TGINF **April 23, 2018**

Learning Algorithms from Natural Proofs

Notes: Chi-Ning Chou

This time we are going to see a recent breakthrough by Carmosino, Impagliazzo, Kabanets, and Kolokolova [CIKK16] in which they gave a quasi-polynomial time learning algorithm with membership queries for $\mathbf{AC_0}[p]$ circuits for any prime p. Their approach is highly related to the *natural proofs* [RR97] and hardness vs. randomness [NW94, IW01] framework. In this notes, we will start with some background in computational learning theory to let people get a sense what is the landscape we have right now. Next, we are going to see the high-level proof of [CIKK16]. In the end, several key proofs will be provided and we will discuss the limitations and open problems.

1 Computational learning theory

In this notes, we focus on PAC learning[1] for the uniform distribution. Let $\epsilon, \delta \in (0,1)$ and $\mathcal{C}$ be a class of boolean functions, *e.g.,* $\mathbf{AC_0}$, k-DNF, monotone functions etc., an algorithm A is a (ϵ, δ)-learning algorithm for $\mathcal{C}$ if the following holds. For any $f \in \mathcal{C}$,

- (*with uniform samples*) given samples of the form $(x, f(x))$ where x is uniformly sampled from $\{0,1\}^n$,
- (*with membership queries (MQ)*) given access to an oracle for f,

A outputs a circuit g such that with probability at least $1-\delta$,

$$\Pr_{x \in \{0,1\}^n}[f(x) \neq g(x)] \leq \epsilon.$$

The running time of the learning algorithm is measured by $T = T(n, 1/\epsilon, 1/\delta)$.

1.1 Overview of common approaches

There are two main approaches for classic learning algorithms: reducing to PTFs and Fourier analysis.

Reducing to PTFs It has been well-known that there is polynomial-time learning algorithm for constant degree PTFs. More generally, we have the following theorem.

Theorem 1 *Let $d \in \mathbb{N}$ and $\epsilon, \delta \in (0,1)$. Let $\mathcal{C}$ be the class of boolean functions such that every $f \in \mathcal{C}$ has a degree-d PTF. There exists a (ϵ, δ)-learning algorithm for $\mathcal{C}$ runs in time poly$(n^d, 1/\epsilon, \log(1/\delta))$.*

[1]Probably approximately correct learning introduced by Valiant [Val84].

Many common classes of boolean function have been know to have tight connection to PTFs. For instance, k-DNF, k-CNF, k-decision tree, and k-decision list all have degree $O(k)$ PTFs. A seminal work by Klivans and Servedio [KS04] showed that s-term DNF has degree $O(n^{1/3}\log s)$ PTF and thus having sub-exponential time learning algorithm.

Fourier analysis If the Fourier spectrum of every function in a class is concentrated in small number of coefficients, then there exists efficient learning algorithm for this class. Specifically, when the Fourier spectrum is concentrated in low-degree coefficients, then the *low-degree algorithm (LDA)* can efficiently learn the function with uniform samples. The quasi-polynomial time learning algorithm for $\mathbf{AC_0}$ by [LMN93] is a class example.

When we do not know if the Fourier spectrum is concentrated in low-degree coefficients, the Goldreich-Levin algorithm [GL89] still works, however, the resulting learning algorithm requires membership queries. Class examples are polynomial-time learning algorithm for decision tree of polynomial size by Kushileevitz and Mansour [KM93] and DNF of polynomial size by Jackson [Jac97].

1.2 Landscape of learning boolean functions

In Table 1, we summarize the current state-of-the-art learning algorithms for common classes of boolean functions to the best of our knowledge. For simplicity, we omit the definitions of these classes. Please refer to this lecture notes by Rocco Servedio for a comprehensive introduction.

Class	Running time	Reference	Approach	MQ?
PTFs[2] of degree d	$n^{O(d)}$	[BEHW89]	Linear programming	N
k-CNF, k-DNF, k-DT, k-DL	$\text{poly}(n^k)$	[Val84]	Reducing to PTFs	N
DT of size s	$\text{poly}(n^{\log s})$	[Blu92]	Reducing to PTFs	N
DT of size s	$\text{poly}(n,s)$	[KM93]	Fourier ([GL89])	Y
Monotone functions	$\text{poly}(n,s)$	[OS05]	Fourier (LDA)	N
s-term DNF	$\text{poly}(n,s)$	[Jac97]	Fourier ([GL89])	Y
s-term DNF	$2^{O(n^{1/3}\text{poly}\log(n,s))}$	[KS04]	Reducing to PTFs	N
$\mathbf{AC_0}$	$n^{\text{poly}\log n}$	[LMN93]	Fourier (LDA)	N
$\mathbf{AC_0}[p]$	$n^{\text{poly}\log n}$	[CIKK16]	Natural proofs	Y

Table 1: Current state-of-the-art learning algorithms for common classes of boolean functions.

