U.C. Berkeley
CS294: Pseudorandomness and Combinatorial Constructions
Professor Luca Trevisan

Handout N8
September 22, 2005
Scribe: Madhur Tulsiani

Notes for Lecture 8

1 The Goldreich-Levin Theorem: Learning Linear Functions

The Goldreich-Levin theorem proves existence of probabilistic learning algorithms for linear functions. An application of the theorem also proves existence of hard-core predicates for one way permutations.

One way of describing the problem is to consider a function $f : \{0,1\}^n \to \{0,1\}$ which agrees with a linear function $l_a(x) : \{0,1\}^n \to \{0,1\}$ at a $(\frac{1}{2} + \epsilon)$ fraction of the inputs. The function $l_a(\cdot)$ is defined as

$$l_a(x) := a \cdot x = \sum_{i=1}^{n} a_i \cdot x_i \pmod 2$$

Then the problem is to find all possible candidates for the bit-vector a given oracle access to the function f. This can also be viewed as a problem in error-correcting codes as the function $l_a(\cdot)$ is nothing but the Hadamard code for a and $f(\cdot)$ represents a version of the codeword with upto $(\frac{1}{2} - \epsilon)$ fraction of bits corrupted. It is required to find all possible values for the original message.

The theorem says that it is possible to do so with high probability of success in time polynomial in n and $1/\epsilon$. Formally:

Theorem 1 (Goldreich-Levin) *There is a probabilstic algorithm such that given $f : \{0,1\}^n \to \{0,1\}$ and $\epsilon > 0$ it runs in time poly(n, ϵ) and outputs a list L so that for every $a \in \{0,1\}^n$*

$$\mathop{\mathbf{Pr}}_{x \sim \{0,1\}^n} \left[\sum_{i=1}^{n} a_i x_i = f(x) \right] \geq \frac{1}{2} + \epsilon \quad \Rightarrow \quad \mathbf{Pr}[a \in L] \geq \frac{1}{2}$$

2 The case of low-error data

We first consider the special case when f is guaranteed to be correct (i.e. have value equal to l_a) on at least 7/8th of the inputs (The treatment here can actually be generalized to the case of agreement on $(\frac{3}{4} + \epsilon)$ fraction of inputs). Since, the distance between two functions is a metric, using triangle inequality shows that there must be a unique function $l_a(\cdot)$ and hence a unique a for this case. The following algorithms finds the solution with high probability (e_i represents the n-bit vector with 1 only in the i^{th} position):

```
Find-Unique-Function()
  for i = 1 to n do
    for j = 1 to t do
      Pick an x ∈ {0,1}^n uniformly at random
      Compute a_ij = f(x + e_i) − f(e_i)
    Take a_i = maj(a_i1, ..., a_it)
```

Note that in each iteration of the inner loop, each of $f(x+e_i)$ and $f(x)$ is correct with probability at least 7/8. Thus, the probability of finding the correct value of a_i is at least 3/4 for each iteration of the inner loop. This gives that probability of error in the majority over t trials is at most $e^{-\Omega(t)}$ by Chernoff bounds. Using this for all i we have:

$$\mathbf{Pr}[a_i \text{ is correct for all i}] \geq (1 - e^{-\Omega(t)})^n$$

Taking $t = O(\log n)$ with an appropriate constant gives an $O(n \log n)$ algorithm with arbitrarily small error. We assume this to be bounded by some constant, say δ.

3 "Almost an algorithm" for the general case

The previous algorithm assumes that the oracle gives the correct value of $l_a(\cdot)$ at least 7/8th of the time. We now try to construct a subroutine which does that. However, we assume that this function has the correct values of $l_a(\cdot)$ at a few points "hard-coded" into it. The values are assumed to be obtainable from some other oracle.

To construct the routine, we proceed as follows:

- Pick $X^{(1)}, \ldots, X^{(k)} \in \{0,1\}^n$ independently at random
- Obtain $l_a(X^{(1)}), \ldots, l_a(X^{(k)})$ from the other oracle
- Define the function $A_{X^{(1)},\ldots X^{(k)}}$ as:

$$A_{X^{(1)},\ldots,X^{(k)}}(z) := maj[(f(z+X^{(1)}) - l_a(X^{(1)})), \ldots, (f(z+X^{(k)}) - l_a(X^{(k)}))]$$

To analyze the probability of correctness of this function, we define: the variables $E_1, \ldots, E_k$ where $E_j = 1$ when $f(z+X^{(j)}) = l_a(z+X^{(j)})$ and 0 otherwise. Thus, $\mathbb{E}[E_j] \geq (\frac{1}{2} + \epsilon)$. Also

$$\begin{aligned}
\mathop{\mathbf{Pr}}_{X^{(1)},\ldots,X^{(k)},z}[A_{X^{(1)},\ldots,X^{(k)}}(z) \neq l_a(z)] &= \mathbf{Pr}\left[\sum_j E_j < k/2\right] \\
&\leq \mathbf{Pr}\left[\left|\sum_j E_j - \mathbb{E}\left[\sum_j E_j\right]\right| < \epsilon k\right] \\
&\leq \frac{\mathbf{Var}[\sum_j E_j]}{\epsilon^2 k^2} \quad \text{(using Chebyshev's inequality)} \\
&= \frac{\sum_j \mathbf{Var}[E_j]}{\epsilon^2 k^2} \quad \text{(since the X}'_\text{j}\text{s are pairwise independent)} \\
&= \frac{1}{4\epsilon^2 k^2}
\end{aligned}$$

