No Data, No Problem: Improving Linear Regression by Increasing Computer Memory

Vafa Behnam Roudsari

February 2020

1 Introduction

Paper 1: Fast Learning Requires Good Memory: A Time-Space Lower Bound for Parity Learning (Ran Raz) [1]

Paper 2: Memory-Sample Tradeoffs for Linear Regression with Small Error (Sidford, A; Valiant, G; Sharan, V) [2]

In 2011, Michael Jordan polled a number of distinguished senior statisticians about the major open problems in Bayesian Statistics, and concluded that the second most important open problem was

> ...a more thorough integration of computational science and statistical science, noting that the set of inferences that one can reach in any given situation are jointly a function of the model, the prior, the data and the computational resources, and wishing for more explicit management of the tradeoffs among these quantities.

Moreover,

> respondents ... expressed a feeling that a large amount of data does not necessarily imply a large amount of computation; rather, that somehow the inferential strength present in large data should transfer to the algorithm and make it possible to make do with fewer computational steps to achieve a satisfactory (approximate) inferential solution.

The two papers I will cover here affirm these speculations: that indeed, there is a tradeoff between these the amount of memory an algorithm can access, and the number of samples required in order to reach a given level of accuracy in the context of parity learning and linear regression.

The conclusion of this question can perhaps help clarify exactly how much of a benefit is conferred to less memory-intensive first-order gradient algorithms, like gradient descent, than second-order techniques, like Newton-Raphson, which require $O(n^2)$ memory bits.

But also, it may suggest techniques for identifying such tradeoffs in more computationally intensive algorithms, such as kernel density estimation or deep learning. It would be fascinating to know for example that, if we have a small sample, we would not be able to accurately summarize a distribution unless our neural network has at least $O(n)$ memory bits.

The first of the two papers is Ran Raz's work that can be thought of as regression in a field modulo 2. He introduced branching programs as a way to model all possible algorithms, and this is the common ingredient in both papers. Then, we turn to Valiant, Sharan and Sidford's result that built on Raz's work to linear regression. This is the key result, and we present the main theorem here precisely.

Theorem 1.1. *Let $(a_1, b_1), (a_2, b_2), ...$ be a stream of labeled examples, where $b_i = \langle a_i, x \rangle + \eta_i$, where $\eta_i \sim \mathcal{U}(-2^{-d/5}, 2^{d/5})$ and $||x||_2 = 1$. Any algorithm with at most $d^2/4$ bits of memory requires at least $\Omega(dlogr)$ samples to approximate to within ℓ_2 ϵ-error with probability of success at least 2/3, where $\epsilon = 1/d^r$ and $r \leq O(d/logd)$*

It is important to see what this is claiming. Strictly speaking, this is not a tradeoff but rather an impossibility result: if our memory is restricted, then we can guarantee an arbitraily good approximation with high probability only if we have enough samples. It does not say, away from $O(d^2)$ memory bits, how much we can compensate for a lack of sample data for computer memory, which could be a direction for future research. The next section covers Raz's paper that builds the foundation for analyzing memory tradeoffs in the linear regression setting.

2 Parity Learning and Branching Programs

The setting here is that of parity learning. We have access to a stream of samples $(a_1, b_1), ...$, where $a_t \in \{0,1\}^n$ and $b_t \in \{0,1\}$. We seek to learn $x \in \{0,1\}^n$, which is related to (a_t, b_t) by $b_t = \langle a_t, x \rangle$ where the inner product $\langle \cdot, \cdot \rangle$ is modulo 2. So this is a sort of regression over a field with two elements, and the next paper will generalize this further to over all the real numbers which is the scenario most familiar in applications. A critical element in this formulation is that once we receive a sample (a_t, b_t), we must store it explicitly in our memory or we lose it.

One technique is to solve the linear equation $Ax = b$, where b is a vector with entries of $[b_1, b_2, ...]$, and A is a matrix consisting of $a_1, a_2, ...$ stacked on top of each other. The matrix A is the largest contributor to memory, and has a size of $n \times n$, therefore we need a memory of $O(n^2)$ bits. In this case we only need $O(n)$ examples.

In contrast, noting that the set of all possible values of x has cardinality 2^n, we can do a brute-force search, where we only need to store $O(n)$ bits. We simply initialize x to a vector in $\{0,1\}^n$, taking up n bits, and loop through samples (a_t, b_t) until we hit an inconsistency. We then change our hypothesis x to rectify this inconsistency. In the worst-case, we would need exponential or $O(2^n)$ samples to shrink the hypothesis space to only one vector x.