2 Learning algorithms from natural proofs

As we have seen in the previous section, most of the learning algorithms before [CIKK16] is either reducing to PTFs or based on Fourier analysis. [CIKK16] proposes a general and naive framework of learning algorithm based on the famous *natural proofs* and hardness vs. randomness framework. In this section, we are going to sketch the high-level idea without assuming any background in natural proofs and hardness vs. randomness framework.

2.1 Natural proofs

Rudich and Razborov [RR97] introduced the notion of *natural proofs* to capture the lower bound techniques people had been used. The important message from the so called *natural proofs barrier* is that proving lower bound for complicated circuit family, *e.g.,* $\mathbf{P/poly}, \mathbf{TC_0}$, etc., is difficult under certain mild cryptographic assumption. For more details on natural proofs, please refer to another notes. Here, we will only define what is natural proof and see how one can use it to design learning algorithms.

Let $\mathcal{F}_{n,m} := \{f : \{0,1\}^n \to \{0,1\}^m\}$ to be the set of all boolean function of input length n and output length m. For simplicity, we let $\mathcal{F}_n = \mathcal{F}_{n,1}$. We denote a family of function by using bold face character such as $f = \{f_n\}_{n\in\mathbb{N}}$ where $f_n \in \mathcal{F}_n$ for each $n \in \mathbb{N}$. A property $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ is a family of collection of functions where $\mathcal{P}_n \subseteq \mathcal{F}_n$ for each $n \in \mathbb{N}$. Semantically, $\mathcal{P}$ contains the functions that satisfy the *property*.

It's not difficult to see that to prove that a typical circuit class $\mathcal{C}$ does not contain a function family f is equivalent to find a property $\mathcal{P}$ such that

- $f \in \mathcal{P}$, and
- for any $g \in \mathcal{C}$, $g \notin \mathcal{P}$.

Now, we define the notion of natural proofs without providing motivations. Please refer to the other notes for intuition if interested.

Definition 2 *Let Γ and Λ be two complexity classes and $\delta(n)$ be some function in n, we call a property $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ is Γ-natural and $\mathcal{C}$-useful with density $\delta(n)$ if the following conditions hold.*

- ***(constructive)** there exists a test T computable in Γ such that for any $f = \{f_n\}_{n\in\mathbb{N}}$, $T(f_n) = 1$ for all $n \in \mathbb{N}$ iff $f \in \mathcal{P}$. Note that the parameter used in Γ is the size of the truth table of f_n, i.e., 2^n.*
- ***(largeness)** $\mathbb{P}_{f_n \leftarrow \mathcal{F}_n}[f_n \in \mathcal{P}_n] \geq \delta(n)$. Equivalently, $|\mathcal{P}_n| \geq \delta(n) \cdot |\mathcal{F}_n|$.*
- ***(usefulness)** for any $f \in \mathcal{C} \cap \mathcal{F}_n$, $f \notin \mathcal{P}_n$.*

We say a proof is a natural proof if the property $\mathcal{P}$ involved in the proof can be efficiently constructed and is non-negligibly large.

Definition 3 (natural proof) *We say $\mathcal{P}$ is a Γ-natural proof for $f \notin \mathcal{C}$ for some function family f if $f \in \mathcal{P}$ and $\mathcal{P}$ is a Γ-natural against $\mathcal{C}$ with density $1/poly(n)$.*

Lemma 4 *There exist $\mathbf{P}$-natural proofs against $\mathbf{AC_0}$ and $\mathbf{AC_0}[p]$ for any prime p.*

Before we state the main theorem of [RR97], we need one last definition.

Definition 5 (pseudorandom generator (PRG)) *A function $G_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is a PRG with hardness $s(n)$, stretch $\ell(n)$, and error $\epsilon(n)$ if G is computable in time poly($\ell(n)$) and for any circuit C of size at most $s(n)$,*

$$|\mathbb{P}[C(G_n(U_n)) = 1] - \mathbb{P}[C(U_{\ell(n)}) = 1]| \geq \epsilon(n),$$

where U_n is the uniform distribution on $\{0,1\}^n$.

Theorem 6 **([RR97])** *Suppose there exists a* $\mathbf{P}/\mathbf{poly}$*-natural proof against* $\mathbf{P}/\mathbf{poly}$*, then there's no PRG against circuit of size* 2^{n^ϵ} *for any* $\epsilon > 0$.

