6.080 / 6.089 Great Ideas in Theoretical Computer Science
Spring 2008

6.080/6.089 GITCS April 24, 2008

Lecture 19

Lecturer: Scott Aaronson *Scribe: Michael Fitzgerald*

1 Recap And Discussion Of Previous Lecture

In the previous lecture, we discussed different cryptographic protocols. People asked: "In the RSA cryptosystem, why do people raise to a power greater than three?" Raising to a power greater than three is an extra precaution; it's like adding a second lock on your door. If everything has been implemented correctly, the RSA we discussed (cubing the message $\pmod n$) should be fine. This assumes that your message has been padded appropriately, however. If your message hasn't been padded, "small-exponent attacks" can be successful at breaking RSA; sending the message to a bunch of different recipients with different public keys can let the attacker take advantage of the small exponent. Raising to a power greater than three mitigates this risk.

There are a couple of other attacks that can be successful. "Timing attacks" look at the length of time the computer takes to generate numbers to get hints as to what those numbers are. Other attacks can look at the electromagnetic waves coming from the computer to try and get hints about the number. Then there are attacks that abuse the cryptosystem with constructed inputs and try to determine some information about the system based on the error messages they receive. In general, modern cryptosystems are most often defeated when attackers find bugs in the *implementations* of the systems, not in the systems themselves. Social engineering remains the most successful way of breaching security; often just calling someone on the phone, pretending to be a company tech support person, and asking for their password will get you a response.

We also talked about zero-knowledge proofs and general interactive protocols in the last lecture. Twenty years ago, a revolution in the notion of "proof" drove home the point that a proof doesn't have to be just a static set of symbols that someone checks for accuracy. For example, a proof can be an interactive process that ends with you being convinced of a statement's truth, without learning much of anything else. We gave two examples of so-called *zero-knowledge protocols*: one that convinces a verifier that two graphs are not isomorphic, and another that proves *any* statement with a short conventional proof, assuming the existence of one-way functions.

2 More Interactive Proofs

It turns out that this notion of an interactive proof is good for more than just cryptography. It was discovered in the early 1990s that interactive proofs can convince you of solutions to problems that we think are much harder than NP-complete ones. As an analogy, it's hard to tell that an author of a research paper knows what he's talking about just from reading his paper. If you get a chance to ask him questions off the cuff and he's able to respond correctly, it's much more convincing. Similarly, if you can send messages back and forth with a prover, can you use that to convince yourself of more than just the solution to an NP-complete problem? To study this in the 1980s, people defined a complexity class called IP, which stands for "interactive proof." The details of this story are beyond the scope of the class, but it's being mentioned because it's important.

Consider the following scenario. Merlin and Arthur are communicating. Merlin has infinite computational power, but he is not trustworthy. Arthur is a PPT (probabilistic polynomial time)

king; he can flip coins, generate random numbers, send messages back and forth with Merlin, etc. What we want from a good protocol is this: if Merlin is telling the truth, then there should be some strategy for Merlin that causes Arthur to accept with probability 1. On the other hand, if Merlin is lying, then Arthur should reject with probability greater than 1/2, *regardless* of Merlin's strategy. These correspond to the properties of completeness and soundness that we discussed a while ago.

How big is the class IP, then? It certainly contains NP, as Merlin's strategy could just be to send a solution over to Arthur for the latter to check and approve. Is IP bigger than NP, though? Does interaction let you verify more statements than just a normal proof would? In 1990, Lund, Fortnow, Karloff, and Nisan showed that IP contains $coNP$ as well. This isn't obvious; the key idea in the proof involves how polynomials over finite fields can be judged as equal by testing them at random points. This theorem takes advantage of that fact, along with the fact that you can reinterpret a Boolean formula as a polynomial over a finite field. An even bigger bombshell came a month later, when Shamir showed that IP contains the entire class $PSPACE$, of problems solvable with polynomial memory. Since it was known that IP is *contained* in $PSPACE$, this yields the famous result $IP = PSPACE$.

What does this result mean, in intuitive terms? Suppose an alien comes to earth and says, "I can play perfect chess." You play the alien and it wins. But this isn't too surprising, since you're not very good at chess (for the purposes of this example, at least). The alien then plays again your local champion, then Kasparov, then Deep Blue, etc., and it beats them all. But just because the alien can beat anyone on earth, doesn't mean that it can beat anything in the universe! Is there any way for the alien to prove the stronger claim?

Well, remember that earlier we mentioned that a generalized $n \times n$ version of chess is a $PSPACE$ problem. Because of that, we can transform chess to a game about polynomials over finite fields. In this transformed game, the best strategy for one of the players is going to be to move randomly. Thus, if you play randomly against the alien in this transformed game and it wins, you can be certain (with only an exponentially small probability of error) that it has an optimal strategy, and could beat anyone.