Are these two cases exhaustive? Raz's key result is that, yes, they are, up to additive and multiplicative constants. As you can imagine, proving that a property holds for arbitrary algorithms is quite involved, and hence calls for some heavy technical machinery. We will focus on a higher level description as opposed to a rigorous proof. The core here is the branching program.

The strategy here is three steps:

1. All algorithms can be represented as a branching program
2. The alleged tradeoff holds for affine branching programs
3. All branching programs can be reduced to an affine branching program

1. A branching program is a directed acyclic graph with length m and width d. In our setting, m is the number of samples, where d is our memory.

A vertex represents the memory state of the program at that particular instance. Each layer $t \leq m$ represents one sample data (a_t, b_t), and the edges outgoing from that layer is a model of what the algorithm does with (a_t, b_t) to yield the next memory state. In order for the branching program to accurately represent any sample and any parity learning algorithm, we need 2^{n+1} outgoing edges at every vertex (with each edge representing a particular labeling of $(a_t, b_t) \in \{0,1\}^n \times \{0,1\}$).

Now, given our branching program, the samples $(a_1, b_1), ..., (a_n, b_n)$ define a computation path, which basically determines how the algorithm uses the stream of examples. At the last layer, or leaf nodes, the program outputs $w(v)$, which is not necessarily a single vector but a set of vectors in $\{0,1\}^n$. The target is to have $w(v)$ as small as possible with the condition that $x \in w(v)$.

2. An affine branching program is a branching program with the following two properties:

1. The starting vertex is empty

2. Each vertex v is labeled by a set of linear equations E_v such that if (u, v) is an edge, labeled by (a_t, b_t) as before, then $E_v \subset \{E_u \cup (\langle a_t, x\rangle = b_t)\}$

The second property is much more interesting. What it is saying is that the set of linear equations at a given vertex can only add the new linear equation from the new sample $\langle a_t, x\rangle = b$ to the subspace of linear equations inherited before. This guarantees that if our computation path reaches a vertex v, that the true x is contained within the set that satisfies E_v (so the algorithm is free to forget anything it wants but we cannot add in any new linear equations).

Further, this property will bound the growth of $dim(E_v)$ at each layer and this will be a critical ingredient in the proof of the tradeoff. A larger $dim(E_v)$ is better for learning: it will mean more equations will have to be satisfied, hence less degrees of freedom and a smaller hypothesis space (under the assumption that x will be in the hypothesis space, which is guaranteed by the affine branching program).

Therefore, suppose we have a branching program of width at most $2^{\epsilon n^2}$ and length $2^{\epsilon n}$ for some small ϵ. We will prove that we have an exponentially small probability of arriving at a fixed vertex v with $dim(E_v) \geq k$. If we do this, we can use the union bound over all the $2^{O(n^2)}$ vertices in our branching program, and, making sure constants check out, guarantee that getting to at least one vertex with such a dimension is exponentially low.

WLOG, we can focus on the case that $dim(E_v) = k$, since, by the nature of the affine program, there must have been a previous vertex that had a dimension exactly equal to k associated to it. If it takes exponentially low probability to hit this preliminary vertex, then it will by definition take exponentially low probability to hit the vertex with dimension greater than k case.

Let $V_0, ..., V_m$ be the vertices on the graph. However, for simplicity of exposition here I further assume that if v is within the last layer, so if v is within our computation path, it must be that $V_m = v$. A critical quantity is $r_i = dim(E_v \cap E_{Vi})$, where $r_0 = 0$ and, $r_i = k$ if $V_m = v$. Note that by affine branching $r_{i+1} \leq r_i + 1$, so in order for r_i to reach k, r_i must grow at each step (because we have only m data which implies we have a branching program of width $2^O(m)$).

By the rules of the branching program, the only case that $r_{i+1} = r_i + 1$ is when $(a_{i+1}, b_{i+1}) \in Span\{E_{V_i} \cup E_v\}$. Therefore, $Pr(r_{i+1} > r_i) = Pr(r_{i+1} = r_i + 1) \leq 2^{-(n-2k)}$, since this is the probability that a random sample (a_{i+1}, b_{i+1}) is within the aforementioned span. (To see this explicitly, we have k from E_v, and E_{V_i} is bounded above by k to yield $2k$. We subtract off n to avoid double counting the intersection of E_{V_i} and E_v.)

