2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS) | 979-8-3503-1894-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/FOCS57990.2023.00070

Tight Time-Space Lower Bounds for Constant-Pass Learning

Xin Lyu
EECS
University of California, Berkeley
Berkeley, US
xinlyu@berkeley.edu

Avishay Tal
EECS
University of California, Berkeley
Berkeley, US
atal@berkeley.edu

Hongxun Wu
EECS
University of California, Berkeley
Berkeley, US
wuhx@berkeley.edu

Junzhao Yang
IIIS
Tsinghua University
Beijing, China
yang-jz20@mails.tsinghua.edu.cn

***Abstract*—In his breakthrough paper, Raz showed that any parity learning algorithm requires either quadratic memory or an exponential number of samples [FOCS'16, JACM'19]. A line of work that followed extended this result to a large class of learning problems. Until recently, all these results considered learning in the streaming model, where each sample is drawn independently, and the learner is allowed a single pass over the stream of samples. Garg, Raz, and Tal [CCC'19] considered a stronger model, allowing multiple passes over the stream. In the 2-pass model, they showed that learning parities of size n requires either a memory of size $n^{1.5}$ or at least $2^{\sqrt{n}}$ samples. (Their result also generalizes to other learning problems.)**

In this work, for any constant q, we prove tight memory-sample lower bounds for any parity learning algorithm that makes q passes over the stream of samples. We show that such a learner requires either $\Omega(n^2)$ memory size or at least $2^{\Omega(n)}$ samples. Beyond establishing a tight lower bound, this is the first non-trivial lower bound for q-pass learning for any $q \geq 3$. Similar to prior work, our results extend to any learning problem with many nearly-orthogonal concepts.

We complement the lower bound with an upper bound, showing that parity learning with q passes can be done efficiently with $O(n^2/\log q)$ memory.

***Index Terms*—Multi-pass, streaming, parity learning**

I. Introduction

A growing recent line of works studied the efficiency of learning under memory constraints [1]–[18]. This study was initiated by the beautiful work of Shamir [1] and Steinhardt, Valiant, and Wager [2]. Specifically, Steinhardt et al. [2] conjectured that any learning parity algorithm requires either quadratic memory or an exponential number of examples. In a breakthrough result, Raz [3] proved this conjecture. While we have two simple algorithms for parity learning: (i) Gaussian Elimination that uses $O(n^2)$ space and $O(n)$ samples, and (ii) Brute-force search that uses $O(n)$ space and $O(2^n)$ samples, Raz showed that there is no learning algorithm that uses $o(n^2)$ space and $2^{o(n)}$ samples [3]. This demonstrated that efficient learning requires a large memory – in this case, at least $\Omega(n^2)$ memory bits.

Follow-up work extended and generalized the lower bounds techniques to a wide array of learning problems such as learning sparse parities, learning DNFs, learning decision trees, learning juntas, [4], [9] learning low-degree polynomials [9], [10], learning from sparse equations and low-degree equations [9], learning codewords from random coordinates [7]–[9], learning parities with noisy inputs [16], and more. In all the above, it is shown that any learning algorithm for the corresponding concept class on input size n, requires either super-linear size memory, or super polynomial number of samples. Work towards a tight characterization of memory-samples lower bounds was done by [19], but such a full characterization is still missing with polynomial gaps on the memory required for efficiently learning classical concepts classes such as juntas, DNFs, decision trees [4].

Most of the works above modeled the learner as a streaming algorithm, observing the random labeled examples one at a time. More precisely, the lower bounds proved were in the stronger model of read-once branching programs that captures bounded-space streaming computation in a non-uniform setting. Recent exciting work by [18] extended the model to include quantum memory in addition to classical memory and showed that Raz's result extends even if the learner has additionally $o(n)$ qubits at its disposal.

Dagan and Shamir [11] and Garg, Raz, and Tal [12] considered the model of multi-pass learners. In this model, the learner makes several passes over the stream of examples in the same order. Dagan and Shamir [11] proved polynomial lower bounds on the number of samples in such setting. Garg, Raz and Tal [12] obtained a subexponential lower bound on the number of samples $2^{\Omega(\sqrt{n})}$ for any *two-pass* learning parity algorithm with $o(n^{1.5})$ space. The result more generally

Avishay Tal is supported by Sloan Research Fellowship and NSF CAREER Award CCF-2145474. Xin Lyu and Hongxun Wu are supported by Avishay Tal's Sloan Research Fellowship, NSF CAREER Award CCF-2145474, and Jelani Nelson's ONR grant N00014-18-1-2562.

