Lecture 8

The million dollar question in computational complexity is "do all problems in NP have efficient algorithms?" The conventional belief is that it is very unlikely, but a proof is still lacking after more than 50 years of research. In the first few lectures we gave examples of computational hardness in some restricted models of computation: decision trees, small depth circuits, and restricted branching programs. If we could generalize these methods to prove that, for example, solving satsifiability of 3CNF formulas on n inputs requires circuits of size $2^{n^{1/100}}$, we would have an example of an NP problem that does not have small circuit families and so no efficient algorithms either.

There is an interesting explanation about why this seemingly promising approach has failed to yield consequential results. The reason is that all known proofs of computational hardness are *learning algorithms* in disguise. The learning algorithm asks to see outputs of the circuit for inputs of its choice and can then predict the value of the circuit at a previously unseen input.

On the other hand, it is widely believed that there exist efficiently computable functions that are not efficiently learnable. Such functions are called *pseudorandom functions* and can be constructed from sufficiently strong pseudorandom generators. So if there was a proof that 3SAT on n inputs requires circuits of size $2^{n^{1/100}}$, either this proof would look very different from, say, the proof that parity on n bits requires depth 3 circuits of size $2^{n^{1/3}}$, or else pseudorandom functions do not exist, everything is learnable efficiently, and cryptography is impossible.

1 Largeness and constructivity

The circuit lower bounds we proved in the first few lectures all followed the same pattern: We came up with a separating property P that is true for the hard function but false for all functions computed by small circuits in the class. For example, in the proof that $PARITY$ requires decision tree depth n, one separating property $P(f)$ is

`restzero`(f): f does not restrict to a constant after fixing its first $n-1$ inputs to zero.

PARITY satisfies `restzero`. Depth-$(n-1)$ decision trees to not. Hence $PARITY$ cannot be computed by any depth-$(n-1)$-decision tree.

Razborov and Rudich noticed that all properties P underlying known proofs of circuit lower bounds (for complete models) have two features (properties of properties): Smallness and constructivity.

Largeness

This property "f does not restrict to a constant after fixing its first $n-1$ inputs to zero" holds not only for PARITY but also for large fraction of all possible functions: the probability that this is true for a random function R is precisely $1/2$. This is an example of a $1/2$-large property.

Definition 1. A property P of boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ is $(1-\delta)$-*large* if the probability that a random function fails to satisfy P is at most δ.

In words, the proof that PARITY requires decision tree depth n that is based on the separating property P also proves that at least half of all possible functions require decision tree depth n. This is the largeness feature: A lower bound proof for a specific function like PARITY also applies to a random function.

Let's look at some other proofs. The proof that $PARITY$ on n bits requires depth d AND/OR circuits of size $s = 2^{\Omega(n^{1/(d-1)})}$ in Lecture 2 was based on the following property:

`restrand`(f): At most a $1/3$-fraction of all $(n-1)$-restrictions fix f to a constant.

The constant 1/3 is somewhat arbitrary but will serve to illustrate the point. `restrand` certainly holds for PARITY. We used the switching lemma to show that it doesn't hold for any size-s, depth-d AND/OR circuit on n inputs. The probability that a random function has this property is overwhelming: For every index i, there are 2^{n-1} $(n-1)$-restrictions that leave the i-th variable unrestricted. The events "this restriction fixes f" are independent of probability 1/2. By a Chernoff bound the probability that more than 1/3 of those restrictions fix f to a constant is $2^{-\Omega(2^{n-1})}$. By a union bound over the n values of i, the probability that `restrand` fails to hold for a random function is at most $n \cdot 2^{-\Omega(2^{n-1})}$, so `restrand` is $(1-2^{-\Omega(2^n)})$-large.

The proof that MAJORITY on n inputs requires depth-d circuits of size $2^{\Omega(n^{1/2d})}$ relied on the property

> `polyapx`(f): There does not exist an $\mathbb{F}_2$-polynomial of degree $n/2-\sqrt{n}$ that agrees with f on all but a 1/16-fraction of inputs.