Finally, in order for r_i to equal k, the above probability needs to happen at least k times. By union bound over these k events, we yield $m^k 2^{-(n-2k)k} = 2^{-\Omega(n^2)}$ for appropriate constants (like $k = \frac{1}{5}n$).

3. Finally, the last ingredient is to prove that a branching program can be reduced to an affine branching program, and that this reduction is not wasteful to the extent that the tradeoffs we have already established breakdown. If we have too many vertices, for example, the union bound applied in 2. by itself might result in a superexponential probability of success.

Let $\mathcal{U}_{E_v}$ represents the uniform distribution over all elements satisfying the equations in E_v. We define an ϵ-accurate branching program to be one such that the following holds:

$$\mathbb{E}_{E_v} |P_{x|E_v} - \mathcal{U}_{E_V}|_1 \leq \epsilon \tag{1}$$

Our target is to have a branching program that achieves a satisfactorily high-level of approximation without blowing up the number of vertices. An $\epsilon = 4m2^{-(r-\frac{n}{2}}$ is used in the paper

Mathematical induction is used in the proof, as each layer of the branching program to an affine branching program layer by layer. Take a vertex in this layer i. By the inductive hypothesis on layer $i-1$, which is now assumed to have already been converted to an affine branching program, this vertex is ϵ-close to a convex combination of uniform distributions over affine subspaces. We could split this vertex v into one vertex for each

affine subspace, but this would cause the graph to blow up. The author shows that we can group some of these affine subspaces together, but we do not go over that because it is quite detailed.

Once this is done, we can take an arbitrary branching program of length at most $2^{\epsilon n}$ and width at most $2^{\epsilon n^2}$, and simulate it with an affine branching program as we did in 3., then apply basically the same reasoning as we did 2. This schema turns out to be quite delicate, as constants play a critical role to ensure that the reduction to an affine branching program does not cause the probability of success to increase. In the author's own words, "We note that this reduction is very wasteful and expands the width of the branching program by a factor of $2^{\Theta(n^2)}$. Nevertheless, since we allow our branching program to be of width up to $2^{O(n^2)}$, this is still affordable (as long as the exact constant in the exponent is relatively small)"

3 From Parity Learning to Real-Valued Regression

Of course, while a fascinating result, Raz's result does not seem to be awfully applicable to statistics or machine learning. Valiant, Sidford and Sharan prove a similar result but this time for linear regression, a cornerstone of statistics and machine learning, using Raz's work on branching programs as a solid foundation. It is a fascinating transformation of a result in a discrete space to a continuous space.

The foundations provided by Raz's branching program are retained, and the proof proceeds in a similar manner layer by layer. The big difference is that the error bounding at each layer now occurs in a continuous space as opposed to a discrete one. What changes are needed to go from parity learning to linear regression? Going from a combinatorial space like $\mathbb{F}_2$ to a continuous space like $\mathbb{R}$ requires us to adapt multiple parts of the previous proof.

1. Instead of edges $\{u, v\}$ representing a mapping from an example to each potential labeling $\{0,1\}^n \times \{0,1\}$, we have a transition function $f_u : \mathbb{R}^n \times \mathbb{R} \to v$ since now we have an infinite number of possible labelings

2. First-order linear methods can exploit the topological structure of $\mathbb{R}$ to yield non-trivial error at each layer. This is in contrast to the brute-force approach in the discrete case

3. As is typical in real-valued settings, success is defined as an ϵ-close approximation as opposed to an exact match

Despite these differences, the proofs share critical characteristics. The proofs assumes, by contradiction, that we have a branching program of at most width $2^{\epsilon n^2}$ and length $2^{\epsilon n}$ for some small ϵ. Then it proves the probability of hitting a significant vertex is very low, using mathematical induction on the layers again.

Where significant vertex in Raz's paper meant an algebraically high-dimensional subspace of linear equations (which then limits the number of potential targets x), here, what is meant by significant vertex is one that, conditional on the true x, $||\mathbb{P}_{x|s}||_2 \geq \epsilon$ for some threshold parameter ϵ. Heuristically, a high value of $||\mathbb{P}_{x|s}||$ means non-uniformity of the distribution and hence concentration of the distribution on a value (and like the algebraic case, informativeness of the estimate). This is a microcosm of how Valiant, Sidford and Sharan adjust Raz's original technique to accommodate the real line.