979-8-3503-1894-4/23/$31.00 ©2023 IEEE

DOI 10.1109/FOCS57990.2023.00070

implies lower bounds for any of the aforementioned learning problems. Indeed, the lower bounds are proved in the extractor-based framework of [9] and all the aforementioned learning problems fall under this framework.

Despite the strong lower bound, the GRT result was not known to be tight for two-pass learning, as no efficient algorithm with $o(n^2)$ space was known in this setting. Moreover, their result did not translate to the multi-pass setting with more than two passes, and, as indicated in their paper, some of their techniques are quite delicate, and it is far from clear how to extend them to more than two passes [12].

Proving lower bounds for multi-pass learners is much more challenging, as such learners can store information during the first pass that would make examples in the second pass somewhat predictable, correlated with one another, or correlated with the hidden vector.

One might wonder whether more passes can help in learning. Indeed, when the number of passes is quasi-polynomial, a parity learning algorithm with $n^{O(\log n)}$ passes, $n^{O(\log n)}$ samples, and $O(n)$ space follows from the following two facts: (i) solving linear equations can be done in $O(\log^2 n)$ depth [20] (ii) Barrington's simulation of $O(\log^2 n)$ depth by length $n^{O(\log n)}$ read-once branching programs [21].

A. Our Results

We study time-memory lower bounds for multi-pass learning problems. We provide a nearly tight lower bound for two-pass learning parity algorithms:

Theorem 1 (Informal)**.** *Any two-pass algorithm for n-bit parity learning requires either $\Omega(n^2)$ bits of memory or $2^{\Omega(n)}$ many samples. Otherwise, the algorithm succeeds with probability at most $2^{-\Omega(n)}$.*

Moreover, our results generalize to any constant-pass learner and, moreover, imply nearly similar bounds for any algorithm with at most $o(\log\log n)$ passes.

Theorem 2 (Informal)**.** *There is a universal constant $C > 0$ such that the following holds. For any $q \geq 2$, letting $c_q = C \cdot 100^{3^q}$, any q-pass algorithm for n-bit parity learning requires either n^2/c_q bits of memory or $\exp(n/c_q)$ many samples. Otherwise, the algorithm succeeds with probability at most $2^{-(n/c_q)}$.*

We stress that the multi-pass lower bound is not a direct generalization of the two-pass one. It requires us to revisit a key technique in the two-pass proof (which we call the "transfer lemma"), and extend the technique to the multi-pass case with a significantly more involved argument.

Extractor-based framework: Our results apply more generally to any learning problem with many nearly pairwise orthogonal concepts (i.e. concepts that agree on roughly half of the inputs). Alternatively, to any learning problem whose associated matrix (as defined in [7]) exhibits an extractor-property [9], as defined next.

Let A be a finite domain, and let X be a concept class over A, where each $x \in X$ represents a function (or concept) mapping A to $\{-1, 1\}$. We naturally associate with the concept class a matrix $M \in \{-1, 1\}^{A \times X}$ whose rows correspond to samples and columns correspond to concepts/functions. Then, M describes the following learning problem: An unknown $x \in X$ is chosen uniformly at random. A learner tries to learn x from a stream of labeled samples, $(a_1, M(a_1, x)), (a_2, M(a_2, x)), \ldots$ where each a_i is uniformly distributed over A. In particular, we consider the setting in which the learner can see the *same* stream of samples for $q \geq 2$ passes.

Our lower bounds apply to any learning problem whose corresponding matrix M has certain extractor properties: Any large submatrix of M has a similar fraction of 1's and -1's. More precisely, we say that M is a (k, ℓ, r)-extractor if for any submatrix of at least $2^{-k} \cdot |A|$ rows and at least $2^{-\ell} \cdot |X|$ columns, the fraction of entries with value 1 is $\frac{1}{2} \pm 2^{-r}$. (For example, parity learning has parameters $k, \ell, r = \Omega(n)$.) We show that any two-pass learning for the learning problem associated with M requires either $\Omega(k \cdot \min(\ell, k))$ memory or at least $2^{\Omega(r)}$ samples. For q-pass learning, we show that the learning problem requires either $\Omega(k \cdot \min(\ell, k))/c_q$ memory or at least $2^{\Omega(r/c_q)}$ samples for $c_q = 100^{3^q}$.