In Lecture 2 we proved that `polyapx` holds for MAJORITY, but it does not hold for size $2^{o(s^{1/2d})}$-size depth-d AND/OR/PARITY circuits. This property also happens to be large. The number of functions having this property is at most $2^{\binom{n}{\leq n/2-\sqrt{n}}}$ (the number of possible polynomials) times $\binom{2^n}{\leq 2^n/16}$ (the number of disagreement patterns). It can be shown that the product of this numbers is at most $2^{0.99 \cdot 2^n}$, so this property is $(1-2^{-0.01 \cdot 2^n})$-large.

The separating property between the IP function on $2n$ bits and read-k-times ranodmized branching programs of width $w = \Omega(2^{n/2k})$ was

> `balancedrect`(f): In all rectangles $X \times Y$ of size $|X| \cdot |Y| \geq 10 \cdot 2^n$ of the matrix f, at least 1/3 of the values are zeroes and at least 1/3 of the values are ones.

It is again true that almost all functions have `balancedrect`: it is $1-2^{-\Omega(2^n)}$-large. (Proof sketch: Any fixed rectangle with $|X| = s, |Y| = t$ is balanced except with probability $2^{\Omega(st)}$ by a Chernoff bound. By a union bound the probability that there exists a balanced rectrangle of these dimensions is at most $\binom{2^n}{s}\binom{2^n}{t}2^{-\Omega(st)} \leq 2^{n(s+t)-\Omega(st)}$. If $st \geq 10 \cdot 2^n$ this is at most $2^{-\Omega(2^n)}$. Taking a union bound over all possible pairs of sizes s, t of which there are at most 2^{2n} gives the desired bound.)

To summarize, all of the lower bound proofs we saw were obtained using a common scheme. We came up with a separating property between the hard function and the circuit model. It happened that all the properties are large so the proof also shows that the circuit model cannot compute most functions. The conclusion is not surprising owing to Shannon's counting argument. What is a bit surprising is that the proofs give an alternative way to reach the same conclusion without counting circuits. This feature of the proofs, when coupled with the next one, turns out to be a weakness in the quest of size lower bounds for general circuits.

Constructivity

Constructivity postulates that the relevant lower bound property is efficiently computable: A fairly small circuit that can *query* the function $f\colon \{0,1\}^n \to \{0,1\}$ at inputs of its choice can determine if the property holds or not.

Such circuits are called *oracle circuits*. An oracle circuits accesses its input function f via *oracle gates* that represent evaluations of f (so they have n input wires that, when fed with x, produce the singe output $f(x)$). Oracle gates count towards the size of the circuit. The execution of an oracle circuit C^f (a circuit C that is given oracle access to the function f) can be modeled as a decision tree whose input is the 2^n-bit long string of values $f(x)$ as x ranges over all 2^n inputs. The depth of the decision tree is then the maximum number of oracle gates on any input-output path in the circuit.

Definition 2. A property P of Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ is *S-constructive* if P can be tested by oracle circuits of size at most S.

It turns out (for not completely understood reasons) that properties behind proofs of Boolean circuit lower boudns are $2^{O(n)}$-constructive. The function f is described by a 2^n-bit long string, so $2^{O(n)}$ time is polynomial in the description size of the input function f.

Let's consider `restzero` first. This property can be tested by a circuit of constant size: The circuit evaluates f on inputs $0 \cdots 00$ and $0 \cdots 01$ and accepts if the outputs are diffeerent.

The property `restrand` can be tested by circuits of size $O(n2^n)$. This circuit accepts if $f(x) \neq f(x')$ for at least a third of all $n2^{n-1}$ input-pairs x, x' that differ in exactly one position. This count can be implemented in size $O(n^2 2^n)$.

The `balancedrect` property is more interesting. It is unclear if this property can be computed by size $2^{O(n)}$ circuits. A brute-force algorithm would have to check all pairs of sets (X, Y) with $|X| = |Y| = \Theta 2^{n/2}$. There are about $\binom{2^n}{\Theta(2^{n/2})}^2 \geq 2^{\Omega(2^{n/2})}$ such pairs of sets. The complexity of this algorithm is exponential in $2^{n/2}$, a far cry from $2^{O(n)}$.