Taking $k = 4/\epsilon^2$ gives

$$\begin{aligned}
&\mathop{\mathbf{Pr}}_{X^{(1)},\ldots,X^{(k)},z}[A_{X^{(1)},\ldots,X^{(k)}}(z) \neq l_a(z)] \geq \frac{1}{16} \\
\Rightarrow &\mathop{\mathbf{Pr}}_{X^{(1)},\ldots,X^{(k)}}\left[\mathop{\mathbf{Pr}}_{z}[A_{X^{(1)},\ldots,X^{(k)}}(z) \neq l_a(z)] \leq \frac{1}{8}\right] \geq \frac{1}{2} \quad \text{(using Markov's inequality)}
\end{aligned}$$

This shows that if we just randomly pick $X^{(1)}, \ldots, X^{(k)}$ and create the subroutine $A_{X^{(1)},\ldots X^{(k)}}()$, then with high probability it provides us with low-error-data. So it is possible use the algorithm Find-Unique-Solution() with the only modification of using $A_{X^{(1)},\ldots X^{(k)}}$ to find values of f instead of querying the oracle. Since the complexity of $A_{X^{(1)},\ldots X^{(k)}}()$ is $O(\frac{1}{\epsilon^2})$, the total complexity now becomes $O(\frac{1}{\epsilon^2} n \log n)$. The probability of success is at least $\frac{1}{2} - \delta$, which can be amplified (to above $\frac{1}{2}$ as stated in the theorem) by repeating the algorithm a constant number of times.

4 The final idea - using pseudorandomness!

Since the enigmatic "other" oracle which facilitated the construction of $A_{X^{(1)},\ldots X^{(k)}}$ does not actually exist, we need to try all possible guesses for the values of $l_a(\cdot)$ at each of these points. However, this would make the complexity exponential in $1/\epsilon$ which is clearly undesirable.

A better solution may be obtained by observing that in the previous proof we do not actually require the points $X^{(1)}, \ldots, X^{(k)}$ to be uniformly random but just pairwise independent. The following construction gives us the value of $l_a(\cdot)$ at k pairwise independent points using only $\log k$ guesses (for proof refer to lecture 1).

- Pick $X_1, \ldots, X_t$ (where $t = \log k + 1$).
- $\forall S \subseteq 1, 2, \ldots, t,\ S \neq \emptyset$ define $X^S = \sum_{i \in S} X_i$ where summation represents bitwise addition modulo 2. Then the $2k - 1$ variables generated are all pairwise independent - we may use any k of them for the algorithm.
- Since l_a is a linear function for bit-vectors, we note that $l_a(X^S) = \sum_{i \in S} l_a(X_i)$

The above construction requires to know the value of $l_a(\cdot)$ at only $\log k + 1$ points. Thus, for these we may try all possible guesses which increase the running time by a factor of $2^t = 2k = O(\frac{1}{\epsilon^2})$. This then completes the algorithm and the overall time complexity is $O(\frac{1}{\epsilon^4} n \log n)$.

Combining the ideas so far, we get the final algorithm as follows:

```
Find-List()
  Pick X_1, ..., X_t independently and uniformly at random
  for every (b_1, ..., b_t) ∈ {0,1}^t do
    for every S ⊆ 1, ..., t, S ≠ ∅ do
      Assign X^S = Σ_{i∈S} X_i
      Assign b^S = Σ_{i∈S} b_i
    Define A(z) = maj_{S⊆1,...,tS≠∅} f(X^S + z) − b^S
    for i = 1 to n do
      for j = 1 to c log n do
        Pick z ∈ {0,1}^n uniformly at random
        Compute a_ij = A(z + e_i) − A(z)
      Take a_i = maj(a_i1, ..., a_ij)
    Add a to the list L
  Output L
```

5 Producing hard-core predicates

We know that pseudorandom generators can be constructed assuming existence of one way permutations having a hard-core predicate. The Goldreich-Levin theorem allows us to to weaken this assumption. We show that every one-way permutation allows us to define another one-way permutation having a hard-core predicate. Formally, we intend to prove the following:

Lemma 2 *Let $p : \{0,1\}^n \to \{0,1\}^n$ be an (S, ϵ) one-way permutation computable by a circuit of size t. Then $q : \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined as $q(x,y) := (p(x), y)$ is another one-way permutation for which the predicate $B(x,y) := x \cdot y$ is (S', ϵ') hard-core. The sizes S and S' are polynomially related.*

Proof: It is easy to see that $q(x,y)$ is a permutation. It is also one-way because a circuit which inverts it will also be able to invert $p(x)$. Suppose that $B(x,y)$ is not (S', ϵ') hard-core for $q(\cdot)$. This means that $\exists A$ (a circuit) with $A = |S'|$ s.t.

$$\begin{aligned} &\Pr_{x,y \sim \{0,1\}^n}[A(p(x), y) = x \cdot y] \geq \frac{1}{2} + \epsilon \\ &\Rightarrow \Pr_{y|x}[A(p(x), y) = x \cdot y] \geq \frac{1}{2} + \epsilon \qquad \text{(As the distribution is uniform)} \end{aligned}$$

Thus, for a given x, $A(p(x), y)$ defines a function on $\{0,1\}^n$ which agrees with $x.y$ with a positive bias. Thus, given $p(x)$, we proceed as follows to invert it:

- Define $f(y) := A(p(x), y)$.
- Compute the list L of the possible values of x using the previous algorithm.
- $\forall a \in L$ compute $p(a)$ and output a if $p(a) = p(x)$.

Thus, each query to the oracle is now replaced by an $O(S')$ circuit. This requires a time of $O(S' \frac{1}{\epsilon^4} n \log n)$ for producing the list and $O(\frac{1}{\epsilon^2})$ time for the check in the last step. So the total running time of the above procedure is $O(S' \frac{1}{\epsilon^4} n \log n + \frac{t}{\epsilon^2})$. Choosing S' appropriately so that this is less that S, we arrive at a contradiction. This proves the desired result. □