The formal version of Theorem 1 and Theorem 2 [1] are actually stated for matrices that are L_2-extractors, since these extractors are more convenient to work with in our proof. However, a simple reduction from [9, Corollary 3] shows that any standard extractor as above is also a $(\Omega(k+r), \Omega(\ell+r), \Omega(r))$-$L_2$-Extractor. Our results thus apply to all the aforementioned concept classes (juntas, DNFs, Decision trees, low-degree polynomials, codewords) as their corresponding matrices form L_2-Extractors with good parameters.

A non-trivial multi-pass algorithm: One might wonder if the lower bound can be strengthened to show that any $n^{o(1)}$-pass learner requires either $\Omega(n^2)$-memory or $2^{\Omega(n)}$ samples to learn parity. Our next result shows that this is not the case, and efficient learning with $o(n^2)$ memory is possible for any $q = \omega(1)$.

Theorem 3 (Informal)**.** *For any $q \leq 2^n$, there is a q-pass algorithm for n-bit parity learning that uses $O(n^2/\log(q))$ bits of memory and $O(qn)$ samples.*

II. Technical Overview

In this section, we will present the road map for our paper, including the difficulties and a sketch of our main ideas for bypassing them.

A. Recap of the One-Pass Lower Bound

Both our work and the previous work on the two-pass learning bound [12] are based on the proof techniques for one-pass lower bound [7], [9]. Let us sketch its main idea.

Computational Model: The proof models the computation as a read-once branching program: The input to the branching program is a sequence of pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_T, b_T) \in A \times \{-1, 1\}$. Each of them

[1] See Theorem 5 and Theorem 6 in the full version of this paper.

represents an equation $M(a_i, x) = b_i$. These $a_1, a_2, \ldots, a_T \in A$ are sampled uniformly at random, while $b_1, b_2, \ldots, b_T$ are all generated according to the hidden vector $x \in X$. (In our paper, for simplicity, we will identify X with $\{0,1\}^n$.) We label the layers of the branching program by $0, 1, 2, \ldots, T$. Let v be the current vertex. Initially, it is equal to the starting vertex of the branching program at layer 0. After it is at layer i, we read (a_{i+1}, b_{i+1}) and move v along one corresponding edge to layer $i+1$. At the end of the computation, v will reach the last layer T. Then it outputs a vector x_v. We say that it is successful if and only if $x_v = x$.

The length of this branching program is the number of samples T. The width is 2^S where $S \le n^2/16$ is the memory bound. We want to prove that when $S \le n^2/16$ and $T \le 2^{n/16}$ the program cannot succeed with constant probability.

Main Idea of the One-Pass Lower Bound: When outputting x_v, the optimal strategy is to output the x' with the highest posterior probability $\mathbb{P}_{x|v}(x')$. Intuitively, if the distribution $\mathbb{P}_{x|v}$ is very spread, measured by its ℓ_2 norm, the vertex v will have a small chance of answering x correctly. We will define its ℓ_2 norm as

$$\|\mathbb{P}_{x|v}\|_2 := \mathop{\mathbf{E}}_{x' \sim X} \left[\mathbb{P}^2_{x|v}(x')\right]^{\frac{1}{2}}.$$

Initially, as $X = \{0,1\}^n$, the uniform prior $\mathbb{P}_x$ has $\|\mathbb{P}_x\|_2 = 2^{-n}$. As v moves along the computational path, the posterior distribution $\mathbb{P}_{x|v}$ evolves. In the end, one can show that, for some $\epsilon > 0$, if $\|\mathbb{P}_{x|v}\|_2 \le 2^{\epsilon n} \cdot 2^{-n}$, the probability that v answers x correctly will be less than $2^{-\Theta(n)}$. (Think of ϵ as a small constant, say $\epsilon = 0.1$.)

Hence, to upper bound the success probability, it is sufficient to upper bound the probability that we ever reach a vertex v with $\|\mathbb{P}_{x|v}\|_2 > 2^{\epsilon n} \cdot 2^{-n}$ on our computational path. To show this, we will enumerate all target vertices t with $\|\mathbb{P}_{x|t}\|_2 > 2^{\epsilon n} \cdot 2^{-n}$ and try to show that the probability of reaching a fixed t is less than $2^{-\Theta(n^2)}$. Then our desired upper bound follows from a union bound over all $2^{n^2/16+O(n)}$ possibilities of vertex t.

