The million dollar question in computational complexity is "do all problems in NP have efficient algorithms?" The conventional belief is "probably not", 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 proofs of computational hardness are often *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. Many proofs of computational hardness do in fact show that all functions computed by the circuit model in question are efficiently learnable (in a precise sense that we will define shortly).

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 there is no cryptography.

1 Smallness and constructivity

The circuit lower bounds we proved in the first few lectures all followed the same basic pattern. First we specified a property that all functions in the circuit class must have. Then we argued that the hard function does not have this property.

For example, to show that PARITY on n inputs requires a decision tree of depth $d = n/2$, we used the following property of depth d decision trees: After fixing some d of the variables to suitably chosen values the function computed by the decision tree becomes a constant. PARITY does not have this property.

To show that PARITY requires decision trees of size $s = 2^{\Omega(n)}$, we used the fact that after applying a 1/2-random restriction a size s decision tree reduces to a depth less than d with probability at least $s \cdot (3/4)^d$. Setting d to equal $n/2$ reduces the problem of lower bounding the size to the solved problem of lower bounding the depth.

Smallness The property "$f\colon \{0,1\}^n \to \{0,1\}$ does not become a constant after fixing a random half of its inputs to random fixed values", which we denote by $P(f)$, certainly applies to the parity function, but it also applies (with high probability) to a *random* function R: After fixing half the inputs R is a random function of the remaining $n/2$ inputs, and the probability that this function is a constant is as small as $2^{-2^{n/2}}$. So our proof that parity requires size $2^{\Omega(n)}$ decision trees also proves that most functions on n inputs require size $2^{\Omega(n)}$ decision trees. Such properties are called *small* since they are shared by a tiny fraction of all functions.

Definition 1. A property P of boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ is *δ-small* if the probability

that a random function satisfies P is at most δ.

Think of δ as a small constant like 1/3. The proof that $PARITY$ on n bits requires depth d circuits of size $2^{\Omega(n^{1/(d-1)})}$ from Lecture 2 relied on a similar property: After restricting all but $n/(K \log s)^{d-1}$ inputs in a size s, depth d circuit, the circuit becomes a constant with constant probability. A calculation similar to the one we just did shows that the property "becomes constant after randomly restricting, say, $n-10$ inputs" is 2^{-10}-small. So the same proof shows that random functions require depth d circuits of size $2^{\Omega(n^{1/(d-1)})}$.

The proof that MAJORITY on n inputs requires depth-d circuits of size $2^{\Omega(n^{1/2d})}$ relied on the property that a size-s depth-d circuit can be approximated by a $\{0,1\}$-valued polynomial of degree at most $D = (\log O(s))^d$ within constant error (say 1/16). We can argue that when D is not too large a random function is unlikely to have this property. By a counting argument similar to the one in lecture 2 the total number of functions of the form (degree-D polynomial) + (small error ε) on n bits is at most $2^{(H(D/n)+H(\varepsilon))\cdot 2^n}$, which (even when D is quite large) is much smaller than the total number of functions 2^{2^n}. In conclusion, the property "can be approximated by a degree-D polynomial with error 1/16 is small.

The proof that the inner product modulo 2 function on $2n$ bits requires read-k-times randomized branching programs of width $w = \Omega(2^{n/2k})$ relied on the following property: For every function $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ computable by a width w, read-k-times randomized branching program there exist set X and Y such that $|X| \cdot |Y| \geq 2^{2n}/2w^{2k}$ such that f is constant on a 2/3 fraction of entries in $X \times Y$ (assuming error 1/3). We then argued that the inner product function does not have this property as long as $|X| \cdot |Y|$ is larger than 2^n by a constant factor. It is again possible to show that the property "is almost constant on $X \times Y$ for every X, Y such that $|X| \cdot |Y| \geq K$" for say $K = 100 \cdot 2^n$ is small. You will work this out in the homework.