Now recall how we proved that `balancedrect` holds for IP. We showed that `balancedrect`(IP) is implied by the fact that the spectral norm of the IP matrix (in ± 1 representation) is at most $2^{n/2}$. We can formalize this property by the following predicate:

`spectral`(f): the ± 1 matrix representing f has spectral norm at most $K = 2^{n/2}$.

Unlike `balancedrect`, `spectral` can be computed by a circuit of size $2^{O(n)}$. This circuit can be derived from algorithms for calculating (in fact approximating with excellent precision) the spectral norm of a matrix which are efficient: they run in time polynomial in the matrix dimensions.

Since `spectral` implies `balancedrect`, it won't be as large, so we need to go back and verify that it holds for most functions. If we introduce some slack in the spectral norm, say by setting K to $10 \cdot 2^{n/2}$, it turns out that this is true: A random ± 1 matrix with dimensions $2^n \times 2^n$ has spectral norm at most $10 \cdot 2^{n/2}$ except with probability $2^{-\Omega(n)}$.

For similar reasons, it is unclear if the `polyapx` property is constructive. In that example the stronger separating property "f has rational degree at least $n/2 - \sqrt{n}$" happens to be both large and constructive.

In summary, all circuit lower bounds that we saw in Lectures 1, 2, and 3 are based on some separating property that is both large and constructive. Identifying the relevant property among several candidates may require reworking (in the words of Razborov and Rudich, naturalizing) the proof.

2 Pseudorandom Functions

To explain what a pseudorandom function is, we should say a bit more about (truly) random functions. A function from $\{0,1\}^n$ to $\{0,1\}$ can be uniquely specified by listing all 2^n values $F(\mathtt{00} \cdots \mathtt{0}), \dots, F(\mathtt{11} \cdots \mathtt{1})$. As there are two choices for each value $F(x)$ there are 2^{2^n} possible functions. A random function R is a random variable whose distribution is uniform over this set of 2^{2^n} possible functions.

A random function R can be chosen by sampling the 2^n values $R(\mathtt{00} \cdots \mathtt{0}), \dots, R(\mathtt{11} \cdots \mathtt{1})$ as uniform and independent random bits. This gives a natural implementation of R by a randomized program: The program first samples and stores 2^n random bits. Then on input x it outputs the x-th value in its memory. In the circuit model, an implementation of a random function $\{0,1\}^2 \to \{0,1\}$ might look like this: On input $x_1 x_2$ and randomness $r_1 r_2 r_3 r_4$, output

$$(x_1 \text{ AND } x_2 \text{ AND } r_1) \text{ OR } (x_1 \text{ AND } \overline{x_2} \text{ AND } r_2) \text{ OR } (\overline{x_1} \text{ AND } x_2 \text{ AND } r_3) \text{ OR } (\overline{x_1} \text{ AND } \overline{x_2} \text{ AND } r_4).$$

The size of this implementation scales exponentially in n. Intuitively this should be unavoidable because any implementation of R must store 2^n independent random values $r_1, \dots r_{2^n}$. In the circuit model this can be made precise: Any randomized circuit that implements a random function must have size exponential in n.[1] In fact, any circuit of substantially smaller size must compute a function that is statistically very far from a truly random function.

[1] We are talking about *stateless* implementations: Once the randomness is fixed the function is determined.

A pseudorandom function is a randomized function P that is "indistinguishable from random" but is computable by small randomized circuit. What does it mean for a function to be indistinguishable from random? Since a function is completely described by listing its values, we would like to say that the lists

$$(P(\mathtt{00\cdots0}), P(\mathtt{00\cdots1}), \dots, P(\mathtt{11\cdots1})) \quad \text{and} \quad (R(\mathtt{00\cdots0}), R(\mathtt{00\cdots1}), \dots, R(\mathtt{11\cdots1}))$$

are computationally indistinguishable. This and more will be true under the following definition of indistinguishability by oracle circuits.