Progress Measure: To study the probability of reaching t, we need to look at the similarity between our current posterior $\mathbb{P}_{x|v}$ and the target $\mathbb{P}_{x|t}$, captured by their inner product $\langle \mathbb{P}_{x|v}, \mathbb{P}_{x|t} \rangle$, defined as

$$\begin{aligned}\left\langle \mathbb{P}_{x|v}, \mathbb{P}_{x|t} \right\rangle &:= \mathop{\mathbf{E}}_{x' \sim X} \left[\mathbb{P}_{x|v}(x') \cdot \mathbb{P}_{x|t}(x')\right] \\ &= \frac{1}{|X|} \sum_{x' \in X} \mathbb{P}_{x|v}(x') \cdot \mathbb{P}_{x|t}(x').\end{aligned}$$

This measures our progress towards t. To show that we reach t with a very small probability, we will show that for a uniform random $a \in A$, reading equation $M(a, x) = b$ will, w.h.p., makes little progress.

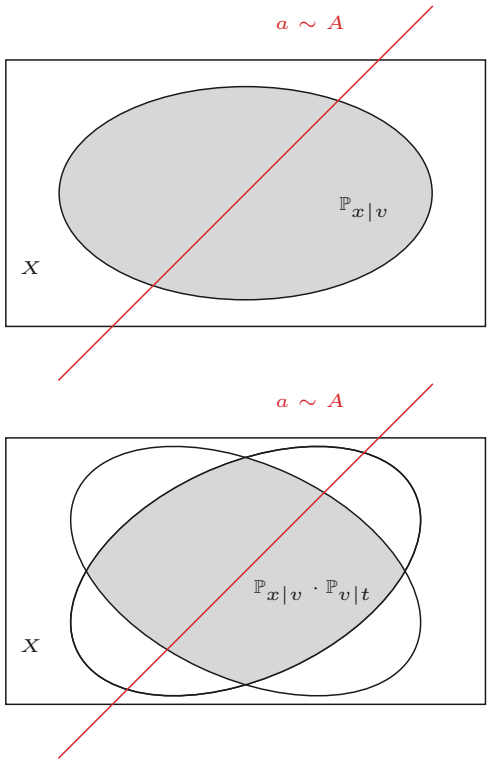

Fig. 1: A uniformly random equation $a \sim A$ will w.h.p. cut both $\mathbb{P}_{x|v}$ and $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ almost evenly into two parts: (1) those $x \in X$ with $M(a, x) = 0$ and (2) those $x \in X$ with $M(a, x) = 1$.

Let the posterior distribution after reading this equation be $\mathbb{P}^{(a,b)}_{x|v}$. Then after normalization, the similarity becomes

$$\begin{aligned}&\left\langle \mathbb{P}^{(a,b)}_{x|v}, \mathbb{P}_{x|t} \right\rangle \\ &= \frac{1}{|X|} \cdot \sum_{\substack{x' \in X \\ M(a,x')=b}} \mathbb{P}_{x|v}(x') \cdot \mathbb{P}_{x|t}(x') \Big/ \sum_{\substack{x' \in X \\ M(a,x')=b}} \mathbb{P}_{x|v}(x').\end{aligned}$$

We say $a \in A$ cuts $\mathbb{P}_{x|v}$ evenly if

$$\sum_{x' \in X : M(a,x')=0} \mathbb{P}_{x|v}(x') \approx \sum_{x' \in X : M(a,x')=1} \mathbb{P}_{x|v}(x').$$

Similarly, we say $a \in A$ cuts the point-wise product $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ evenly if this holds for $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ instead of $\mathbb{P}_{x|v}$.

