CS 2401 - Introduction to Complexity Theory

Lecture #9: Fall, 2015

Lecturer: Toniann Pitassi

Scribe Notes by: Atiyeh Ashari Ghomi

1 Review

In the last class, we gave lower bounds for AC^0_d circuits computing PARITY. For n sufficiently large, any family of $\{\mathrm{C_n}\}$ of depth d circuits of size $\leq 2^{n^{O(\frac{1}{d})}}$ has

$$|\Pr[\mathrm{C_n(x) = PARITY(x)}] - \frac{1}{2}| \leq 2^{-\Omega(n^{\frac{1}{d}})}$$

Which means any small circuit makes mistake on at least one bit. Now, we will see three applications of the AC^0_d parity lower bound.

2 Pseudorandom Generator

There are different versions of pseudorandom generator. Some of them can be computed with polynomial time Turing machines. They take $\log n$ bits as a seed and produce $y_1, y_2, \ldots, y_m$ for approximation. With these m bits we can check:

$$x \in L \Rightarrow \Pr_y[C(x) = 1] \geq \frac{3}{4}$$

$$x \notin L \Rightarrow \Pr_y[C(x) = 1] \leq \frac{1}{4}$$

Because the seed is of length $\log(n)$ and the generator runs in polynomial time, so we can check the circuit for every seed in polynomial time and see if the number of times C returns 1 is more than $\frac{3}{4}$, $x \in L$. But these generators require a strong unproven assumption that P is separated from NP. In this class, we see a pseudorandom generator (proposed in [1]) with quasilinear running time which avoids this problem.

Theorem 1 *$\forall d$ there is a family of functions, $\{g_n : \{0,1\}^l \to \{0,1\}^n\}$ where $l = O((\log n)^{2d+6})$ (seed and running time is $n^{poly(\log n)}$ which is quasilinear), such that:*

- *$\{g(n)\}$ is computed by log-space uniform circuits of polynomial size, depth $d + 4$,*
- *$\forall C_n$ of polynomial size, depth d, $\forall$ polynomial p(n):*

$$|\Pr_{y \in \{0,1\}^n}[C_n(y) = 1] - \Pr_{x \in \{0,1\}^l}[C_n(g_n(x)) = 1]| \leq \frac{1}{p(n)}$$

where y is chosen uniformly in $\{0,1\}^n$ and x is chosen uniformly in $\{0,1\}^l$

Proof *(sketch)* To create this $g_n(x)$, we define a collection of sets $\{S_1, \ldots, S_n\}$. Each $S_i \subseteq [l]$

- $|S_i| = (\log n)^{d+3}$
- $|S_i \cap S_j| \leq \log n, \forall i \neq j$

We set $g_n(x) = parity(x|_{S_1})parity(x|_{S_2}) \ldots parity(x|_{S_n})$. Every S_i gives one bit parity and $g_n(x)$ is concatenation of these bits. We want these parity bits to be independent, so we don't want these subsets to have too much in common, but we need them to be large enough. Note that:

- The generator can be computed by polynomial size circuits of depth $d+4$ since it is just the parity of sets of bits of cardinality $(\log n)^{d+3}$
- $|\Pr_{y\in\{0,1\}^n}[C_n(y) = 1] - \Pr_{x\in\{0,1\}^l}[C_n(g_n(x)) = 1]| \leq \frac{1}{p(n)}$ since otherwise, according to [2][3] one of the bits of $g_n(x)$, let's say i, can be predicted from previous ones. Using the circuit that can predict i and the fact that $|S_i \cap S_j| \leq \log n, \forall i \neq j$, we can build a circuit that computes the $parity(x|_{S_i})$ with size and error contradictory to AC^0_d lower bound.

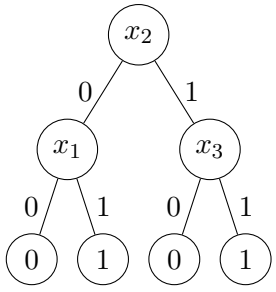
3 Satisfiability Algorithms and #SAT Algorithms

Definition kCNF_SAT: Given kCNF formula $f = C_1 \wedge \ldots \wedge C_m$, $|C_i| \leq k$, is it satisfiable?

Definition #kCNF_SAT: Given kCNF formula, output the number of satisfiability assignment?

