yes," then halt and output ϕ.
   - Otherwise, let $\underline{h} = h_1 h_2 \dots h_n$ be a counterexample, with $\phi(\underline{h}) = 0$, but $f(\underline{h}) = 1$. This is a false negative, so we let
   $$t = \bigwedge_{h_i=1}^{n} x_i.$$
   - Now we have found an additional monomial that we should add to the expression, but this term t still may be too tight, i.e., not be a prime implicant. We can now relax it by moving down the poset.
   - For each index i from 1 to n, check MEMBER($\underline{h}^i$), where $\underline{h}^i$ is $\underline{h}$ with the i-th bit set to zero. If this is a member, then we remove x_i from the conjunction t.
   - We now know that t is a prime implicant, and we add it to our DNF hypothesis by setting $\phi \leftarrow \phi \vee t$.

This runs in polynomial time because the number of prime implicants is linear in size(c), so we have a proof for Proposition 15.4, as desired.

17.2 Learning Regular Languages with a MAT

The following theoretical equivalence provides a motivating example for what follows:

Proposition 17.3 (Equivalence can be approximated by examples)**.** *If there exists a MAT for a concept class $\mathcal{C}$, then $\mathcal{C}$ is also efficiently PAC learnable with examples and membership queries.*

It is known that PAC learning regular languages from examples is enough to break RSA, as we can encode the RSA function inside a finite state automaton (as proven by Kearns and Valiant in [KV94a]). This is intractable in most senses of the word. It turns out, however, that if we are additionally allowed to ask membership queries, it is possible to learn regular languages. We discuss Angluin's algorithm for learning regular languages, published in [Ang87].[4]

Definition 17.4 (Regular language)**.** A *regular language* is a formal language that can be represented by a finite-state automaton, with a starting state, a final state, and characters written on each state transition.

For the rest of this proof, let A be the alphabet of the language, and the notation SA means the set consisting of all members of S concatenated with all members of A. In the proof of MAT learnability, we represent finite-state automata using an *observation table* (S, E, T), where:

- S is a prefix closed set of strings (meaning that any string has all its prefixes included), reflecting the rows of the table, where each element is an example of a sequence of observations.
- E is a suffix closed set of strings (meaning that any string has all its suffixes included), reflecting the columns of the table, where each element is a transition or the end of the string.
- T are labels of a finite function $T : (S \cup SA) \times E \to \{0, 1\}$, which is the table. We usually represent this by having rows named by elements of $S \cup SA$, and columns as elements of E. Each row in the table corresponds to a list of labels, one for each column.

We like to think of each distinct row of the table as a state in a finite-state machine, where there are at most $2^{|E|}$ states. There are two relevant properties of an observation table that make it interesting to study:

- An OT is *closed* if for all $t \in SA$, there exists a string $s \in S$ such that $\text{row}(s) = \text{row}(t)$. This means that if we look-ahead by 1 character, we do not discover a new state.
- An OT is *consistent* if for all $s, t \in S$ such that $\text{row}(s) = \text{row}(t)$, we additionally have that for any character $a \in A$, $\text{row}(sa) = \text{row}(ta)$.

It's easy to see that any finite-state machine corresponds to a closed and consistent observation table. The converse is also true; any closed and consistent observation table can be represented by a finite-state automaton.

Using these background facts from automata theory, we can now describe our MAT algorithm for learning regular languages:

1. Begin with a simple table, with a single row λ (representing the empty string) and a single column λ (representing whether a state should be accepted).
2. Repeat the following steps, forever:

[4] A slightly different variant of this algorithm is presented in Chapter 8 of [KV94b].

- If the table is not closed, expand the set of rows by extending S with SA to make it closed by asking membership queries to the oracle.
- If the table is not consistent, extend it with an additional column so that the contradictory entries are distinct.
- If the table is both closed and consistent, then turn it into a finite-state automaton with one state for every distinct row, which is our tentative hypothesis. Then, run an equivalence query. If equivalent, then we can output the hypothesis. Otherwise, take the returned counterexample and add that to the set S, and continue the algorithm.

That concludes the algorithm. It is not explicitly proven here, but we can also show that the number of steps in this algorithm is equal to the number of states in the minimum finite-state machine that represents the concept.

