Learning, Space, and Cryptography

Andrew Gu* Franklyn Wang† Eric Zhang‡

May 6, 2021

Abstract

This is our final project for CS 221 (Computational Complexity), an expository report on [Raz17], which gives samples-space lower bounds (and *a fortiori* time-space lower bounds) for a broad class of learning problems.

Specifically, it is shown that if we consider the learning problem $M : \Theta \times \mathcal{X} \to \{-1, 1\}$ and a learner is given a stream of samples (x_i, y_i) where $y_i = M_\theta(x_i)$, and the largest singular value of M^T is $|\mathcal{X}|^{1/2} \cdot |\Theta|^{1/2-\epsilon}$, then any learning algorithm for the corresponding learning problem requires either a memory of size at least $O((\epsilon n)^2)$ or at least $2^{\Omega(\epsilon n)}$ samples, where $n = \log_2 |\Theta|$.

We elucidate the seemingly strange singular-value condition, showing that it has connections to the difficulty of ascertaining y from x and given a probability distribution on θ. We then strive to simplify the arguments in the paper in light of these insights.

1 Motivation

Machine learning has risen to incredible prominence in recent years, attaining stunning results on tasks as varied as protein folding [SEJ+20], playing games [SHM+16] and even music generation from audio waveforms [DJP+20]. Its success begs the question:

What are the fundamental limits of machine learning?

There are several attempts to attack this question, but one particularly interesting route is finding lower bounds on space and time complexity needed to solve a particular problem. With lower bounds on say, deep learning problems, we can establish problems that can for sure not be solved with deep learning, which would allow us to further map out the landscape of which questions can be solved.

However, the history of finding lower bounds in complexity theory has been largely disappointing. In fact, we still do not know if there exist problems in PSPACE which are not in P, as determining whether P = PSPACE is still a longstanding open problem.

We focus on space lower bounds in particular, because a fairly distinctive aspect of deep learning algorithms is that they use a relatively small amount of memory. Consider, say, stochastic gradient descent (for simplicity, let the batch size be one). Then, given a stream of samples, stochastic gradient descent uses space equal to D, where D is the number of parameters in the deep model. The result proven by Raz, then, is surprising in that it shows that to learn precisely a D-parameter model we may need D^2 space. While this result somewhat resembles previous work

*MIT, email: agu1@mit.edu
†Harvard, email: franklyn_wang@college.harvard.edu
‡Harvard, email: ekzhang@college.harvard.edu

in this area like [SSSS17], which studies gradient descent, Raz's result is unique in that it makes no assumptions on the algorithm used, instead showing that the space tradeoff is intrinsic to a wide swath of problems and cannot be avoided.

2 Preliminaries

In this section, we describe the setup of the problem and introduce notations. We have a learning problem over parameters Θ and inputs $\mathcal{X}$ so that our concept and hypothesis class (note that we are operating in the exact learning problem) is $f_\theta(x)$.

2.1 Model: Streaming and Branching Programs

In most complexity theory problems, the input is not considered to be part of the space usage. Yet in this question, not including the input as part of the space would be far too generous – observe that, for example, you could solve parities with almost no effort if there were no time constraints, as you could just try every single solution. This motivates a streaming model, where at each step you receive one input, and you are forced to process it then and there (possibly by storing it).

If we were allowed to hold all the input in memory at the same time, this problem would be much easier. However, in this problem we consider a streaming model where the storing input would count against our space usage. This somewhat restrictive model is used because in practice, many algorithms (such as all first-order methods) do satisfy this condition with limited space, even though the input is not counted.

The main result of the paper is stated in terms of *branching programs*, which are a natural model of computation to use in this analysis because they accurately capture the most general notion of space complexity. Branching programs are essentially machines that enable infinite complexity of computation, but have a limited amount of space as determined by the logarithm of the size of the branching programs. One way to think about branching programs is as directed acyclic graphs, where each edge corresponds to learning some additional information.

In the branching program model of the learning problem, the branching program B has m layers $L_1, L_2, \ldots, L_m$, each consisting of 2^k vertices, where k is the space allocated to the program. Edges go from layer L_{i-1} to layer L_i and are labeled with (x_i, y_i), corresponding to the program learning $M_\theta(x_i) = y_i$. Note that this model places no limits on the computational power of the learning problem, and is essentially purely geometric.