As a remark, there are different forms of Theorem 6 in which other circuit families are considered. The high-level idea of the proof of Theorem 6 is one-line: natural proof can serve as the distinguisher for the PRG.

2.2 Hardness vs. randomness

Nisan and Wigderson [NW94] introduced the ground-breaking idea of hardness vs. randomness framework which has great applications such as derandomization [IW01, IKW02, KI04], SAT algorithm [Wil13], and learning [FK09, KKO13]. The idea is to reduce a hard function to a *pseudo-random generator (PRG)* and the high-level idea is the following. First, let f be a hard function (on average) in the sense that for every $g \in \mathcal{C}$, for any n large enough, $\Pr[f(U_n) \neq g(U_n)] \geq 1/2 - 1/n$ where U_n is the uniform distribution over $\{0,1\}^n$. Here, think of $\mathcal{C}$ as a class of *easy* functions such as $\mathbf{AC_0}$. Second[3], consider an efficient transformation $NW : \mathcal{F}_n \to \mathcal{F}_{n,m}$ where $m > n$ such that (i) $NW(f)(x)$ can be computed in $\text{poly}(m, \text{size}(f))$ for any $x \in \{0,1\}^m$ and (ii) for any $g \in \mathcal{C}$, $|\Pr[g(NW(f)(U_n)) = 1] - \Pr[g(U_m)]| \leq 1/n$. The transformation NW is also known as the *Nisan-Wigderson generator*.

For now, let us temporarily forget about how to construct such transformation NW and focus on how to prove such statement. That is, how to prove a transformation can make a hard function into a PRG? The proof is by contradiction. Namely, suppose $NW(f)$ is *not* a PRG and thus there exists a *distinguisher* $g \in \mathcal{C}$ such that $|\Pr[g(NW(f)(U_n)) = 1] - \Pr[g(U_m)]| > \epsilon$ for some constant $\epsilon > 0$. The goal is to show that there exists $h \in \mathcal{C}$ such that $f \equiv h$ with the help of g. Note that here we implicitly hide the transformation of the parameters for the ease of reading. For interested readers, Chapter 7 of this book [V+12] is a very good resource. See Figure 1.

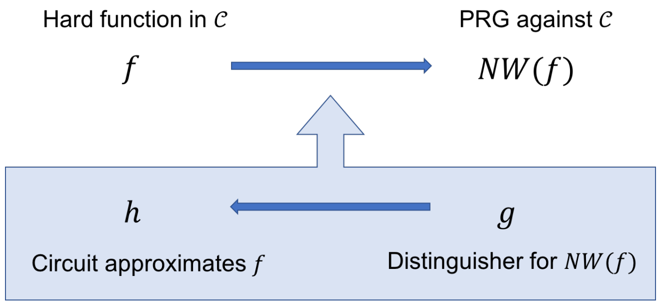
Hard function in $\mathcal{C}$
f
PRG against $\mathcal{C}$
$NW(f)$
h
Circuit approximates f
g
Distinguisher for $NW(f)$

Figure 1: Proof strategy in [NW94] for hardness vs. randomness connection.

To make the proof work, one can see that the construction of the transformation NW should be (i) *complicated* enough so that $NW(f)$ preserves the hardness of f and (ii) *simple* enough so that one can use a distinguisher for $NW(f)$ to build a circuit for f. The *Nisan-Wigderson designs* achieves this goal and even more has very flexible parameters. We will postpone the introduction of Nisan-Wigderson designs to later section and the take home message for now is that the transformation should be both complicated and simple at the same time.

[3]Note that this is exactly the definition of pseudorandom generator (PRG).

2.3 Natural proofs + Nisan-Wigderson generator?

So far we have seen natural proofs, which can be thought of as a distinguisher for PRG, and Nisan-Wigderson generator, which transforms an average-hard function to a PRG, what can we do? Especially, what do they relate to learning algorithm for $\mathbf{AC_0}[p]$?

A very rough idea is th following. Feed the function f we want to learn into the Nisan-Wigderson generator. Next, use the natural proof for $\mathbf{AC_0}[p]$ as a distinguisher and then utilize the proof of hardness vs. randomness connection to get a circuit that approximates f. This idea is very appealing though many parts are questionable.

- The input of Nisan-Wigderson generator should be an average-hard function, but here $f \in \mathbf{AC_0}[p]$.
- During the reconstruction from the distinguisher to the circuit, what kind of information about f is required?
- Is the output circuit sill in $\mathbf{AC_0}[p]$?