You should be aware of this result, as well as the zero-knowledge protocol for the 3-Coloring, since they're two of the only examples we have in computational complexity theory where you take an NP-complete or $PSPACE$-complete problem, and do something with it that actually exploits its *structure* (as opposed to just treating it as a generic search problem). And it's known that exploiting structure in this sort of way—no doubt, at an astronomically more advanced level—will someday be needed to solve the $P = NP$ problem.

3 Machine Learning

Up to this point, we've only talked about problems where all the information is explicitly given to you, and you just have to do something with it. It's like being handed a grammar textbook and asked if a sentence is grammatically correct. Give that textbook to a baby, however, and it will just drool on it; humans learn to speak and walk and other incredibly hard things (harder than anything taught at MIT) without ever being explicitly told how to do them. This is obviously something we'll need to grapple with if we ever want to understand the human brain. We talked before about how computer science grew out of this dream people had of eventually understanding the process of thought: can you reduce it to something mechanical, or automate it? At some point, then, we'll have to confront the problem of learning: inferring a general rule from specific examples when the rule is never explicitly given to you.

3.1 Philosophy Of Learning

As soon as we try to think about learning, we run into some profound philosophical problems. The most famous of these is the *Problem of Induction*, proposed by 18$^{\text{th}}$-century Scottish philosopher David Hume. Consider two hypotheses:

1. The sun rises every morning.
2. The sun rises every morning until tomorrow, when it will turn into the Death Star and crash into Jupiter.

Hume makes the point that both of these hypotheses are completely compatible with all the data we have up until this point. They both explain the data we have equally well. We clearly believe the first over the second, but what grounds do we have for favoring one over the other? Some people say they believe the sun will rise because they believe in the laws of physics, but then the question becomes why they believe the laws of physics will continue.

To give another example, here's a "proof" of why it's not possible to learn a language, due to Quine. Suppose you're an anthropologist visiting a native tribe and trying to learn their language. One of the tribesmen points to a rabbit and says "gavagai." Can you infer that "gavagai" is their word for rabbit? Maybe gavagai is their word for food or dinner, or "little brown thing." By talking to them longer you could rule those out, but there are other possibilities that you haven't ruled out, and there will always be more. Maybe it means "rabbit" on weekdays but "deer" on weekends, etc.

Is there any way out of this? Right, we can go by Occam's Razor: if there are different hypotheses that explain the data equally well, we choose the simplest one.

Here's a slightly different way of saying it. What the above thought experiments really show is not the impossibility of learning, but rather the impossibility of learning in a theoretical vacuum. Whenever we try to learn something, we have some set of hypotheses in mind which is vastly smaller than the set of all logically conceivable hypotheses. That "gavagai" would mean "rabbit" is a plausible hypothesis; the weekday/weekend hypothesis does *not* seem plausible, so we can ignore it until such time as the evidence forces us to.

How, then, do we separate plausible hypotheses from hypotheses that aren't plausible? Occam's Razor seems related to this question. In particular, what we want are hypotheses that are *simpler than the data they explain*, ones that take fewer bits to write down than just the raw data. If your hypothesis is extremely complicated, and if you have to revise your hypothesis for every new data point that comes along, then you're probably doing something wrong.

Of course, it would be nice to have a theory that makes all of this precise and quantitative.

3.2 From Philosophy To Computer Science

It's a point that's not entirely obvious, but the problem of learning and prediction is related to the problem of data compression. Part of predicting the future is coming up with a succinct description of what has happened in the past. A philosophical person will ask why that should be so, but there might not be an answer. The belief that there are simple laws governing the universe has been a pretty successful assumption, so far at least. As an example, if you've been banging on a door for five minutes and it hasn't opened, a sane person isn't going to expect it to open on the next knock. This could almost be considered the definition of sanity.

If we want to build a machine that can make reasonable decisions and learn and all that good stuff, what we're really looking for is a machine that can create simple, succinct descriptions and

hypotheses to explain the data it has. What exactly is a "simple" description, then? One good way to define this is by Kolmogorov complexity; a simple description is one that corresponds to a Turing machine with few states. This is an approach that many people take. The fundamental problem with this is that Kolmogorov complexity is not computable, so we can't really use this in practice. What we want is a quantitative theory that will let us deal with *any* definition of "simple" we might come up with. The question will then be: "given some class of hypotheses, if we want to be able to predict 90% of future data, how much data will we need to have seen?" This is where theoretical computer science really comes in, and in particular the field of *computational learning theory*. Within this field, we're going to talk about a model of learning due to Valiant from 1984: the PAC (Probably Approximately Correct) model.