Definition 3. A randomized function $P\colon \{0,1\}^n \to \{0,1\}^m$ is an (S, ε)-*pseudorandom function* if for every oracle circuit D of size at most S, $\Pr[D^P = 1] - \Pr[D^R = 1]$ is at most ε, where R is a random function from $\{0,1\}^n \to \{0,1\}^m$.

We will use this definition with $m = 1$, but the generality will be helpful for constructing pseudorandom functions.

The aim is to implement a pseudorandom function P by an efficient (randomized) circuit C. Recall that C takes two types of inputs: The input $x \in \{0,1\}^n$ and its internal randomness $K \in \{0,1\}^k$, which is called the *key* of the pseudorandom function. Different values of the key K give rise to different functions $P(x) = P_K(x) = C(K, x)$. The key $K \sim \{0,1\}^k$ is chosen at random but it is kept secret from the distinguisher: The distinguisher can ask to see values $P_K(x)$ on inputs x of its choice but it has no information about the choice of K beyond what it can deduce from these values.

3 The Razborov-Rudich Barrier

Pseudorandom functions P that are computable by circuits of size polynomial in n are believed to exist for every n. We will next show how such functions can be constructed from sufficiently strong pseudorandom generators.

On the other hand, if polynomial-size circuits satisfy some property P that is both large and constructive, then they cannot compute pseudorandom functions:

Theorem 4. *If no function $f\colon \{0,1\}^n \to \{0,1\}$ in some class $\mathcal{C}$ satisfies a property that is $(1-\delta)$-large and S-constructive with probability at most δ then $\mathcal{C}$ cannot compute $(S, 1-\delta)$-pseudorandom functions.*

Proof. Let D be a circuit computing the property. By largeness $\Pr[D^R = 1] \geq 1 - \delta$. Since no function in $\mathcal{C}$ has the property, $\Pr[D^P = 1] = 0$ for all distributions P over functions in $\mathcal{C}$. So

$$\Pr_F[D^P = 1] - \Pr[D^R = 1] \geq 1 - \delta.$$

Since the property is computable by size-S circuits S, $\mathcal{C}$ cannot compute $(S, 1-\delta)$-pseudorandom functions. $\square$

In the next section we show how to construct a $(\sqrt{s}, \sqrt{s}\varepsilon)$ pseudorandom function $P\colon \{0,1\}^n \to \{0,1\}$ from a (s, ε)-pseudorandom generator $G\colon \{0,1\}^k \to \{0,1\}^{2k}$. If, for example, $s = 2^{k/10}$ and $\varepsilon = 2^{-k/10}$ we get a $(2^{k/20}, 2^{-k/20})$-pseudorandom function P from a $(2^{k/10}, 2^{-k/10})$-pseudorandom generator G. If G is computable by circuits of size polynomial in k, P will be computable in size polynomial in n as long as n is polynomial in k. Specifically, by setting say $k = 400n$ we obtain a $(2^{20n}, 2^{-20n})$-pseudorandom function P computable by the class $\mathcal{C}$ of size-n^C circuits (for a suitable constant C).

By Theorem 4, size-n^C circuits cannot satisfy any property that is both 2^{20n}-constructive and 2^{-20n}-large, including any of `restzero`, `restrand`, `spectral`, `ratdegree`. In words, the types of properties that were useful for obtaining the circuit lower bounds in Lectures 1-3 are ineffective against polynomial-size circuits (assuming sufficiently strong pseudorandom generators exist).

4 How to Construct Pseudorandom Functions

A pseudorandom function P_K can in particular be viewed as a pseudorandom generator: The seed is the key $K \in \{0,1\}^k$ and the output can be the list of any say $2k$ distinct evaluations $(P_K(x_1), \dots, P_K(x_{2k}))$. By Definition 3 this is indistinguishable from a uniform random string of length $2k$.

It is remarkable that the following converse is also true: Any pseudorandom generator can be used to implement a pseudorandom function with arbitrarily long inputs. We will show this by induction on the input length n of the pseudorandom function.