To answer these questions, we need to dig into the literature a little bit more. As a quick answer, the first question can be resolved by the so called *hardness amplification* [Yao82] using XOR and direct product. Though for $p \neq 2$, we cannot use the XOR construction and a similar construction, which applies for all prime p,was proposed in [CIKK16]. The second question was first answered by [IW01] in which Impagliazzo and Wigderson showed that uniform algorithm with *black-box queries* to f is sufficient. Namely, this is basically a learning algorithm with membership queries. The last question was answered in affirmative by [CIKK16]. They showed that the Nisan-Wigderson designs can be implemented in $\mathbf{AC_0}[p]$ for any prime p and thus the output of the reconstruction is also in $\mathbf{AC_0}[p]$. See Figure 2.

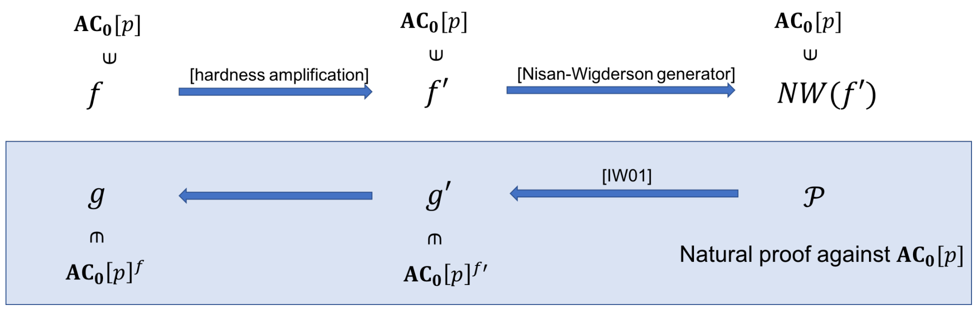

Figure 2: Learning algorithm for $\mathbf{AC_0}[p]$ in [CIKK16].

As we discussed above, the two main contributions of [CIKK16] are (i) showing that Nisan-Wigderson designs can be implemented in $\mathbf{AC_0}[p]$ and (ii) showing that hardness amplification can be done in $\mathbf{AC_0}[p]$ where p is a prime different from 2. Due to the limitation of preparation, here we omit the details and interested reader can find the corresponding part in Theorem 2.12 and Section 4.2 of [CIKK16] (ECCC version) respectively.

3 Limitations and open problems

One natural question after seeing the learning algorithm for $\mathbf{AC_0}[p]$ above could be: can similar approach work for $\mathbf{AC_0}$? Interestingly, the answer is *no*. In chapter 6 of [CIKK16] (ECCC version), they provided proof of why Nisan-Wigderson designs cannot be implemented in $\mathbf{AC_0}$. The high-level reason is that $\mathbf{AC_0}$ circuit has low average sensitivity while the characteristic function of Nisan-Wigderson designs have high average sensitivity.

Finally, we conclude with some open problems.

- Can one make the learning algorithm deterministic[4]?
- Can one remove the requirement of membership queries in the learning algorithm?
- Is there a way to get nontrivial SAT algorithms from natural proofs[5]?

References

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[Blu92] Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters*, 42(4):183–185, 1992.

[CIKK16] Marco L Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 50. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[FK09] Lance Fortnow and Adam R Klivans. Efficient learning algorithms yield circuit lower bounds. *Journal of Computer and System Sciences*, 75(1):27–36, 2009.

[GL89] Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32. ACM, 1989.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001.

[Jac97] Jeffrey C Jackson. An efficient membership-query algorithm for learning dnf with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):414–440, 1997.

[4] There involves randomness in the hardness amplification step.

[5] [Wil13] is a conditional result. Can one come up with an unconditional result?

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[KKO13] Adam Klivans, Pravesh Kothari, and Igor C Oliveira. Constructing hard functions using learning algorithms. In *Computational Complexity (CCC), 2013 IEEE Conference on*, pages 86–97. IEEE, 2013.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[KS04] Adam R Klivans and Rocco A Servedio. Learning dnf in time 2o (n1/3). *Journal of Computer and System Sciences*, 68(2):303–318, 2004.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *Journal of the ACM (JACM)*, 40(3):607–620, 1993.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of computer and System Sciences*, 49(2):149–167, 1994.

[OS05] Ryan ODonnell and R Servedio. Learning monotone functions from random examples in polynomial time, 2005.

[RR97] Alexander A Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

[V$^+$12] Salil P Vadhan et al. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

[Val84] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

[Yao82] Andrew C Yao. Theory and application of trapdoor functions. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 80–91. IEEE, 1982.