2.2 Main Result

We can now represent a learning problem as a $|\Theta| \times |\mathcal{X}|$ matrix M where $M_{ij} = f_{\theta_i}(x_j)$. Now, suppose that $\sigma_{\max}(M^\top) \le |\mathcal{X}|^{1/2} \cdot |\Theta|^{1/2-\epsilon}$ (noting that in general, $\sigma_{\max}(M^\top) \le |\mathcal{X}|^{1/2} \cdot |\Theta|^{1/2}$). Then, the main theorem of [Raz17] states that

Theorem 2.1. *Let $\Theta, \mathcal{X}$ be two finite sets. Let $n = \log_2 |\Theta|$. Let $M : \Theta \times \mathcal{X} \to \{-1, 1\}$ be a matrix, such that $\|M^\top\|_2 \le 2^{\gamma n}$ (where $\gamma < 1$ is not necessarily a constant, but $(1-\gamma) \cdot n \to \infty$ (when $n \to \infty$)).*

For any constant $c' < \frac{1}{3}$, there exists a (sufficiently small) constant $\epsilon' > 0$, such that the following holds: Let $c = c' \cdot (1-\gamma)^2$, and let $\epsilon = \epsilon' \cdot (1-\gamma)$. Let B be a branching program of length at most $2^{\epsilon n}$ and width at most 2^{cn^2} for the learning problem that corresponds to the matrix M. Then, the success probability of B is at most $O(2^{-\epsilon n})$.

3 Key Parts of the Analysis

3.1 Motivating the Spectral Norm Condition

At first, the spectral norm condition may seem a little strange. Why does $\|M^\top\|$ being low correspond to the problem being hard? Here it is useful to consider the role of the matrix $M^\top$. Suppose that we had a prior on Θ given by the vector $v \geq 0$. Then, conditioned on this, what's the vector

$$\begin{bmatrix}\mathbb{E}[f_\theta(x_1)] & \mathbb{E}[f_\theta(x_2)] & \dots & \mathbb{E}[f_\theta(x_n)]\end{bmatrix}^\top?$$

It's equal to $M^\top v$, whose norm is bounded by $\|M^\top\|_2\|v\|_2$ Thus, one way to interpret the condition is that a small amount of information on θ (symbolized by a small value of $\|v\|_2$) corresponds to a small average-case knowledge of x_i (symbolized by a small value of $\|M^\top v\|_2$). This step is crucial to the proof, as it provides a characterization for the hardness of the learning problem. Intuitively, if $\|M^\top\|_2$ is low, its very hard to know the value of $f_\theta(x)$ if we don't know the value of x. The parity functions are a particularly striking example of this. Even if you've narrowed down the set of θ's to two possibilities, for example, we still do not know the value of $f_\theta(x)$ on 50% of all the inputs.

3.2 A Walk on the Branching Program

The branching program is a layered graph whose leaves correspond to output values of θ. For a vertex v of the branching program, we let $\mathbb{P}_{\theta|v}$ denote the probability distribution on Θ corresponding to the distribution of the outputs conditioned on the branching program having reached v. One way to think about this is a posterior. Similarly, we define $\mathbb{P}_{\theta|e}$ as the probability distribution of the output condition on having crossed edge e.

We define a *truncated path* on the branching program which is the same as the regular path of the branching program, but may stop before reaching a leaf. Specifically, the path ends on a vertex v if one of these conditions holds:

- The vertex v is a *significant vertex*, meaning

  $$\|\mathbb{P}_{\theta|v}\|_2 > 2^{\delta n} \cdot 2^{-n}$$

  for some specific constant δ.

- The correct value θ^* is a significant value, meaning

  $$\mathbb{P}_{\theta|v}(\theta^*) > 2^{2(\delta+\epsilon)n} \cdot 2^{-n}.$$

  (The set of θ' satisfying $\mathbb{P}_{\theta|v}(\theta') > 2^{2(\delta+\epsilon)n} \cdot 2^{-n}$ is denoted $\mathrm{Sig}(v)$.)

- The next input (x_{i+1}, y_{i+1}) is a bad edge, meaning

  $$\|(M^\top \cdot \mathbb{P}_{\theta|v})(x_{i+1})\| \geq 2^{(\delta+\gamma+\epsilon)n} \cdot 2^{-n}.$$

The main idea is that these truncations correspond to situations where a large amount of information is known about θ, but they also occur rarely, implying that the branching program cannot effectively learn θ. The central claim shows that the probability of reaching a significant vertex (the first condition for truncation) is small.