Next time, in Les's last lecture, we will discuss applications of learning theory to Boolean circuits of various lengths, which will be used to prove intractability results. This will indicate that certain cryptographic problems such as RSA that are conjectured to be hard (discrete logarithms) can be reduced to learning various concept classes.

18 April 2

Today, we will be listening to a lecture given by Ben on Deep Learning & Learning Theory. The hope is that we can use tools from learning theory to understand the efficacy of deep learning. The schedule is as follows.

1. *Lecture 1*: Introduction to Deep Learning, why deep learning generalizes, and how the PAC learning framework can help understand it. More advanced distribution-level: Rademacher Complexity.
2. *Lecture 2*: Uniform Convergence Bounds, SQ-algorithms, hardness of learning parities.
3. *Lecture 3*: Kernel Methods, Geometry: 1-Nearest Neighbors

18.1 Introduction to Deep Learning

The perceptron algorithm is a single neuron. The goal is to learn a linear threshold function. The algorithm involves initializing a weight vector $w = \mathbf{0}$, and slowly moving it closer in the optimal direction. This can only learn linear decision boundaries, which is problematic.

One approach is to use instead $f_w(\varphi(x))$. This is known as a kernel perceptron, where $\varphi : \mathcal{X} \to \mathcal{Z}$ is an appropriate kernel function. We discussed this in Section 9.1.

Another popular approach is to compose neural network layers with each other and train with backpropagation. Then the idea is that while we have weights in earlier layers, we can move the other weights in addition to the first one by calculating derivatives using the chain rule. This is also known as a multi-layer perceptron. However, the perceptron has a threshold function at each layer, so it looks like $\sigma(w^T x + b)$. You can also use various nonlinear functions for the "activation," such as a sigmoid or a ReLU activation.

Then we can fit the weights via techniques like gradient descent (compute loss on entire dataset) or stochastic gradient descent (loss on individual data points or small batches).

Deep learning is very successful, particularly on tasks in computer vision and natural language processing. In practice, these networks are massive, and have far more parameters (i.e. weights) than they have input examples. However, the structure of these neural networks is usually determined empirically based on heuristics and experiments of what may work well (e.g., transformers, convolutional layers).

Another issue is that the VC dimension of neural networks grows like the number of parameters, but you oftentimes have huge neural networks with more parameters than the number of input examples. Empirically, large neural networks find consistent, generalizable hypotheses, which is seen by low error on withheld test data. To analyze the problem of neural networks, we need to formalize a couple of things first:

- What is the concept class?

  Stating the concept class is a big question in the statistical learning theory of deep learning, and there's no single good answer for this, as neural networks can take advantage of various fuzzy classifications or features of the input.

- What is the hypothesis class?

  This is another interesting question. We can hopefully answer this better; for example, we might take the hypothesis class to simply be represented by all vectors of parameters for the neural network. However, this is perhaps too broad because the VC dimension is very high (as we saw before); instead, it may be better to take only those neural networks which could have been *probably obtained through training* on normal inputs.

- What is the distribution of the inputs?

  In the past lectures, we have discussed VC dimension and the PAC model, which assume the worst-case possible distribution of inputs. However, for neural networks that are more complicated to analyze on complicated real-world tasks, this is not good enough, and we should assume some regularity constraints on the distribution of inputs. One tool for solving this is to use the *Rademacher complexity.*

18.2 Rademacher Complexity

Now let's discuss Rademacher complexity, a measure of complexity which is often used in learning theory these days. First, we define the empirical version.

Definition 18.1 (Empirical Rademacher complexity)**.** Given a family of sets $\mathcal{F}$ representing a concept class, the *empirical Rademacher complexity* of $\mathcal{F}$ with respect to a set of m labeled examples $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ is defined by

$$\hat{R}_s(\mathcal{F}) = \mathop{\mathbb{E}}_{\sigma_1,\ldots\sigma_n \sim \{-1,1\}} \left[\sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^{m} \sigma_i f(x_i) \right) \right].$$

This is, roughly speaking, how well the model can fit random labels on S. In essence, how correlated to random labels do we expect the most correlated thing in our model to be?

Now we can define the actual Rademacher complexity, which is somewhat of an "expected value" version of this over some distribution of examples.