3.3 PAC Learning

To understand what this model is all about, it's probably easiest just to give an example. Say there's a hidden line on the chalk board. Given a point on the board, we need to classify whether it's above or below the line. To help, we'll get some sample data, which consists of random points on the board and whether each point is above or below the line. After seeing, say, twenty points, you won't know *exactly* where the line is, but you'll probably know roughly where it is. And using that knowledge, you'll be able to predict whether most future points lie above or below the line.

Suppose we've agreed that predicting the right answer "most of the time" is okay. Is any random choice of twenty points going to give you that ability? No, because you could get really unlucky with the sample data, and it could tell you almost nothing about where the line is. Hence the "Probably" in PAC.

As another example, you can speak a language for your whole life, and there will still be edge cases of grammar that you're not familiar with, or sentences you construct incorrectly. That's the "Approximately" in PAC. To continue with that example, if as a baby you're really unlucky and you only ever hear one sentence, you're not going to learn much grammar at all (that's the "Probably" again).

Let's suppose that instead of a hidden line, there's a hidden squiggle, with a side 1 and a side 2. It's really hard to predict where the squiggle goes, just from existing data. If your class of hypotheses is arbitrary squiggles, it seems impossible to find a hypothesis that's even probably approximately correct. But what is the difference between lines and squiggles, that makes one of them learnable and the other one not learnable?

Well, no matter how many points there are, you can always cook up a squiggle that works for those points, whereas the same is not true for lines. That seems related to the question somehow, but why?

What computational learning theory lets you do is delineate mathematically what it is about a class of hypotheses that makes it learnable or not learnable (we'll get to that later).

3.4 Framework

Here's the basic framework of Valiant's PAC Learning theory, in the context of our line-on-the-chalkboard example:

S: Sample Space - The set of all the points on the blackboard.

D: Sample Distribution - The probability distribution from which the points are drawn (the uniform distribution in our case).

Concept - A function $h : S \to \{0, 1\}$ that maps each point to either 0 or 1. In our example, each concept corresponds to a line.

C: Concept Class - The set of all the possible lines.

"True Concept" $c \in C$: The actual hidden line; the thing you're trying to learn.

In this model, you're given a bunch of sample points drawn from S according to D, and each point comes with its classification. Your goal is to find a hypothesis $h \in C$ that classifies future points correctly almost all of the time:

$$\Pr_{x \in D}[h(x) = c(x)] \geq 1 - \epsilon$$

Note that the future points that you test on should be drawn from the same probability distribution D as the sample points. This is the mathematical encoding of the "future should follow from the past" declaration in the philosophy; it also encodes the well-known maxim that "nothing should be on the test that wasn't covered in class."

As discussed earlier, we won't be able to achieve our goal with certainty, which is why it's called *Probably* Approximate Correct learning. Instead, we only ask to succeed in finding a good classifier with probability at least $1 - \delta$ over the choice of sample points.

One other question: does the hypothesis h have to belong to the concept class C? There are actually two notions, both of which we'll discuss: *proper learning* (h must belong to C) and *improper learning* (h can be arbitrary).

These are the basic definitions for this theory.

Question from the floor: Don't some people design learning algorithms that output confidence probabilities along with their classifications?

Sure! You can also consider learning algorithms that try to predict the output of a real-valued function, etc. Binary classification is just the simplest learning scenario – and for that reason, it's a nice scenario to focus on to build our intuition.

3.5 Sample Complexity

One of the key issues in computational learning theory is *sample complexity*. Given a concept class C and a learning goal (the accuracy and confidence parameters ϵ and δ), how much sample data will you need to achieve the goal? Hopefully the number of samples m will be a finite number, but even more hopefully, it'll a *small* finite number, too.

Valiant proposed the following theorem, for use with finite concept classes, which gives an upper bound on how many samples will suffice:

$$m \geq \frac{1}{\epsilon} \log \frac{|C|}{\delta}$$

As ϵ gets smaller (i.e., as we want a more accurate hypothesis), we need to see more and more data. As there are more concepts in our concept class, we also need to see more data.

A learning method that achieves Valiant's bound is simply the following: find any hypothesis that fits all the sample data, and output it!

As long as you've seen m data points, the theorem says that with probability at least $1-\delta$, you'll have a classifier that predicts at least a $1 - \epsilon$ fraction of future data. There's only a logarithmic dependency on $\frac{1}{\delta}$, which means we can learn within an exponentially small probability of error using only a polynomial number of samples. There's also a log dependence on the number of concepts $|C|$, which means that even if there's an exponential number of concepts in our concept class, we

can still do the learning with a polynomial amount of data. If that weren't true we'd really be in trouble.

Next time: proof of Valiant's bound, VC-dimension, and more...