To summarize, all of the lower bound proofs we saw followed the same pattern: We described some property of functions, showed that it holds for all functions computed by small circuits, and then argued that it does not hold for the hard function in question. It also happens that the property does not hold for most functions, so the proof also shows that random functions are hard for the model in question. This is not surprising as we know by a counting argument that most functions should in fact be hard for any "reasonable" circuit model. It can be proved that any property based on a "formal complexity measure" — a concept we won't define — is small as long as it is not satisfied by all functions.

Constructivity Constructivity postulates that the relevant lower bound property is efficiently computable: A relatively small circuit that can evaluate 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*. Instead (or in addition to) taking an actual input, oracle circuits access their output by special *oracle gates* with n input wires and one output wire that, when given input x, 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 (randomized) oracle circuits of size at most S given the ability to evaluate f at inputs of its choice.

It turns out (for not completely understood reasons) that properties behind proofs of Boolean circuit

lower boudns are $2^{O(n)}$-constructive. (The "input" f is a 2^n-bit long string, so "$2^{O(n)}$-efficient" is polynomial in the input length.)

Let's consider the example of decision trees and small-depth AND/OR circuits first. The relevant property "f becomes constant after a random $(n-t)$-restriction" can be tested by a random circuit of size about 2^t. After picking a random restriction, the circuit has to evaluate the restricted function on its 2^t inputs and confirm that all values are equal.

The property behind the branching program lower bound is a bit more tricky. It is not clear that "f is constant on a 2/3 fraction of entries of $X \times Y$ for some X, Y such that $|X| \cdot |Y| \geq K$" is something that can be computed in size $2^{O(n)}$ as it might require looking at all pairs of sets (X, Y) of size (at least) K, of which there are $2^{O(n \log K)}$.

Let's take a step back and remember how we proved that the inner product function has this property. We derived it from the stronger statement that

$$\left|\sum_{x,y \in \{0,1\}^n} p(x) \cdot (-1)^{\langle x,y \rangle} \cdot q(y)\right| \leq \sqrt{\frac{2^n}{|X||Y|}}$$

where p and q are the probability mass functions of the uniform distributions on the sets X and Y. The proof we gave in Lecture 3 in fact works not only for probability mass functions, but for any two functions p and q of ℓ_2 norm $1/\sqrt{|X|}$ and $1/\sqrt{|Y|}$, respectively. In fact, our argument from Theorem 14 in Lecture 3 more generally proves that

$$\left|\sum_{x,y \in \{0,1\}^n} p(x) \cdot (-1)^{f(x,y)} \cdot q(y)\right| \leq 2^{n/2} \cdot \sqrt{\sum_{x \in \{0,1\}^n} p(x)^2} \cdot \sqrt{\sum_{x \in \{0,1\}^n} q(x)^2}$$

for all functions $p, q \colon \{0,1\}^n \to \mathbb{R}$. This says that the largest singular value of the $2^n \times 2^n$ matrix $F(x,y) = (-1)^{f(x,y)}$ is at most $2^{n/2}$. Singular values of a 2^n by 2^n matrix are computable in time polynomial in 2^n, so stated in this way the property becomes $2^{O(n)}$-constructive.

To summarize, the property $P(f) =$ "$F(x,y) = (-1)^{f(x,y)}$ has a singular value larger than $2^{\alpha n}$" is sufficient to prove that F requires read-k-times branching programs of width $\Omega(2^{\alpha n/k})$. The property P is constructive. It turns out that it is also large (although we won't prove it): The largest singular value of a random $2^n \times 2^n$ ± 1 matrix is $O(2^{n/2})$ with high probability.

The property "$f = p + e$ for a low-degree p and sparse e" behind the AND/OR/PARITY lower bound might similarly not be constructive, but can be replaced by one that is both small and constructive.