Definition 18.2 (Rademacher complexity)**.** Given a family of sets $\mathcal{F}$ and a distribution $\mathcal{D}$ over X, the *Rademacher complexity* of $\mathcal{F}$ with respect to $\mathcal{D}$ is defined by

$$R_m(\mathcal{F}) = \mathop{\mathbb{E}}_{S \sim \mathcal{D}} \hat{R}_S(\mathcal{F}),$$

where the expectation is taken over all sets S of m elements sampled from $\mathcal{D}$.

Note. This definition of Rademacher complexity always gives us a number in the interval $[0, 1]$.

As it turns out, Rademacher complexity gives you good generalization bounds. Specifically, for a given distribution $\mathcal{D}$, and a hypothesis class $\mathcal{H}$, then it must be the case that

$$\mathrm{err}_{\mathcal{D}}(h) \leq \widehat{\mathrm{err}}(h) + R_m(\mathcal{H}) + 3\sqrt{\frac{\log 1/\delta}{m}}$$

with probability $1 - \delta$.

We used union bounds and Chernoff bounds to prove our original generalization bounds. What Chernoff-Hoeffding bounds say is that the sum of independent and identically distributed random variables is tightly concentrated around it's mean.

McDiarmid's inequality shows that a non-sensitive function of independent variable concentrates, where the non-sensitive means that

$$|f(x_i, x_{-i}) - f(x_i', x_{-i})| \leq c_i.$$

It's known that the concept class of neural networks is somewhat simpler if we only consider those that can be obtained through training on a dataset of a given size (for example, CIFAR-10). It is not yet clear exactly how it is simpler, though.

19 April 7

Today we will be covering another application of deep learning in learning theory. Last lecture, we explained the high-level challenges about explaining why deep learning generalizes: $\mathcal{H}, \mathcal{C}, \mathcal{D}$. The VC dimension is big. We tried developing distribution-independent bounds. We introduce the Rademacher complexity, which is a distribution-specific analogue of VC-dimension. We then saw from [ZBH+17] that the Rademacher complexity of DNNs on natural distributions $\mathcal{D}$ is very close to exactly one.

19.1 Margin-Based Bounds

What can we do now? Originally, we took a hypothesis class being all neural networks that could be obtained by training on a set $x \in \mathcal{D}$, $y \sim \{-1, 1\}$. However, if the y's are realistic ("natural" labels), then complexity of neural networks trained on these labels is likely low.

We've actually encountered something like this before! Let's talk about AdaBoost. Weak learners become strong learners. If you run it for a sufficiently large number of iterations, it converges. With too many iterations, the VC dimension of the boosted class gets really large, and there is no guarantee of generalizability – yet the accuracy kept improving. In the 90s, the mystery was why "boosting the margin" works.

One tool for this is *margin-based bounds*, which look at the "confidence" of the model's predictions. For example, consider AdaBoost, which creates a sequence $h_1, h_2, \ldots, h_T$ of classifiers.[5] In the end, we can define a weighted sum

$$f(x) = \sum_{t=1}^{T} a_t h_t(x).$$

If you imagine that the hypotheses have outputs in $\{-1, 1\}$, then this is a linear combination of the weights of the classifiers. The margin of f on (x, y) is equal to the value of $yf(x)$, which is how confident the prediction was. In general, the margin is a continuous function that is positive for correct predictions and negative for incorrect predictions. The intuition is that if f has large margin, it will likely generalize. And in practice, AdaBoost generates larger and larger margins, and this is why generalization error doesn't suffer if we keep running.

Intuitively, if f has large margin, then most perturbations of the weights of the classifier will not change much. In the space of all possible hypotheses, this means that there is a small ball around each large-margin classifier that does not change the classifier much. For each of these, there is an "approximating hypothesis" within the ball.

This is called a "covering number" argument, where you take a bunch of spheres where each of the hypotheses are close to each other. Let's formalize this with the following theorem.

Proposition 19.1 ([SFB+98]). *For all $f(x)$ expressible as a weighted average of classifiers*

$$f(x) = \sum_{t} a_t h_t(x),$$

with $h_t \in \mathcal{H}$, $a_t \geq 0$ and $\sum a_t = 1$, and all margins $\theta > 0$, we have

$$\Pr_{\mathcal{D}}(yf(x) \leq 0) \leq \Pr_{S}(yf(x) \leq \theta) + \tilde{O}\left(\frac{1}{\sqrt{m}}\left(\frac{\log|\mathcal{H}|}{\theta^2} + \log\frac{1}{\delta}\right)^{1/2}\right).$$

In other words, the error on $\mathcal{D}$ is bounded by the high-margin error on S, plus some VC-like term.