We assume that the generator $G\colon \{0,1\}^k \to \{0,1\}^\ell$ is length-doubling, i.e. $\ell = 2k$. This can be achieved by the stretch-extension procedure from last lecture. Such a generator can be viewed as a pseudorandom function P_1 with one bit of input and k bits of output: The values $P(0)$ and $P(1)$ are the k left bits $G_0(K)$ and the k right bits $G_1(K)$ of G respectively, namely

$$P(b) = G_b(K), \quad b \in \{0,1\},$$

providing a base case for the induction.

Now suppose we have constructed a pseudorandom function P with $n-1$ bits of input (and k bits of output). Extending the input of P by one bit amounts to "stretching" the 2^{n-1} values $P(x)$ to 2^n pseudorandom values $P'(y)$, where P' has n bits of input. This can be done with another another application of G:

$$P'(bx) = G_b(P(x)), \quad x \in \{0,1\}^{n-1}, b \in \{0,1\}. \tag{1}$$

Unwinding this inductive definition gives the following construction for any input length n.

The Goldreich-Goldwasser-Micali (GGM) pseudorandom function:

$$P(x_1 x_2 \dots x_n) = G_{x_1}(G_{x_2}(\cdots (G_{x_n}(K) \cdots)).$$

We will prove that P is a pseudorandom function by induction on n. The key is understanding the effect of transformation (1).

Lemma 5. *If P is an (s, q, ε)-pseudorandom function and G is an $(s + q(t + O(k)), \varepsilon')$-pseudorandom generator of size t then P' is an $(s - O((n+k)q^2), q, \varepsilon + q\varepsilon')$-pseudorandom function.*

The additional parameter q here refers to the number of oracle queries that the distinguisher makes: "(s, q, ε)-indistinguishable" means indistinguishable with advantage at most ε by a circuit of size at most s that makes at most q oracle queries. The number of queries can never exceed the circuit size, so q is by default upper bounded by s. This parameter plays an important role in the induction so it will be useful to track it separately from the size.

For the proof of correctness it is helpful to introduce the following concept. The *view* of an oracle ciruit C^F is the sequence of values $(F(x_1), F(x_2), \dots, F(x_q))$ that C receives in response to its queries $x_1, \dots, x_q$ issued to F.

To prove that $P' = G_b \circ P$ is a pseudorandom function we need to compare the view of the distinguisher D' interacting with P' to the view of D' interacting with a truly random function $R'\colon \{0,1\}^n \to \{0,1\}^k$. To do this we'll look at the hybrid function $G_b \circ R$ where $R\colon \{0,1\}^{n-1} \to \{0,1\}^k$ is a truly random function.

Claim 6. *If P is (s, q, ε)-pseudorandom then $G_b \circ P$ and $G_b \circ R$ are $(s - q(t + O(k)), q, \varepsilon)$-indistinguishable.*

Proof. A distinguisher D that tells $G_b \circ P$ from $G_b \circ R$ can be turned into a distinguisher D' that tells P from R by applying G_b to every query. More precisely, D' simulates D. Whenever D makes query $y = bx$ to its oracle, D' makes query x to its oracle and applies G_b to the answer. The view D'^F is identical to the view $D^{G_b \circ F}$ for any F, so D has the same distinguishing advantage as D' on the relevant oracles. The circuit D has to do all the work that D' does plus q additional evaluations of G and $O(k)$ extra work to select the correct half G_b so its size is larger by $q(t + O(k))$. □

Claim 7. *If G is an (s', ε')-pseudorandom generator then $G_b \circ R$ is an $(s' - O((n+k)q^2), q, q\varepsilon')$-pseudorandom function.*

To gain intuition about Claim 7 take $n = 2$ and consider a distinguisher that queries its oracle F at 000, 101, and 011. When $F = G_b \circ R$, the distinguisher sees the values $G_0(R(00))$, $G_1(R(01))$ and $G_0(R(11))$. Since $R(00)$, $R(01)$, and $R(11)$ are independent random values this is like seeing (part of) the output of three independent copies of G, which we would also expect to be pseudorandom. Here is a useful composition lemma that captures this intuition.