2 Pseudorandom Functions

To explain what a pseudorandom function is, we should say a bit more about (truly) random functions. Such a function can be uniquely specified by listing all 2^n values $F(\mathtt{00\cdots0}), \ldots, F(\mathtt{11\cdots1})$. Since 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 $R(\mathtt{00\cdots0}), \ldots, R(\mathtt{11\cdots1})$ 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_1x_2 and randomness $r_1r_2r_3r_4$, output

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

This type of implementation scales exponentially in n. Intuitively this should be unavoidable because any implementation of R must store 2^n independent random values $r_1, \ldots r_{2^n}$. 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.

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

$$(P(\mathtt{00}\cdots\mathtt{0}), P(\mathtt{00}\cdots\mathtt{1}), \ldots, P(\mathtt{11}\cdots\mathtt{1})) \quad \text{and} \quad (R(\mathtt{00}\cdots\mathtt{0}), R(\mathtt{00}\cdots\mathtt{1}), \ldots, R(\mathtt{11}\cdots\mathtt{1}))$$

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 Pseudorandom functions and natural proofs

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 all functions $f\colon \{0,1\}^n \to \{0,1\}$ in some class $\mathcal{C}$ satisfy some (possibly randomized) property that is δ-small and S-constructive with probability at least $1-\delta$ then $\mathcal{C}$ cannot compute $(S, 1-2\delta)$-pseudorandom functions.*

Proof. Let D be a circuit computing then property in question. By smallness $\Pr[D^R = 1] \le \delta$. Since all functions in $\mathcal{C}$ have the property with probability $1-\delta$, for any distribution on F in $\mathcal{C}$, $\Pr[D^P = 1] \ge 1-\delta$. So

$$\Pr_F[D^P = 1] - \Pr[D^R = 1] \ge 1 - 2\delta.$$

Since the property is computable by size-S circuits S, $\mathcal{C}$ cannot compute $(S, 1-2\delta)$-pseudorandom functions. □

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

In the next section we show how to construct a $(s, \sqrt{s}\varepsilon)$ pseudorandom function $P\colon \{0,1\}^n \to \{0,1\}$ from a (s,ε)-pseudorandom generator G. If, for example, $s = 2^{k/2}$ and $\varepsilon = 2^{-k/2}$ we get a $(2^{k/4}, 2^{-k/4})$-pseudorandom function P from a $(2^{k/2}, 2^{-k/2})$-pseudorandom generator G. Therefore efficient pseudorandom functions quite plausibly exist, and so it is unlikely that natural proofs can succeed in proving circuit lower bounds against general polynomial-size circuits.

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 $(P_K(x_1), \ldots, P_K(x_{2k}))$. 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 Lecture 2. 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:

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

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 $(F(x_1), F(x_2), \ldots, F(x_q))$ that C receives in response to its queries $x_1, \ldots, x_q$ 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 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 $X_1, \ldots, X_q$ and $Y_1, \ldots, Y_q$ be independent random variables. If X_i and Y_i are (s,ε)-indistinguishable for every i then $(X_1, \ldots, X_q)$ and $(Y_1, \ldots, Y_q)$ 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

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

with $x_1, \ldots, x_q$ 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. □

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

To prove Lemma 5 we combine the two claims, setting s' to $s+q(t+O(k))$ (this will turn out to be a convenient choice). 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.

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 likely to become constant under a random restriction, that is $f|_\rho(x)=f|_\rho(x')$ for most random restrictions ρ and a random pair of inputs x and x'. If we call y and y' the inputs to f that restrict to x and x' under ρ, then the value $f(y')$ can be predicted from the value $f(y)$: Most of the time they are the same.

In contrast, a pseudorandom function cannot be learned, not by 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 $(x_1, f(x_1)), \ldots, (x_q, f(x_q))$, 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 see the exam question beforehand)!

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, the learner cannot do any better at passing the test than guessing the answer at random!

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

Natural proofs were introduced and studied 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. Theorem 9 was proved by Goldreich, Goldwasser, and Micali.