[5] This analysis will generalize to any weighted sum of classifiers, not just those obtained by boosting.

Note. In many papers in the literature, the authors instead choose to fix θ to some value (for example, 1) and take a *convex combination* of the individual classifiers, instead of a weighted average. This is equivalent up to multiplication by a constant (similar idea to hinge loss).

The problem with these margin-type arguments is that they aren't good enough for many neural networks. In practice, the norms of the weight matrices in deep neural networks are big, which makes the margin small and the generalization bounds weak. Unfortunately, many of these bounds are not good enough for neural networks, even though they are good enough for boosting.

19.2 Nagarajan & Kolter (2019)

We discuss the paper "Uniform Convergence may be unable to explain generalization in DL" [NK19].

Definition 19.2 (Uniform convergence)**.** Uniform convergence has not been stated formally in class, but basically it states that for sufficiently large m,[6]

$$\Pr_{S\sim\mathcal{D}^m}\left(\sup_{h\in\mathcal{H}}|\mathrm{err}_{\mathcal{D}}(h)-\widehat{\mathrm{err}}_S(h)|>\epsilon_{\text{unif}}\right)\le\delta.$$

In other words, all possible hypothesis within a hypothesis class have a small bound on their generalization error based on the number of samples in S.

Uniform convergence bounds are a broad category that include all of the bounds discussed in class, such as Occam bounds, VC dimension, Rademacher complexity, and even margin-based bounds. However, a recent paper has shown through an experiment that no two-sided uniform convergence bound can explain the generalization gap of deep learning.

The trick is to find the "narrowest possible" hypothesis class $\mathcal{H}$ yet still show that uniform convergence will be vacuous. For a learning algorithm A, define the hypothesis class

$$\mathcal{H}_{\mathcal{S}}=\{h\text{ generated by }A(S)\mid S\subset\mathcal{S}\},$$

where $\Pr_{S\sim\mathcal{D}^n}(S\in\mathcal{S})>1-\delta$. Now let $X=\mathbb{R}^d$, and consider a distribution $\mathcal{D}$ on $\mathbb{R}^d$ defined as:[7]

- With probability $\frac{1}{2}$, take a point uniformly at random from the hypersphere with radius 1.
- With probability $\frac{1}{2}$, take a point uniformly at random from the hypersphere with radius 1.1.

Now, let the ground truth be $f(x)=\mathbf{1}[x\text{ in the smaller sphere}]$.

If we train a neural network to learn this concept, the empirical finding will be that the learned decision boundary is a kind of rough approximation of the interstitial space between the two hyperspheres, sometimes intersecting the spheres.

Because this hypothesis is "bumpy," we can construct a new set S' where every point in S is mirrored between the two spheres, taking the ray through that point and the origin. Then, we can see that there is a big gap between $\mathrm{error}_{S'}(h_S)$ and $\mathrm{error}_{\mathcal{D}}(h_S)$, so any two-sided uniform bound is vacuous. For a blog post, see https://locuslab.github.io/2019-07-09-uniform-convergence/.

[6] Note that this definition is *two-sided.* It shows that with high probability, the generalization error is neither much larger nor much smaller than the error on a sample.

[7] In the paper, the number of dimensions is $d = 1000$.

20 April 9

Today, in Les's lecture, we discuss representation-independent hardness for PAC learning, by reducing various learning problems to widely conjectured cryptographic assertions.

20.1 Cryptographic Hardness of Learning Boolean Circuits

Suppose that we are given X, $\mathcal{C}$, and $\mathcal{D}$, and we have some sets of labeled examples of polynomial size drawn from $\mathcal{D}$. We show that it is impossible to efficiently learn the concept class for certain cases, based on cryptographic conjectures. Examples include:

- The concept class of regular languages.
- The concept class of Boolean formulas of polynomial size.
- The concept class of Boolean circuits of logarithmic depth.

First, let's see the kind of conjectures that we will use.

Example 20.1 (Discrete cube roots)**.** Suppose that we have a semiprime $N = pq$, for distinct primes p and q. We do not know p and q, only N. Then, given some y, it is difficult to find an integer x such that $x^3 \equiv y \pmod{N}$.