One way of solving these problems is using brute force algorithm with runtime of $2^n poly(m)$ where k is constant so m is $poly(n)$ because m is less than $\sum_{i=1}^{k}\binom{2n}{i}$. We sill get (ZPP algorithm with zero error) algorithm for both kCNF_SAT and #kCNF_SAT with expected runtime $2^{n(1-\frac{1}{k})}poly(m)$.

Let f be a boolean function, we build a decision tree for f by creating a binary tree with internal nodes label with the variables of f. At each node by visiting the left child we assign zero to the variable at this node, and by visiting the right child we assign one to this variable. An example of SAT decision tree for formula $f = (x_1 \vee x_2)(\overline{x_2} \vee x_3)(x_1 \vee x_2 \vee \overline{x_3})$:



Here after assigning zero to x_2 and simplifying f we have, $(x_1)(x_1 \vee \overline{x_3})$. By assigning zero to x_1, f returns zero, so we need to backtrack.

Lemma 2 *Switching Lemma*

Let f be a rCNF*, let ρ be a random restriction: $\rho : \{x_1, \ldots, x_n\} \to \{0, 1, *\}$, ρ sets pn variables $*$'s*

Pr_ρ[canonical decision tree of $(f \restriction_\rho)$ has depth $\geq s$] $\leq (8pr)^s$

If we set $p \sim \frac{1}{16r}$, then $(8pr)^s = (\frac{1}{2})^s$

ρ partitions, at random, the variables into two groups: $|x| = n - pn$ variables, each one is either 0 or 1, and $|Y| = pn$ variables unset. We build a decision tree (of size $2^{n(1-p)}$) to set variables in X, and for each setting we look in the restriction ρ. Then create canonical decision tree (according to switching lemma these trees' height would be small) and count the number of satisfying settings. Counting the number of satisfying assignment is easy, at each branch that is one, add 2 times the number of unset variables. When the algorithm running time is proportional to the canonical decision tree, we can calculate the expected time. The expected time is equal $\sum_{\text{all settings of x}} \times$O(size of the decision tree restricted by lemma). We have random partitioning and random setting so this expectation should be small according to the switching lemma. The running time of this algorithm is $2^{n-pn} \times$small tree size. This is the best algorithm for $\#SAT$.

4 PAC learning algorithm for AC^0_d circuits

One of the algorithmic uses of the switching lemma is learning depth of formulas using Fourier Approximation. Switching lemma gives us a lower bound on depth d, unbounded fan-in formulas and that's what our algorithm is going to receive here, but instead of receiving the formula, it has to learn what the formula is. In other words, the algorithm gets access to some device containing the formula rather than the formula itself.

Definition PAC: Probably approximately correct (ϵ, δ), is a framework for mathematical analysis of machine learning. It was proposed in 1984 by Leslie Valiant.[4]

In the training, the algorithm obtains $(x_1, f(x_1)), \ldots, (x_T, f(x_T))$ where x_i is uniformly random generated inputs and we have the value of f at these inputs. After seeing a certain number of random samples, we want to generate the hypothesis that might not look like f, but we want the probability of success (hypothesis looking like f) to be high. In other words, The goal is that, with high probability (the "probably" part), the selected function will have low generalization error (the "approximately correct" part). The learner must be able to learn the concept given any arbitrary approximation ratio, probability of success, or distribution of the samples.

Definition Concept class
A concept class is a family of functions over instance space. Examples of concept classes are: DNF formulas of size $poly(n)$, AC^0_d formulas of size $poly(n)$ and p/poly of size $poly(n)$.

Definition Learning algorithm
Learning algorithm A (with underlying probability distribution D on $\{o,1\}^n$, $f_n \in C_n$) given samples $(x, f_n(x))$, x drawn randomly from D, A outputs "hypothesis", $h_n : \{0,1\}^n \to \{0,1\}$. A is (ϵ, δ) correct if: $\forall f_n \in C_n, \forall D$ with probability $\geq (1-\delta)$,

$$Error_D = Pr_{x \in D}[h_n(x) \neq f_n(x)] \leq \epsilon$$

Ideally, we would like a running time polynomial in $\frac{1}{\epsilon}$ and the circuit complexity of f, but the algorithm presented in this class is going to be quasi-polynomial in the parameters. The technique used is based on the using Fourier representation of Boolean functions and giving an approximation to the function in the Fourier basis.