As shown in Figure 1, due to the extractor property of M, when $\mathbb{P}_{x|v}$ and $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ are spread enough, a *uniformly random* $a \in A$ cuts both $\mathbb{P}_{x|v}$ and $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ in half with high probability $(1 - 2^{-\Theta(n)})$. Hence each time we see a random equation, it will most likely halve both the numerator and the denominator, which will not help us make progress. Suppose we are unlucky. the rare event with probability $2^{-\Theta(n)}$ happens. Let us see how the similarity might change:

1) If $a \in A$ cuts $\mathbb{P}_{x|v}$ unevenly: When the denominator $\sum_{x' \in X, M(a,x')=b} \mathbb{P}_{x|v}(x') < 2^{-c}$, the similarity may be

larger by a factor of 2^c. This causes huge progress. For now, we ignore this issue and assume it never arises. We will later handle it by designing certain "stopping rules".

2) If $a \in A$ cuts $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ unevenly but still cut $\mathbb{P}_{x|v}$ evenly: In this case, the worst case is that the numerator does not decrease at all, while the denominator is still halved. Then the similarity doubles.

Initially, the similarity between uniform prior and target t is $\langle \mathbb{P}_x, \mathbb{P}_{x|t} \rangle = 2^{-2n}$. In order to reach the target node t, which has similarity $\langle \mathbb{P}_{x|t}, \mathbb{P}_{x|t} \rangle = \|\mathbb{P}_{x|t}\|_2^2 \geq 2^{2\epsilon n} \cdot 2^{-2n}$ with itself, the second case has to happen $2\epsilon n$ times. Intuitively, this tells us the probability of reaching t is less than $\left(2^{-\Theta(n)}\right)^{2\epsilon n} = 2^{-\Theta(n^2)}$.

Stopping Rules: However, we still have to handle the first case. Although it also happens with probability only $2^{-\Theta(n)}$, it only needs to happen once to make huge progress. The $2^{-\Theta(n)}$ probability is not enough to afford the union bound over all $2^{\Theta(n^2)}$ many targets t.

Luckily, we do not have to union bound. Observe that whether $a \in A$ cuts $\mathbb{P}_{x|v}$ evenly is independent of the target t. Whenever we see an equation $a \in A$ that cuts $\mathbb{P}_{x|v}$ unevenly in our computational path, we can stop right away. Since there are $T \approx 2^{n/16}$ many layers in our branching program, a simple union bound over them shows that the overall probability of stopping is still $2^{-\Theta(n)}$. Moreover, if we did not stop, then the previous argument shows that we reach any target vertex t with probability $2^{-\Theta(n^2)}$. Overall, our algorithm succeeds with a very small probability.

But this is not the only stopping rule. Recall that a uniformly random $a \in A$ cuts both $\mathbb{P}_{x|v}$ and $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ evenly w.h.p. (by extractor property) only when they are spread enough. We also need stopping rules to guarantee this. Formally, we have the following stopping rules.

- (Bad Edge) If a does not cut $\mathbb{P}_{x|v}$ evenly, we stop.
- (Significant State) If $\|\mathbb{P}_{x|v}\|_2 \geq 2^{\epsilon n} \cdot 2^{-n}$, we stop. This guarantees that the distribution of $\mathbb{P}_{x|v}$ will be spread enough for the extractor property.
- (Significant Value) If $\mathbb{P}_{x|v}(x) > 2^{\epsilon n} \cdot 2^{-n}$, we stop. After applying this rule, we know $\|\mathbb{P}_{x|v}\|_\infty \leq 2^{\epsilon n}$. Since $\|\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}\|_2 \leq \|\mathbb{P}_{x|v}\|_\infty \cdot \|\mathbb{P}_{x|t}\|_2$, this guarantees that $\mathbb{P}_{x|v} \cdot \mathbb{P}_{x|t}$ will be spread enough for the extractor property.

B. The Proof Framework. Two Passes

Our work builds on the approach taken by the previous two-pass lower bound [12]. We will now sketch their proof framework.

Computational Model: A two pass branching program reads its input $(a_1, b_1), (a_2, b_2), \ldots, (a_T, b_T)$ twice in the *exact same order*. At the first pass, the starting vertex is v_0, and after reading its input, the computational path reaches a vertex v_1 at the end of the first pass (which is also the first layer of the second pass). In the second pass, the computational path starts from v_1 and reaches v_2 at the last layer after reading the input again. Then it will output a vector $x_{v_2} \in X$.

For any two vertices u and v in the program, we use $u \overset{\sim}{\to} v$ to denote the following event (over $x, a_1, a_2, \ldots, a_T$): Imagine that we set the starting vertex of the branching program at u, the path from u determined by $x, a_1, a_2, \ldots, a_T$ reaches v without stopping.