Lemma 8. *Let $X_1, \dots, X_q$ and $Y_1, \dots, Y_q$ be independent random variables. If X_i and Y_i are (s, ε)-indistinguishable for every i then $(X_1, \dots, X_q)$ and $(Y_1, \dots, Y_q)$ are $(s, q\varepsilon)$-indistinguishable.*

Before we do the proof of Claim 7 let's consider another example. Suppose the distinguisher now queries its oracle at 000, 101, and 100. The first and third value returned by $G_b \circ R$ are no longer independent: They are evaluations of the left and right half of G on the same seed 00. It would be convenient if we could forbid the distinguisher from making sequences of queries of this type. This can be achieved by forcing the distinguisher to make the queries in pairs $0x$, $1x$. If the distinguisher wants to know $F(bx)$, we ask it to remember both $F(0x)$ and $F(1x)$ for future reference. The view of such a distinguisher will always look like the sequence

$$F(0x_1), F(1x_1), F(0x_2), F(1x_2), \dots, F(0x_q), F(1x_q) \tag{2}$$

with $x_1, \dots, x_q$ all distinct.

Proof. Assume D distinguishes $G_b \circ R$ from a random function R'. We convert D into a distinguisher D^* that operates as follows: Whenever D queries its oracle at bx, D^* checks if it has made this query before and if so it returns its previous answer. If not, it queries its oracle at both $0x$ and $1x$ and uses the relevant answer. D^* is larger than D by $O((n+k)q^2)$ gates, which is the amount of circuit hardware it takes to implement the memorization of previous queries and answers.[2]

The view of D^* looks like sequence (2). When $F = G_b \circ R$ this is a sequence of q independent outputs of G, while when $F = R'$ these are q independent random strings. By Lemma 8, if D^*'s distinguishing advantage is $q\varepsilon'$ then the output of G is not (s', ε')-pseudorandom. □

To prove Lemma 5 we combine the two claims, setting s' to $s + q(t + O(k))$. We now apply the lemma inductively. Assume G is an $(s + q(t + O(k)), \varepsilon')$-pseudorandom generator. When $n = 1$, by Claim 7 the function P is $(s - O((1+k)q^2), q, q\varepsilon')$-pseudorandom. Applying Lemma 5 we get that when $n = 2$, P is $(s - O(1 + 2 + 2k)q^2, q, 2q\varepsilon')$-pseudorandom. Continuing this argument inductively, we conclude that for general n, P is $(s - O(n(n+k)q^2), q, nq\varepsilon')$-pseudorandom. After renaming parameters we obtain the main theorem of this lecture.

Theorem 9. *If G is an (s, ε)-pseudorandom generator then the GGM function is an $(s - qt - O((n(n+k)q^2), q, nq\varepsilon)$-pseudorandom function.*

What does this mean? To get a sense let's ignore the contribution of the terms n, k, t of "polynomial" magnitude. If we set q to a bit below $\sqrt{s}$ then we get that P is about $(s, \sqrt{s}, \sqrt{s}\varepsilon)$-pseudorandom. Since the number of queries never exceeds circuit size, P is in particular $(\sqrt{s}, \sqrt{s}\varepsilon)$-pseudorandom as promised.

While this level of security is quite reasonable, one weakness of the GGM construction is its efficiency: To calculate an output of P one needs to make n sequential calls to G. Natural proofs show that this is in some sense unavoidable: If a function family can be implemented in a parallel model of computation like bounded-depth circuits with AND/OR and possibly PARITY gates then it cannot be pseudorandom by Theorem 4.

[2] In more reasonable models of computations like RAM programs D^* would be more efficient and the loss in security would be lower.

