CS 395T Computational Learning Theory

Lecture 20: November 19, 2008 — Lecturer: Adam Klivans

Scribe: Vishvas Vasuki

20.1 Introduction to hardness of learning results

There are two classes of hardness of learning results:

1. Hardness results for proper learning: Usually, using the $RP \neq NP$ assumption, we prove that proper learning a representation class is hard. For example, for $k \geq 3$, learning k-term DNF formulae and producing a k-term DNF as a hypothesis is intractable.
2. Cryptographic hardness of learning results: Here, typically using the assumption that factoring is hard, you show that a certain concept class is hard to learn, even if the learner is allowed to produce a hypothesis which does not belong to the same representation class as the target concept.

In this lecture, we will show the following cryptographic hardness of learning result: If factoring is hard, learning the concept class of polynomial sized circuits of log depth is hard.

First, we make some definitions.

20.2 Introducing some notions

Definition 1 *A Function Family is an exponential sized set $F = \{f_i | i \in I\}$ of polynomial sized boolean circuits with input length n, equipped with a samplable index set I; such that there exists an algorithm S which does the following:*

S accepts as input $i \in I$ and simulates the input/ output behavior of f_i: that is, it accepts x and returns $f_i(x)$ in polynomial time.

Definition 2 *Let RAND be the set the set of all boolean functions over $\{0, 1\}^n$.*

Definition 3 *A Distinguisher D is a polynomial time algorithm which, when given black box access to a function f, outputs 1 or 0.*

In the context of the present lecture, D will output 1 if it thinks that f is not chosen uniformly at random from RAND.

Definition 4 *A funciton family F is Pseudorandom Function Family (PFF) if for every distringuisher D, $Pr_{f \in_U RAND}(D^f = 1) - Pr_{i \in_U I}(D^{f_i} = 1) < O(e^{-n})$. This property of F is called the Indistinguishability property.*

The following notion, from David Zuckerman's Randomized Algorithms course is also helpful.

Definition 5 *A function* $G : \{0,1\}^l \rightarrow \{0,1\}^n$*, computable in time poly(l), is an* $(\epsilon, s(n))$ *Pseudo-random Generator if, for all circuits c of size s(n), the following property holds:* $Pr_{y\in\{0,1\}^n}[c(y) = 1] - Pr_{x\in\{0,1\}^l}[c(G(x)) = 1] \leq \epsilon$.

Fact 1 *From a result due to Goldreich, Goldwasser and Micali, we know that if one way functions exist (that is, if factoring is hard), then pseudorandom function families exist.*

Definition 6 *The Blum-Blum-Shub (BBS) pseudorandom generator is an algorithm with the following behavior:*

1. *It accepts as input the following:*

   An n bit integer $N = pq$*, where p and q are prime numbers which are equivalent to* $3 \bmod 4$.

   An intial seed s_0 *of length n bits.*

2. *It outputs a stream of poly(n) bits* b_i*, each of which is the least significant bit of the number* s_i *calculated as follows:* $s_i = s_{i-1}^2 \bmod N = s_0^{2^i} \bmod N$.

Fact 2 *If factoring is hard, no polynomial time algorithm can distinguish between a truly random m bit string and an m bit string obtained by choosing the seed* s_0 *at random and running a BBS generator.*

20.3 Hardness of learning circuits which compute the ith bit of the output of a BBS generator

Definition 7 *Let* $\mathbb{C}$ *represent any circuit class which contains circuits* $f_{s_0,N,t}$ *with the following behavior:*

1. $\forall i > t : f_{s_0,N,t}(i) = 0$.
2. $\forall i \leq t : f_{s_0,N,t}(i) = b_i$*, the ith bit output by the BBS pseudorandom generator specified by N and the seed* s_0.

Theorem 1 *If* $\mathbb{C}$ *is efficiently learnable, then the BBS generator can be broken.*

Sketch of Proof *If* $\mathbb{C}$ *is efficiently learnable, then there exists an* $O(n^{ck})$ *time algorithm A to learn* $\mathbb{C}$ *with error* $\leq 2^{-1} - n^{-k}$*; where k and c are constants. Let d be any integer such that* $dc \neq 1$.

We show that, using A, you can build a distinguisher D which, given a string b of $n^{(d+1)ck}$ *bits, can distinguish a BBS generated string from random string. This distinguisher works as follows:*

Let b_i be the ith bit of b. Then, tuples of the form (i, b_i) are referred to as examples. Using the Uniform Distribution over the examples, D draws n^{ck} examples. Using A with this sample, D then obtains a hypothesis h with error $\leq 2^{-1} - n^{-k}$.

D then picks uniformly at random another bit index j. It then tries predicting b_j using h. If its guess turns out to be correct, it outputs 1, which stands for the identification of b as the output of a 'generator'.

On truly random b, $Pr(D^{rand} = 1) \geq 2^{-1} + \frac{n^{ck}}{n^{(d+1)ck}}$; but $Pr(D^{f_{s_0,N,t}} = 1) \geq 2^{-1} + n^{-k}$. The difference between these, $n^{-dck} - n^{-k}$ is not negligible. ■

20.4 Hardness of learning small cicrcuits

Let the order of the group Z_N^* be $\varphi(N) = (p-1)(q-1)$.

Consider the circuit $f_{s_0,N,t}$. On input i, it needs to compute $f_{s_0,N,t}(i) = s_0^{2^i} \bmod N = s_0^{2^i \bmod \varphi(N)} \bmod N$.

If we know the precomputed values of $2^0, 2^1, 2^2 .. \bmod \varphi(N)$, given any number k, we can find $j = 2^k \bmod \varphi(N)$ by multiplying together the appropriate precomputed powers of 2. Similarly, if we know precomputed values $s_0^0, s_0^1, s_0^2 .. \bmod N$, we can find s_0^j for any j by multiplying together the appropriate powers of s_0.

Thus, our circuit to compute $f_{s_0,N,t}$ must be able to remember these precomputed values, and should be able to multiply n n-bit numbers. Thus, $f_{s_0,N,t}$ can be realized using a polynomial sized circuit of depth $O(\log n)$.

Thus, using the theorem we proved earlier, we see that classes of circuits of polynomial size and log depth are hard to learn.