CSCI699: Theory of Machine Learning Fall 2021

Lecture 20

Instructor: Vatsal Sharan *Scribes: Ali Omrani*

1 Computational-Statistical Tradeoffs

In the first part of the class, we studied the statistical complexity of learning, i.e. how much data do we need in order to learn? After that, we switched to computational aspects, and asked when learning is possible in polynomial time. A natural, and very active line of research, is to understand if and when there are tradeoffs between computational and statistical resources. When does computational efficiency come at the cost of statistical power?

One of the most studied problem in this direction is the planted clique problem.

Definition (Planted Clique Problem)**.** *Given a graph generated from one of the following 2 distributions, decide which distribution generated the graph*

1. *$G(n, 1/2)$ denoting Erdös-Renyi graph with n vertices and probablity of $\frac{1}{2}$.*
2. *Generate an instance of $G(n, 1/2)$ and plant a clique on k randomly chosen vertices of the graph.*

The notion of statistical power of an algorithm here is the minimum clique which can be detected by the algorithm.

Question: What is the smallest k at which these two distributions are information-theoretically distinguishable?

The following Lemma helps us understand the information theoretic limits here, i.e. the best statistical power among all possible algorithms.

Lemma 1. *An Erdös-Renyi random graph $G(n, 1/2)$ does not have a clique of $k \geq 3 \log n$ w.h.p.*

Using this Lemma, we have an inefficient brute force search algorithm for $k \geq 3 \log n$:

Algorithm 1 Brute-Force Search

```
Search every subset of $k$ vertices
if $\exists$ a clique on any subset then
    return (graph comes from a planted clique mode)
else
    return (graph comes from $G(n, 1/2)$)
end if
```

The running time of the algorithm is $k^2\binom{n}{k} = \Omega(n^{\log n})$, hence it does not run in polynomial time. But it does prove that it is information-theoretically possible to detect the planted clique for $k \geq 3\log n$.

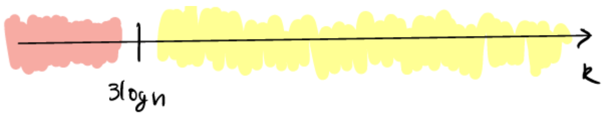

Figure 1: For $k << \log n$ it is information-theoretically impossible to distinguish the two cases but if $k \geq 3\log n$, information-theoretically it is possible to distinguish.

Question: When can we do this efficiently?

There is in fact a simple, efficient algorithm when $k >> \sqrt{n\log n}$, based on the following simple Lemma.

Lemma 2. *The number of edges in an Erdös-Renyi random graph $G(n, 1/2)$ lies in* $\left[\frac{n(n-1)}{4} - 100\sqrt{\log n}\ ,\ \frac{n(n-1)}{4} + 100\sqrt{\log n}\right]$ *w.h.p.*

By adding a clique on k vertices we will add $\approx \binom{k}{2}\frac{1}{2} = \frac{k(k-1)}{4}$ edges (repeat previous lemma for rigorous detail.)

$k = \sqrt{cn\log n}$ for some large c, we will add $\approx \frac{cn\log n}{4}$ edges.

- W.h.p., for $k = \sqrt{cn\log n}$, $G(n, 1/2)$ graph with planted clique has at least $\frac{n(n-1)}{4} + \frac{1000n\log n}{4}$ edges for large enough c (in the equation $c = 1000$ for large value of c.)
- For $k >> \sqrt{n\log n}$ we can efficiently detect the clique w.h.p. by counting the number of edges in the graph.
- For $k >> \sqrt{n}$ there is an efficient algorithm based on eigenvalue decomposition of the adjacency matrix of the graph.

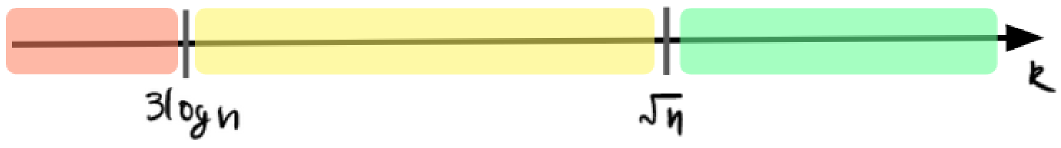

Figure 2: It is believed to be impossible to efficiently detect the presence of a planted clique in the yellow region ($3\log n \leq k \leq \sqrt{n}$). For the green region ($k >> \sqrt{n}$) efficient algorithms do exist. The planted clique conjecture makes this precise.

Definition (Planted Clique Conjecture)**.** *There is no efficient algorithm to detect a planted clique of size $k = o(\sqrt{n})$ in a $G(n, 1/2)$ graph.*

This conjecture is still open, but is known to be true for restricted class of algorithms such as

- A version of SQ algorithms.
- generalization of Semi-Definite Programs (SDPs)
- Markov Chain Monte-Carlo (MCMC) methods.
- ... and many other algorithmic classes.