Proof of Lemma 8. To keep the notation simple let's prove it for $q = 2$. Assume that some D of size at most s distinguishes (X_1, X_2) and (Y_1, Y_2) with advantage more than 2ε. Then D must distinguish (X_1, X_2) and (X_1, Y_2) or it must distinguish (X_1, Y_2) and (Y_1, Y_2) with advantage more than ε. The two cases are completely analogous so we focus on the first one. The bits X_1 can be fixed to the value that maximizes the advantage of D, giving a distinguisher of size s between X_2 and Y_2. More formally, if

$$\Pr[D(X_1, X_2) = 1] - \Pr[D(X_1, Y_2) = 1] > \varepsilon$$

then by averaging

$$\mathrm{E}_{X_1}\left[\Pr_{X_2}[D(X_1, X_2) = 1] - \Pr_{Y_2}[D(X_1, Y_2) = 1]\right] > \varepsilon$$

so there must be some particular choice $X_1 = x_1$ for which $D(x_1, \cdot)$ distinguishes X_2 from Y_2. □

5 Natural properties and efficient learning

By the equivalence between distinguishing and prediction from the last lecture, the ability to distinguish a function f from a random one implies the ability to predict f at a previously unseen value. Instead of proving a general theorem let us illustrate this with an example. Suppose that f is computable by a small-depth circuit. Then f is very likely to become constant under a $(n-1)$-random restriction, that is $f(x) = f(x')$ for most pairs of inputs x, x' that differ in one random coordinate. Thus the value $f(x)$ can be predicted by querying $f(x')$ where x' is obtained by flipping a random coordinate of x.

In contrast, a pseudorandom function cannot be learned, not owing to lack of information, but because of limitations in computational power. In the model of probably approximately correct (PAC) learning, a learning algorithm can observe input-output samples of the form $(x, f(x))$, where f is some efficient but unknown function. For example, f could be the "circuit of the mind" that takes as input some visual scene x and outputs 1 if x has a dog in it. The objective of the learning algorithm is, after being trained on some sequence of samples $(x_1, f(x_1)), \dots, (x_q, f(x_q))$, to make a prediction for $f(x)$ on some previously unseen sample x.

We can model this learning process as a game in which the following type of learner L interacts with an unknown concept f:

Training phase:

Query sample x_1. Receive example $f(x_1)$.
Query sample x_2. Receive example $f(x_2)$.
$\vdots$
Query sample x_q. Receive example $f(x_q)$.

Prediction phase:

Choose test sample x different from x_1 up to x_q.
Output a prediction y for $f(x)$.

The learner succeeds if y equals $f(x)$. This is an extremely weak notion of learning: Not only does the learner get to choose which examples to see, but she also chooses what to be tested on (as long as she doesn't get to train on the test question itself).

Theorem 10. *Assume $P_K\colon \{0,1\}^n \to \{0,1\}$ is an $(s, q+1, \varepsilon)$-pseudorandom function. For every learner L of size at most s there is a key K such that L^{P_K} succeeds with probability at most $1/2 + \varepsilon$.*

In words, no learner can do any better at passing the test than guessing the answer at random. Therefore, assuming pseudorandom generators exist, it is possible to design efficiently computable functions that are extremely hard to predict by any learning algorithm. Such functions have many uses in cryptography.

Proof. We prove the contrapositive. Suppose L^{P_K} succeeds with probability $1/2$ for every K. Then it succeeds with this probability for a random K. Let D be a distinguisher that simulates L, making the same q training queries, but then makes one more query at x and outputs L's exam grade: 1 if $y = f(x)$ and 0 if not. The accepting probability of D^P then equals the probability that L^P succeeds, which is $1/2 + \varepsilon$. On the other hand, if R is a truly random function, the value $R(x)$ is independent of everything the learner has seen, so it is independent of his prediction y. The probability that they are equal is exactly $1/2$, so D^R outputs 1 with probability exactly $1/2$. It follows that D distinguishes P and R, so P is not pseudorandom. □

References

The concept of a natural proof was proposed by Razborov and Rudich. Their paper contains an extensive study of all Boolean circuit lower bounds proved before 1995 and shows they all rely on constructive and small properties. There hasn't been substantial progress in circuit lower bounds for "explicit" functions since 1995. Theorem 9 was proved by Goldreich, Goldwasser, and Micali. The PAC learning model was invented by Valiant.