Theorem 3.1. *The probability that $\mathcal{T}$ reaches a significant vertex is $O(2^{-en})$.*

Theorem 3.1 implies that the other conditions for truncation (x being a significant value, or the next edge being a bad edge) also occur with small probability. To prove Theorem 3.1, we fix a significant vertex s and show that the probability of reaching s is small. Then taking a union bound over significant vertices proves the result.

3.3 Measuring Progress of the Branching Program

For each layer i of the branching program, let L_i be the set of vertices on layer i and Γ_i be the set of edges from layer $i-1$ to layer i. We then define

$$\mathcal{Z}_i = \sum_{v \in L_i} \Pr(v) \cdot \langle \mathbb{P}_{\theta|v}, \mathbb{P}_{\theta|s} \rangle^n,$$
$$\mathcal{Z}'_i = \sum_{e \in \Gamma_i} \Pr(v) \cdot \langle \mathbb{P}_{\theta|e}, \mathbb{P}_{\theta|s} \rangle^n.$$

Intuitively, these are measures of progress toward reaching a distribution similar to $\mathbb{P}_{\theta|s}$. When $\mathbb{P}_{\theta|v}$ is very close, $\mathbb{P}_{\theta|s}$ it will mean that this vertex has basically "figured it out."

Lemma 3.2.

$$\mathcal{Z}_i \le \mathcal{Z}'_i.$$

Proof. Omitted. □

The goal is to show that $\mathcal{Z}_i$ grows slowly with i. The key claim is the following, which expresses a bound on the progress made by the branching program.

Lemma 3.3. *For every vertex v of B with $\Pr(v) > 0$,*

$$\sum_{e \in \Gamma_{\text{out}}(v)} \frac{\Pr(e)}{\Pr(v)} \cdot \langle \mathbb{P}_{\theta|e}, \mathbb{P}_{\theta|s} \rangle^{\beta n} \le \langle \mathbb{P}_{\theta|v}, \mathbb{P}_{\theta|s} \rangle^{\beta n} \cdot (1 + 2^{-1.9\epsilon n}) + (2^{-2n+2})^{\beta n}.$$

This lemma is the hardest step of the proof and we will not prove it in full detail. We first show how this leads to bounds on $\mathcal{Z}_i$.

Lemma 3.4. *For every $i \in \{1, \dots, m\}$,*

$$\mathcal{Z}_i \le \mathcal{Z}_{i-1}(1 + 2^{-1.9\epsilon n}) + (2^{-2n+2})^{\beta n}.$$

Proof. Combining Lemmas 3.2 and 3.3, we have

$$\begin{aligned}
\mathcal{Z}_i &\le \mathcal{Z}'_i \\
&= \sum_{e \in \Gamma_i} \Pr(e) \langle \mathbb{P}_{\theta|e}, \mathbb{P}_{\theta|s} \rangle^{\beta n} \\
&= \sum_{v \in L_{i-1}} \Pr(v) \cdot \sum_{e \in \Gamma_{out}(v)} \frac{\Pr(e)}{\Pr(v)} \langle \mathbb{P}_{\theta|e}, \mathbb{P}_{\theta|s} \rangle^{\beta n} \\
&= \sum_{v \in L_{i-1}} \Pr(v) \cdot \left(\langle \mathbb{P}_{x|v}, \mathbb{P}_{x|s} \rangle^{\beta n} \cdot (1 + 2^{-1.9\epsilon n}) + (2^{-2n+2})^{\beta n} \right) \\
&= \mathcal{Z}_{i-1} \cdot (1 + 2^{-1.9\epsilon n}) + \sum_{v \in L_{i-1}} \Pr(v) \cdot (2^{-2n+2})^{\beta n} \\
&= \mathcal{Z}_{i-1} \cdot (1 + 2^{-1.9\epsilon n}) + (2^{-2n+2})^{\beta n}.
\end{aligned}$$

□

We can now give an upper bound on $\mathcal{Z}_i$.