2 Memory-Sample Tradeoffs

We'll now turn to a different kind of statistical-computational tradeoff. For most of the class, we've used the runtime of the algorithm as the measure of its computational efficiency. Perhaps the next most important computational resource is memory usage. In light of the computational-statistical tradeoffs for running time as the computational resource, we can ask the following natural question: What is the tradeoff between available memory and number of samples needed for learning?

Memory-Sample Tradeoffs for Parity Learning

We'll understand this for the parity problem. The setting is as follows. The data comes in streaming fashion. We get datapoints one at a time, only get a single pass over your data stream. The goal is to learn the parity function:

$$\begin{aligned} \mathcal{X}^d &= \{0,1\}^d \\ \mathcal{Y} &= \{0,1\} \\ C &= \{w(x) =< w, x > \mod 2 : w \in \{0,1\}^d\}. \end{aligned}$$

There is some unkonwn $w^* \in \{0,1\}^d$ which we want to find.

At every timestep we get a labelled example. We can store the example in memory if we like, or do some computation based on the example and store the result of the computation, but we don't get to see the example again i.e. we only get one pass over the datastream:

at $t = 1$

- Get $x_1 \sim Unif(\{0,1\}^d)$
- Get $b_1 =< x_1, w^* > mod\ 2$

at $t = 2$

- Get $x_2 \sim Unif(\{0,1\}^d)$

- Get $b_2 =< x_2, w^* > mod\ 2$

.
.
.

Let us try to understand what might be possible here. The unknown vector x^* is d-dimensional, so $\Omega(d)$ memory is necessary for even storing the solution, and hence for solving the problem. Also, since we're looking at a linear system (over GF(2)) in d-dimensions, $\Omega(d)$ examples are also necessary for learning x^*. So what if we try to solve the problem with only $O(d)$ memory, could we still get the optimal $O(d)$ sample complexity? It does not seem easy, because with $O(d)$ memory we can only store a constant number of examples at a time, and maybe in that case we would need to see many more than $O(d)$ examples to solve the problem. So we ask,

Question: For the parity problem, what is the tradeoff between the available memory and number of samples needed for learning?

We first consider two very natural algorithms for the problem.

Algorithm 2

Store $n = O(d)$ examples in memory and solve the linear system.

$$\begin{pmatrix} \text{---}x_1\text{---} \\ \text{---}x_2\text{---} \\ . \\ . \\ . \\ \text{---}x_n\text{---} \end{pmatrix} \begin{pmatrix} \\ w^* \\ \\ \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_n \end{pmatrix} \text{mod } 2$$

Since w^* is d-dimensional with $n >> 100d$ examples, the system is full-rank w.h.p.

Samples $= n = O(d)$
Memory $= nd$ bits $= \Omega(d^2)$

So the algorithm is great in terms of its sample complexity, but uses a lot of memory. The following brute force search algorithm achieves the other extreme.

Algorithm 3 Brute-Force Search

```
for every $w \in \{0,1\}^d$ do
    if $w$ is consistent over the next $o(d)$ examples we receive then
        return w
    end if
end for
```

Memory $= O(d)$
Samples $= d2^d = \Omega(2^d)$

Question: What else is possible?

In a surprising result, Ran Raz showed that the above two algorithms are essentially all that is possible for the problem, and therefore we have almost a complete understanding of the algorithmic landspace of the problem.

Theorem 3 (Raz '17)**.** *Any algorithm for solving the above parity problem either requires* $\Omega(d^2)$ *memory, or at least* $2^{\Omega(d)}$ *samples.*

Interestingly, all our previous computational lower bounds were based on assumptions such as RP$\neq$ NP, but the above theorem is unconditional. It seems that understanding memory-sample tradeoffs is much more information-theoretic, and we can indeed show stark, unconditional lower bounds.

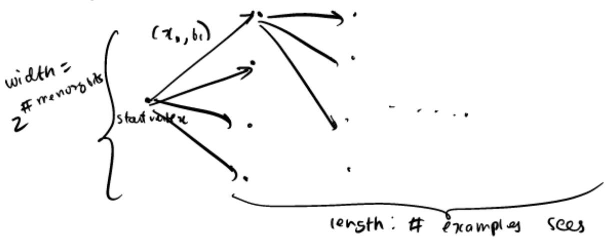

Figure 3: The proof is based on the idea of analyzing a branching program for the problem.

Subsequent work has extended the result to a much larger class of problems. For instance, the following result shows a memory lower bound based on the SQ dimension of the learning task.

Theorem 4 (Garg, Raz, Tal '18)**.** *Consider a hypothesis class* $\mathcal{H}$ *with SQ-dim*$(\mathcal{H}) = s$*. Then any algorithm for learning* $\mathcal{H}$ *requires* $\Omega(\log^2 s)$ *memory, or at least* $s^{\Omega(1)}(poly(s))$ *samples.*

References

[1] Raz, Ran. "A time-space lower bound for a large class of learning problems." 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2017.

[2] Garg, Sumegha, Ran Raz, and Avishay Tal. "Extractor-based time-space tradeoffs for learning." Manuscript. July (2017).

[3] Raz, Ran. "Fast learning requires good memory: A time-space lower bound for parity learning." Journal of the ACM (JACM) 66.1 (2018): 1-18.