Proposition 20.2 (DCRCA)**.** *The* discrete cube root complexity assumption *states that for any polynomial p, there does not exist a $p(n)$-time algorithm that takes n-bit inputs N and $y \in \mathbb{Z}_N^*$, and outputs with probability greater than $\frac{1}{p(n)}$ an x such that $y \equiv x^3 \pmod{N}$, where:*

- *N is the product of two primes p, q chosen at random among the $n/2$-bit primes such that $3 \nmid \phi(N) = (p-1)(q-1)$, and*
- *y is chosen randomly from $\mathbb{Z}_N^*$.*

Here, the probabilities are taken over all choices of p, q, y, and any randomization in the algorithm.

Note. Note the similarity between this problem and RSA encryption with $e = 3$.

Observe that if $3 \nmid (p-1)(q-1)$, then $f_N(x) = x^3 \pmod{N}$ is a permutation of $\mathbb{Z}_N^*$, so this problem is well-defined. Furthermore, if additionally we know p and q, it is easy to find an exponent d such that $x^d \equiv \sqrt[3]{x} \pmod{N}$, so it is essential that we keep p and q hidden. This can be shown with just some basic number theory.

Lemma 20.3. *If $3 \nmid (p-1)(q-1)$, then let d be the modular inverse of $3 \pmod{(p-1)(q-1)}$. Then, for all x relatively prime to N, $x^d \equiv \sqrt[3]{x} \pmod{N}$.*

Proof. Note that the order of $\mathbb{Z}_N^*$ is equal to $(p-1)(q-1)$, so by Lagrange's theorem,[8]

$$x^{(p-1)(q-1)} \equiv 1 \pmod{N}$$

for all $x \in \mathbb{Z}_N^*$. The desired result follows from this, since $3d = 1 + k(p-1)(q-1)$ for some k. □

That's enough about number theory and RSA for now. We can get the following consequence about PAC learnability from this hardness conjecture.

[8] Equivalently, we could use Euler's totient theorem.

Proposition 20.4. *Assuming DCRCA, the concept class of discrete cube roots*

$$\mathcal{C} = \{c_N(y) = \sqrt[3]{y} \mid N = pq;\ 3 \nmid (p-1)(q-1)\}$$

is not PAC learnable, where we assume that $\operatorname{size}(N) = n$.

Proof. Note that DCRCA says that $c_N(y)$ is hard to compute, but we we want to show that it is hard to learn cube roots given *examples* of numbers and their cube roots, which is a statement in the other direction.

We need a slightly subtle argument. Suppose for the sake of contradiction that you could approximately compute $c_N(y)$ from seeing many examples of c_N. Then,

1. Telling the learner the value of N could not make the problem harder.
2. If the learner knows N, then it doesn't need examples, as it could just generate (x^3, x) itself.
3. But then, the learner can solve DCRCA, which we just assumed was hard.

This completes the proof. □

Proposition 20.5. *If DCRCA holds, then the class of polynomial-size Boolean circuits is not PAC learnable.*

Proof. It suffices to construct a polynomial size Boolean circuit, for any N, that computes discrete cube roots modulo N. However, this is easy if we know d from Lemma 20.3. Simply introduce a Boolean circuit that uses a repeated squaring algorithm to compute y^d, which easily has complexity $O(\text{multiplication} \cdot \log d) = O(n^3)$ and circuit depth $O(n)$. □

Note, however, that the difficulty of discrete cube roots lies not in computing the exponent y^d, but rather in finding the correct value of d. By slightly altering our previous learning problem, we can show the stronger statement that logarithmic-depth Boolean circuits are not PAC learnable.

Proposition 20.6. *If DCRCA holds, then the class of logarithmic-depth Boolean circuits is not PAC learnable.*

Proof. Consider a new learning problem in which your input is not the n-bit string y, but rather the n^2-bit string

$$y^* = \{y \bmod N, y^2 \bmod N, y^4 \bmod N, \ldots, y^{2^n} \bmod N\}.$$

Then, the function to be learned is

$$g_{N,i}(y^*) = \begin{cases} 1 & \text{if } y^* \text{ is well-formed and the } i\text{th bit of } y^d \mod N \text{ is } 1, \\ 0 & \text{otherwise.} \end{cases}$$

We claim that if $g_{N,i}$ is PAC learnable, then so is $c_{N,i}$. This is a simple learning reduction, which comes from taking $c_{n,i}$, taking an input y for it, and feeding it into our learning algorithm for $g_{N,i}$.