Lemma 3.5. *For every $i \in \{1, \ldots, m\}$,*

$$\mathcal{Z}_i \leq 2^{2(\beta+\epsilon)n - 2\beta n^2}.$$

Proof. We use lemma 3.4. By expanding and inducting, we have

$$\mathcal{Z}_i \leq \mathcal{Z}_0 x^i + x^{i-1}y + x^{i-2}y + \cdots + xy + y$$

where $x = 1 + 2^{-1.9\epsilon n}$ and $y = 2^{-(2n-2)\beta n}$. Note that $\mathcal{Z}_0 = (2^{-2n})^{\beta n} < y$ and $x > 1$. The right hand side is increasing with i and at $i = m$, we have the upper bound

$$\begin{aligned}
\mathcal{Z}_i &\leq mx^m y \\
&= 2^{\epsilon n}(1 + 2^{-1.9\epsilon n})^m (2^{-(2n-2)\beta n}) \\
&\leq 2^{\epsilon n} \cdot 2 \cdot 2^{-2\beta n^2} \cdot 2^{2\beta n} \\
&\leq 2^{2(\beta+\epsilon)n - 2\beta n^2}.
\end{aligned}$$

using $m = 2^{\epsilon n}$. □

We now present one step in the proof of Lemma 3.3 to show where the singular value condition comes into play.

Lemma 3.6. *Define a function $f : \Theta \to \mathbb{R}^+$ by*

$$f(\theta') = \begin{cases} 0 & \text{if } \theta' \in \mathrm{Sig}(v) \\ \mathbb{P}_{\theta|v}(\theta')\mathbb{P}_{\theta|s}(\theta') & \text{if } \theta' \notin \mathrm{Sig}(v) \end{cases}$$

and let

$$F = \sum_{\theta \in \Theta} f(\theta).$$

Then for every edge $e \in \Gamma_{out}(v)$, there exists a normalization factor

$$c_e > \frac{1}{2} - 2 \cdot 2^{-2\epsilon n}$$

such that

$$\mathbb{P}_{\theta|e}(\theta') \cdot \mathbb{P}_{\theta|s}(\theta') = \begin{cases} 0 & \text{if } M_\theta(a) \neq b \\ f(\theta') \cdot c_e^{-1} & \text{if } M_\theta(a) = b \end{cases}$$

and furthermore

$$\langle \mathbb{P}_{\theta|e}, \mathbb{P}_{\theta|s} \rangle < 2^{-n} \cdot (F + |(M^\top \cdot f)(x)|) \cdot (1 + 2^{-2\epsilon n + 3}).$$

Proof. For brevity, we omit the part regarding the normalization factor. We have:

$$\begin{aligned}
\langle \mathbb{P}_{\theta|e}, \mathbb{P}_{\theta|s} \rangle &= \mathbb{E}[\mathbb{P}_{\theta|e}(\theta) \cdot \mathbb{P}_{\theta|s}(\theta)] \\
&= c_e^{-1} \cdot 2^{-n} \sum_{\theta | M_\theta(x) = y} f(\theta) \\
&= c_e^{-1} \cdot 2^{-n} \left(\frac{F + b \cdot (M^\top \cdot f)(x)}{2} \right) \\
&\leq (2c_e)^{-1} \cdot 2^{-n} \cdot \Big(F + |(M^\top \cdot f)(x)| \Big) \\
&\leq 2^{-n} \cdot \Big(F + |(M^\top \cdot f)(x)| \Big) \cdot (1 + 2^{-2\epsilon n + 3}).
\end{aligned}$$

□

With this lemma in mind, the term $M^\top \cdot f$ allows us to use the singular value condition to further work on computing upper bounds. The proof of Lemma 3.3 proceeds by splitting into cases $F \leq 2^{-n}$ and $F \geq 2^{-n}$, extracting an $\|M^\top\|_2^2$ term in the latter case.

4 Applications to Cryptography

Space-lower bounds are a powerful tool in our arsenal of theoretical techniques. When we can prove that a task requires a significant amount of space, there are countless applications. For example, a topic even hotter than machine learning at the moment is the *blockchain* algorithm for securing a distributed ledger. Cryptocurrencies are often criticized for their heavy environmental impacts (among other things). This stems from the proof-of-work scheme that is required to prove the existence of a scarce resource, which in the case of Bitcoin is computation time. This takes a toll on the environment due to soaring energy bills and electricity costs. However, proof-of-space algorithms may be able to capture the same Byzantine fault tolerance benefits, without nearly as much energy expenditure.

Perhaps most notably, FileCoin is a recently developed cryptocurrency which aspires to be a secure segment of IPFS, the largest decentralized filesystem in the world. Receiving FileCoin is contingent on providing space to others, which itself depends on proving that you have the space through proofs of space. It's possible that a ParityCoin could be developed, based on this result or similar results of this nature.

References

[DJP+20] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[Raz17] Ran Raz. A time-space lower bound for a large class of learning problems. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 732–742. IEEE, 2017.

[SEJ+20] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[SSSS17] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, pages 3067–3075. PMLR, 2017.