- For a vertex v_1 in the last layer of the first pass, $v_0 \overset{\sim}{\to} v_1$ means that the first pass ends at v_1.
- For any vertex v_1 in the last layer of the first pass, and vertex v_2 in the last layer of the second pass, $v_1 \overset{\sim}{\to} v_2$ means that the second pass will end at v_2 if it were starting at v_1.

First Attempt: Moving from one pass to two passes, one might consider the following natural approach: First, apply the above argument to the first pass and conclude that, at the end of the first pass, the similarity $\langle \mathbb{P}_{x|v}, \mathbb{P}_{x|t} \rangle$ is small. Second, apply it to the second pass and argue that such similarity grows slowly in the second pass too.

However, such a direct approach would not work. Consider a program that (1) magically learns x, (2) remembers $x \oplus a_1$ (thinking of x and a_1 as bit strings), and (3) forgets x and a_1 at the end of the first pass. Conditioning on what v remembers, $x \oplus a_1$, the distribution $\mathbb{P}_{x|v}$ is uniformly random, just like the prior $\mathbb{P}_x$. This is because x is encrypted by the one-time pad using a_1. So in the eyes of our analysis, this magical first pass is no different from a trivial first pass. If we do not rule out the possibility of such a magical first pass, what could happen in the second pass is that, after seeing a_1 again, the program combines a_1 with its knowledge of $x \oplus a_1$ and immediately decodes x.

Remembering the First Pass: To prove any non-trivial lower bound for two passes, it is necessary to rule out such a program. This program shows that analyzing two passes separately would not work (at least for this specific argument). Therefore, we will analyze two passes together.

The first observation is that one can w.l.o.g. assume that, when at the i-th layer of the second pass, the program knows which vertex it was at in the i-th layer of the first pass. This is because the program can keep a copy of the first pass in its memory, which only blows up the memory by a factor of two.

More formally, we modify the second pass (See Figure 2), so that every vertex is now a pair of an original first-pass vertex and an original second-pass vertex. The initial starting vertex of the second pass becomes (v_0, v_1). When the program reads the first equation (a_1, b_1), if in the first pass $v_0 \overset{\sim}{\to} v'$ and in the second pass $v_1 \overset{\sim}{\to} v$, in the modified second pass, $(v_0, v_1) \overset{\sim}{\to} (v', v)$.[2]

Now in the modified program, every modified vertex v in the second pass corresponds to ("remembers") a unique vertex v' in the first pass. The event $v_1 \overset{\sim}{\to} v$ now implies $v_0 \overset{\sim}{\to} v'$, in the sense that for any input $x, a_1, a_2, \ldots, a_T$ such that $v_1 \overset{\sim}{\to} v$ happens, the event $v_0 \overset{\sim}{\to} v'$ must also happen.

[2]For the exact details of this modification, please refer to Section 4 of the full version.

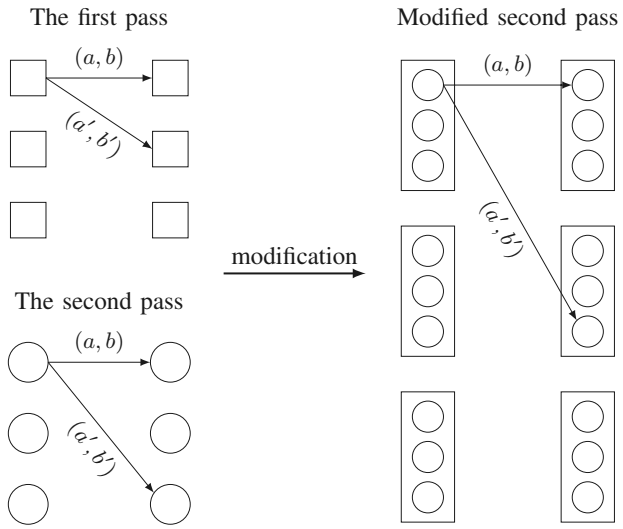

Fig. 2: Remembering the first pass.

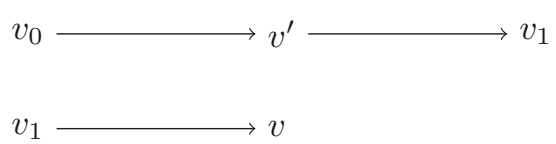

Fig. 3: The computational path of two passes.