Theorem 3 *LMN*
AC^0_d *is PAC learnable in quasilinear time in the uniform distribution.*

Fourier representation has two ways of representing a function:

True	1	-1
False	0	1

$$\text{Parity function} = \begin{cases} \sum x_i & \text{in 0, 1 representation} \\ \prod x_i & \text{in 1, -1 representation} \end{cases}$$

The first one is an algebraic representation and the second one is geometric representation. The second one is cleaner for parity function because it's going to be useful when thinking about functions as polynomials. Here, we switch between two representation. If we view functions as 2^n length vectors over $\{1,-1\}^n$, we have:

$$F(x) = \begin{array}{<{\quad}ccccc>{\quad}} 1 & \text{-1} & \dots & \text{-1} & \\ F(0\dots0) & F(0\dots1) & \dots & F(1\dots1) & \end{array}$$

We can think about it as embedded to a 2^n-dimensional space. The size of vector is $\sqrt{\sum_x |F(x)|} = 2^{\frac{n}{2}}$, so by multiplying by $\frac{1}{2^{\frac{n}{2}}}$, we make length equal to 1.

To see how two functions are similar we calculate their dot product.

$$< \vec{F}, \vec{G} >= \vec{F}.\vec{G} = \frac{1}{2^{\frac{n}{2}}}\frac{1}{2^{\frac{n}{2}}}\sum_x F(x)G(x) = \begin{cases} 1 & \text{if } F(x) = G(x) \\ -1 & \text{if } F(x) \neq G(x) \end{cases}$$

$$= \frac{1}{2^n}[(\text{number of x's where } F(x) = G(x)) - (\text{number of x's where } F(x) \neq G(x))]$$

$$= [\Pr[F(x) = G(x)] - \Pr[F(x) \neq G(x)]]$$

$$= 1 - 2\Pr[F(x) \neq G(x)]$$

If they are very similar the dot product is close to 1 and if they are ati-correlated the dot product is close to -1. If there is no correlation between them the dot product is zero. The last one happens when $\Pr[F(x) \neq G(x)] = \Pr[F(x) = G(x)] = \frac{1}{2}$

In linear algebra, we can look at points under different basis. We can pick any orthonormal basis which is set of Boolean functions $F_1, F_2, \dots, F_{2^n}$ with $< F_i, F_j >= 0$ where $i \neq j$, and do an orthogonal transformation (this is like a rotation in space).

We define $\chi_S = \prod_{i \in S} x_i$ which is parity function of input x restricted to subset S. These parity functions are orthogonal because $\Pr[\chi_S(x) = \chi_T(x)] = \Pr(\chi_{S \triangle T}(x)) = \frac{1}{2}$. Orthogonality of parity functions lets us use them for transformation. Therefore, any Boolean function f can be written as: $\sum_{S \subseteq [n]} \hat{f}(S)\chi_S$ where $\hat{f}(S) =< f, \chi_S >= 1 - 2\Pr[f(x) \neq \chi_S(x)]$, so $\hat{f}(S)$ measures correlation between f and parity of x restricted to S.

Corollary 4 *of Hastad Switching Lemma*

Theorem 5 *LMN*
$\forall f \in$ *size* s $\mathrm{AC}^0_\mathrm{d}, \sum_{|S|>t} \hat{f}^2(S) \leq \epsilon$ *for* $t = O(\log \frac{s}{\epsilon})^{d-1}$

AC^0_d Learning algorithm:

1. Get estimate $\hat{f}(S)$ of $f(S), \forall S \in \zeta$ where $\zeta = \{S \subseteq [n] || S| \leq t\}$
2. If each estimate is within $\frac{\epsilon}{|\zeta|}$ of correct value, then total error for all $S \in \zeta$ is $\frac{|\zeta|\epsilon}{|\zeta|} = \epsilon$
3. Let $h = \sum_{S \in \zeta} \hat{f}(S)\chi_S$ output $sign(h)$.
4. Total error (whp) is $\leq \epsilon + \epsilon$

References

[1] S. Nisan and A. Wigderson, Hardness vs randomness, J. Comput. System Sci. 49, 149-167 (1994).

[2] S. GOLDWASSER AND S. MICALI, Probabilistie encryption, J. Comput. System Sci. 28, No. 2 (1984).

[3] A. C. YAO, Theory and applications of trapdoor functions, in "23rd FOCS, 1982," pp. 80-91.

[4] L. Valiant. A theory of the learnable. Communications of the ACM, 27, 1984.