By the contrapositive of this reduction, this implies that $g_{N,i}$ is not PAC learnable. Furthermore, there is a Boolean circuit of logarithmic depth that calculates $g_{N,i}$ by simply doing multiplication of $O(n)$ integers [BCH86], so the result follows. □

20.2 Applications of Representation-Independent Hardness

We now show that the concept class of regular languages is cryptographically hard to learn, by reducing Boolean circuits to regular languages.

Lemma 20.7 (Memory complexity of simulating circuits)**.** *A Boolean formula of size s can be emulated with $O(\log s)$ memory.*

Proof. Imagine playing a pebble game on the Boolean formula, represented as a binary tree. Each node with a single child can be simulated recursively without any additional memory, by passing a pebble. However, every node with two children (2-ary operations like AND) can be emulated by remembering previous intermediate results in a depth-first traversal, where we always compute the larger child ("heavy" edge) first. Then, the amount of memory used is the maximum number of "light" edges on any path, which is at most $O(\log s)$. □

This is great, and now we can reduce formula evaluation to regular language recognition.

Lemma 20.8. *Evaluation of a Boolean formula of size s can be reduced to recognizing a regular language with complexity $O(s)$.*[9]

Proof. This is a learning reduction. Assume that we have some input $\underline{x} = x_1 \dots x_n$, where $x_i \in \{0, 1\}$, to the Boolean formula. We can then give the input

$$\underline{x}\$\underline{x}\$\dots\$\underline{x},$$

where the number of repetitions of the input will be the number of steps required to evaluate the formula with $k = O(\log s)$ bits of memory, using the algorithm from Lemma 20.7. Then, we can write the state of the learning algorithm as a finite-state automaton with one state for each of the $2^k = s^{O(1)}$ possible memory configurations, as desired. □

[9] Recall that the *complexity* of a regular language is the number of states in the smallest DFA that recognizes it.

21 April 14

Today is the last lecture on new material, which is Part 3 of the deep learning series. We review the issues with uniform convergence as an explanation of deep learning's generalization properties, and we explore interesting new directions in the area.

21.1 Deep Learning Does Not Uniformly Converge

We're going to go over Nagarajan & Kolter (2019) again. First, let's define uniform convergence, which we think of in terms of a uniformly bounded "generalization gap" of all hypotheses on a given training set S. Most of these bounds look like

$$|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| \leq \epsilon$$

with probability $1-\delta$, for all $h \in \mathcal{H}$, where $S \sim \mathcal{D}^m$. The standard approach of ERM is to find a hypothesis for which $\mathrm{err}_S(h)$ was small, so if generalization holds, the error on the entire dataset would also be small.

We can imagine a table of hypotheses and training sets, and then imagine that for all h_i and S_j, we hope that $|\mathrm{err}_{\mathcal{D}}(h_i) - \mathrm{err}_{S_j}(h_i)|$ is very close to ϵ, by the uniform convergence approach.

However, considering the Nagarajan-Kolter concept, we have an instance space $X \subset \mathbb{R}^d$ consisting of an inner sphere (where all points are red) and an outer sphere (where all points are blue). Now given a set of points $S \sim \mathcal{D}^m$, we can define a hypothesis h_S trained from S. We can also define an opposing set $\tilde{S}$ where every red point is mapped to a blue point along the same ray from the origin, and vice versa, which gives us a hypothesis $h_{\tilde{S}}$ trained from the opposing set. The issue is that experimentally, if we train a deep neural network h_S on S, the network will have a "bumpy" margin around each point that often misclassifies the corresponding point on the opposing sphere, so we end up with

$$\mathrm{err}_S(h_{\tilde{S}}) \approx 1,$$
$$\mathrm{err}_{\mathcal{D}}(h_{\tilde{S}}) \approx 0.$$

This means that we can construct an h for every S where $|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)|$ is large in absolute value, which contradicts two-sided uniform convergence.

Note. In this counterexample, we actually have that the trained hypothesis $h_{\tilde{S}}$ is *much better* on the entire distribution than the specific subset S, which means that the generalization error is actually negative. However, even though this is a good thing in principle, it still unfortunately shows that most previous statistical techniques fail, as they always show two-sided bounds.