Furthermore, we require every vertex in the second pass to remember v_1, the starting vertex of the second pass. This can be achieved with a similar modification. By this modification, for any vertex v in the second pass (which remembers its corresponding vertex v' in the first pass),

$$\begin{aligned}(v_0 \overset{\frown}{\to} v) &= (v_0 \overset{\frown}{\to} v') \wedge (v' \overset{\frown}{\to} v_1) \wedge (v_1 \overset{\frown}{\to} v) \\ &= (v_1 \overset{\frown}{\to} v) \wedge (v' \overset{\frown}{\to} v_1).\end{aligned}$$

Here the last equality holds because $v_1 \overset{\frown}{\to} v$ implies $v_0 \overset{\frown}{\to} v'$, and v_1 is unique since v remembers it.

When $v = v_2$ for some vertex v_2 in the last layer (of the second pass), this simplifies to

$$\begin{aligned}(v_0 \overset{\frown}{\to} v_2) &= (v_0 \overset{\frown}{\to} v_1) \wedge (v_1 \overset{\frown}{\to} v_2) \\ &= (v_1 \overset{\frown}{\to} v_2).\end{aligned}$$

Progress Measure: When the program reaches v_2, the optimal strategy for it is to output the $x' \in X$ with the highest $\mathbb{P}_{x|v_0 \overset{\frown}{\to} v_2}(x')$. Note that this equals $\mathbb{P}_{x|v_1 \overset{\frown}{\to} v_2}(x')$ (by the equation above).

Hence, similar to the one-pass case, when the distribution $\mathbb{P}_{x|v_1 \overset{\frown}{\to} v_2}$ is spread enough ($\|\mathbb{P}_{x|v_1 \overset{\frown}{\to} v_2}\|_2 \geq 2^{\epsilon n} \cdot 2^{-n}$), vertex v_2 cannot answer x correctly. To upper bound the probability of answering x correctly, we only need to upper bound the probability of reaching any target state t with $\|\mathbb{P}_{x|v_1 \overset{\frown}{\to} t}\|_2 \geq 2^{\epsilon n} \cdot 2^{-n}$.

Initially, $\mathbb{P}_{x|v_1 \overset{\frown}{\to} v_1} = \mathbb{P}_x$. As the current vertex v moves along the computational path from v_1, the posterior $\mathbb{P}_{x|v_1 \overset{\frown}{\to} v}$ evolves similarly to the one-pass case. Let the similarity $\langle \mathbb{P}_{x|v_1 \overset{\frown}{\to} v}, \mathbb{P}_{x|v_1 \overset{\frown}{\to} t} \rangle$ be the progress measure, and let $\mathbb{P}^{(a,b)}_{x|v_1 \overset{\frown}{\to} v}$ be the posterior after reading a new equation (a,b). We have,

$$\begin{aligned}&\left\langle \mathbb{P}^{(a,b)}_{x|v_1 \overset{\frown}{\to} v}, \mathbb{P}_{x|v_1 \overset{\frown}{\to} t} \right\rangle \\ &\quad = \frac{1}{|X|} \sum_{\substack{x' \in X \\ M(a,x')=b}} \mathbb{P}_{x|v_1 \overset{\frown}{\to} v}(x') \cdot \mathbb{P}_{x|v_1 \overset{\frown}{\to} t}(x') \\ &\qquad\qquad \Big/ \sum_{\substack{x' \in X \\ M(a,x')=b}} \mathbb{P}_{x|v_1 \overset{\frown}{\to} v}(x').\end{aligned}$$

Until now, this seems like a natural generalization of the one-pass case. However, for one pass, we heavily rely upon the fact that $a \in A$ is uniformly random. In the second pass, we no longer have this property. For example, the program could simply remember $a_1 \in A$ from the first pass, then in the second pass, a_1 is completely deterministic, with no randomness at all.

High-Probability Edges: The previous work [12] calls such $a_i \in A$ that is remembered by the program a *high-probability edge*. Formally, for a vertex v in the i-th layer of the second pass, we say that a is a high-probability edge at v (denoted by $a \in \mathrm{High}(v)$) if and only if

$$\mathbf{Pr}[a_{i+1} = a \mid v_0 \overset{\frown}{\to} v] \geq 2^{\epsilon \cdot n} \cdot 2^{-n}.$$

Since these edges occur with too large probability (much higher than the uniform case, 2^{-n}), we cannot simply stop when they cut distributions unevenly (like we did for one-pass).