Mathematical induction is used very similarly as the way Raz used it. The proof proceeds layer by layer, and it shows that hitting a significant vertex is small for every layer.

A potential function is defined (this is like the measure of the dimensions of the two subspaces of linear equations in Raz), where L_i denotes the set of vertices at the i-th layer,

$$\mathcal{Z}_i = \sum_{v \in L_i} Pr(v) \langle \mathbb{P}_{x|v}, \mathbb{P}_{x|s} \rangle^{d/2}$$

(This potential function plays the same role as the measure of the dimension $r_i = dim(E_v \cap E_{Vi})$ in Raz's paper.)

Trivially, the following inequality holds if $s \in L_i$

$$\mathcal{Z}_i \geq Pr(s)||\mathbb{P}_{x|s}||_2^d$$

Therefore, if we prove $\mathcal{Z}_i$ is small, then the probability of reaching the significant vertex s is also quite small.

We define another potential function, which measures how much progress an edge of the program makes towards $\mathcal{Z}_i$.

$$\mathcal{Z}_i^{'} = \int_{e \in support(f_v)} p(e)\langle \mathbb{P}_{x|e}, \mathbb{P}_{x|s} \rangle^{d/2} de$$

Where in Raz's paper we had $r_{i+1} \leq r_i + 1$, and it was assumed by the inductive hypothesis that r_i is not large, here we also seek a bound of the form $\mathcal{Z}_{i+1} \leq \mathcal{Z}_i$ for $c \in \mathbb{R}$,

We therefore prove $\mathcal{Z}_{i+1} \leq \mathcal{Z}'_{i+1}$ and $\mathcal{Z}'_{i+1} \leq \mathcal{Z}_i A + B$, for some $A, B \in \mathbb{R}$.

For the first inequality,

$$\mathcal{Z}_i = \sum_{v \in L_i} Pr(v)\langle \mathbb{P}_{x|v}, \mathbb{P}_{x|s} \rangle^{\beta d} \underset{\text{Jensen's ineq.}}{\leq} \sum_{v \in L_i} Pr(v) \int_{e \in support(f_v)} \frac{p(e)}{Pr(v)} \langle \mathbb{P}_{x|e}, \mathbb{P}_{x|s} \rangle^{\beta d} de$$

$$= \int_{e \in support(f_v)} p(e)\langle \mathbb{P}_{x|e}, \mathbb{P}_{x|s} \rangle^{\beta d} de = \mathcal{Z}'_i$$

For the second inequality, a string of lemmas and similar convex analysis techniques are used, which we omit here for brevity. Patching these two inequalities together we yield a statement in the form $\mathcal{Z}_{i+1} \leq \mathcal{Z}_i A + B$. By Lemma 10 of the paper, $\mathcal{Z}_0$ is bounded usefully, so we can iteratively plug in the values for $\mathcal{Z}_{i-1}$ to ultimately yield

$$\mathcal{Z}_i \leq \frac{(d^3/\delta)(1/\epsilon_t)^{2\beta d^2} 2^{1.25\beta d^2}}{C_d^2}$$

Letting $Pr(s)$ denote the probability of a given vertex being a significant vertex, and noting once again that

$$\mathcal{Z}_i \geq Pr(s)\langle \mathbb{P}_{x|s}, \mathbb{P}_{x|s} \rangle$$

and using a number of other inequalities, we ultimately yield

$$Pr(s) \leq (d^3/\delta)^{\beta d} 2^{-0.75\beta d^2}$$

So we have bounded the probability of one vertex being a significant vertex. Now we union bound over all vertices in the program.[1]. This ultimately yields a probability of reaching a significant vertex at any part of the branching program being 2^{-d}, concluding the proof.

References

[1] R. Raz, "Fast learning requires good memory: A time-space lower bound for parity learning," 2016.

[2] V. Sharan, A. Sidford, and G. Valiant, "Memory-sample tradeoffs for linear regression with small error," 2019.

[1] Or precisely bound over all vertices in a given stage of the program, then union bound over all stages

[3] M. Jordan, "What are the open problems in Bayesian statistics," *The ISBA Bulletin*, vol. 18, no. 1, p. 568, 2011. [Online]. Available: http://bayesian.org/sites/default/files/fm/bulletins/1103.pdf

[4] I. for Advanced Studies. Fast learning requires good memory - ran raz. Youtube. [Online]. Available: https://www.youtube.com/watch?v=BC3sIvmUAHg