21.2 Local Interpolating Schemes

We can also use a heuristic argument by approximating with a smaller representation class. In principle, if deep learning learns a function that might be *close* to some $h \in \mathcal{H}$, then if we can explain that $\mathcal{H}$ works with uniform convergence, and the error of the h is small, then we can still show that deep learning generalizes. Now, let's use geometry of concepts. Say that we draw a sample set with $f(x) = \mathrm{Bernoulli}(\eta(x))$. Then, given this sample, we can construct a Voronoi diagram corresponding to the 1-nearest neighbor classifier trained on this set.

Using this model, we can prove certain facts about the smoothness of the geometry of the concept class, which can show generalization through *local interpolating schemes* such as k-nearest neighbors. For more details on this, see [BHM18].

21.3 Connections Between Deep Learning and Kernel Regression

This topic is currently very popular; a few related buzzwords include:

- "random features,"
- "lazy training regime," and
- "neural tangent kernel."

To introduce this, let's consider a one-hidden layer neural network. Basically, we can write this down as a classifier function

$$N(\mathbf{x}) = \sum_{i=1}^{r} u_i \sigma(\langle \mathbf{w}_i, \mathbf{x} \rangle),$$

where our parameters are u and $\mathbf{w}_i$. Here we consider the case when $\sigma(x)$ is a ReLU activation function, i.e.,

$$\sigma(z) = \begin{cases} z & \text{if } z \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

The crucial observation is that in the early part of the training, very few of the hidden-layer preactivations $\langle \mathbf{w}_i, \mathbf{x} \rangle$ flip from positive to negative or negative to positive. Then we can show that the above expression equals

$$\sum_{i=1}^{r} u_i^{(t)} \langle w_i^{(t)}, x \rangle \mathbf{1}[\langle w_i^{(t)}, x \rangle \geq 0].$$

Because very few of the preactivations change sign, this can be approximated by

$$\sum_{i=1}^{r} u_i^{(t)} \langle w_i^{(t)}, x \rangle \mathbf{1}[\langle w_i^{(0)}, x \rangle \geq 0] = \sum_{i=1}^{r} \langle u_i^{(t)} w_i^{(t)}, x \mathbf{1}[\langle w_i^{0}, x \rangle \geq 0] \rangle.$$

We can even remove the weights $w_i^{(t)}$ by combining them with the learned linear factors $u_i^{(t)}$, so our concept just looks like

$$\sum_{i} \langle u_i^{(t)}, \phi_i(x) \rangle,$$

which is simply a linear network over the the kernel functions $\phi_i(x)$. Since the kernel functions are only determined by the initialization of our weights, this is essentially the expression of a linear classifier over "random features" [RR08] (NIPS 2017 test of time award winner), which can be captured by a kernel function

$$\langle \phi(x), \phi(x') \rangle = K(x, x').$$

This has a lot of advantages when capturing the essence of learning with wide neural networks, and it is also connected to the well-understood land of kernel methods.

However, in practice, neural tangent kernel methods have issues because neural networks aren't wide enough for this model to be appropriate. Recent results have even shown that random features cannot learn the concept which is a single ReLU neuron [YS19]. Similar results show that gradient-based methods provably can't learn parity-like classes.

References

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[AS94] Dana Angluin and Donna K. Slonim. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14:7–26, 1994.

[BCH86] Paul W Beame, Stephen A Cook, and H James Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15(4):994–1003, 1986.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[BHM18] Mikhail Belkin, Daniel J Hsu, and Partha Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In *Advances in neural information processing systems*, pages 2300–2311, 2018.

[BR92] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[KV94a] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[KV94b] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.

[Mad16] Mark E. Madsen. Intentionality and cultural evolution—towards a generalized learning theory account. http://notebook.madsenlab.org/essays/2016/03/03/cultural-evolution-intentionality-pac-learning.html, 2016.

[NK19] Vaishnavh Nagarajan and J Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 11611–11622, 2019.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[RR08] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[SFB$^+$98] Robert E Schapire, Yoav Freund, Peter Bartlett, Wee Sun Lee, et al. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[YS19] Gilad Yehudai and Ohad Shamir. On the power and limitations of random features for understanding neural networks. In *Advances in Neural Information Processing Systems*, pages 6594–6604, 2019.

[ZBH$^+$17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.