1) If this $a \in A$ cuts $\mathbb{P}_{x|v_1 \overset{\frown}{\to} v}$ unevenly: When the denominator $\sum_{x' \in X, M(a,x')=b} \mathbb{P}_{x|v_1 \overset{\frown}{\to} v}(x') < 2^{-c}$, the similarity might be larger by a factor of 2^c, causing huge progress. For now, we ignore this issue and assume that it never arises. We will explain how we handle it in Section II-C.
2) If this $a \in A$ cuts $\mathbb{P}_{x|v_1 \overset{\frown}{\to} v} \cdot \mathbb{P}_{v_1 \overset{\frown}{\to} t}$ unevenly, but still cuts $\mathbb{P}_{x|v_1 \overset{\frown}{\to} v}$ evenly: In this case, the worst case is the sae as one-pass. Namely, the numerator does not decrease at all, while the denominator is halved. Then the similarity at most doubles.

Their key observation is the following. Intuitively, to remember a single $a \in A$, we need at least $\Omega(n)$ memory. Since the memory bound $S \leq n^2/16$, the program can only remember $O(n)$ many such a's. So there can be at most $O(n)$ high probability edges.

Hence, to handle the case in Item 2, we observe that these $O(n)$ many high-probability edges only blow up the similarity by $2^{O(n)}$. Since initially similarity $\langle \mathbb{P}_x, \mathbb{P}_{x|v_1 \overset{\frown}{\to} t} \rangle = 2^{-2n}$, and we want to prove that it would increase to $\|\mathbb{P}_{x|v_1 \overset{\frown}{\to} t}\|_2^2 \geq 2^{2\epsilon n} \cdot 2^{-2n}$ with very small probability. As long as the constant hidden by big O is much smaller than ϵ, this blow-up is

negligible. We will pick the correct constants to ensure that this is indeed the case.

C. New Ingredient: Bias Counters

As mentioned in the first case, if a high probability edge a cuts $\mathbb{P}_{x|v_1 \rightrightarrows v}$ unevenly, the similarity might grow a lot. First, we explain how the previous work [12] gets around this issue. Then, we will introduce our new idea. This is the key idea for proving the tight memory lower bound for two passes.

Very-bad edge: To get around this issue, they defined "very-bad edges", which is the high probability edges a that cut $\mathbb{P}_{x|v_1 \rightrightarrows v}$ in a very biased way. Formally, $a \in \mathrm{High}(v)$ is "very-bad" if

$$\sum_{\substack{x' \in X \\ M(a,x')=b}} \mathbb{P}_{x|v_1 \rightrightarrows v}(x') \le 2^{-\sqrt{n}}.$$

- On the one hand, when a high probability edge is not very-bad, it only blows up the similarity at most by a factor of $2^{\sqrt{n}}$.
  If we set the memory bound S to be $O(n^{3/2})$, there can only be $c \cdot \sqrt{n}$ many high probability edges for some constant $0 < c < 1$ (since remembering one needs $\Omega(n)$ memory). Then these $c \cdot \sqrt{n}$ many high probability not-very-bad edges can blow up the potential by at most 2^{cn}. As long as $c \ll \epsilon$, this will be acceptable for our purpose.
- On the other hand, when an edge (a, b) is very-bad, conditioning on we reached this vertex, x is distributed as $\mathbb{P}_{x|v_1 \rightrightarrows v}$. Over the randomness of x, the probability that we traverse this edge (a, b) instead of $(a, -b)$ is at most $2^{-\sqrt{n}}$.
  First of all, note this is completely independent of the target t. So we do not have to union bound over t. Then if we set the sample bound T to be $2^{O(\sqrt{n})}$, we can stop immediately whenever we meet a very bad edge. For each step, we stop with probability $2^{-\sqrt{n}}$, union bound over all T steps, and we can show that the overall stopping probability is still $2^{-\Omega(\sqrt{n})}$.

Therefore, they can prove that for some constant $\epsilon > 0$, any two-pass algorithm with $S \le \epsilon n^{3/2}$ memory and $T \le 2^{\epsilon\sqrt{n}}$ samples cannot succeed with constant probability.

New Idea: Bias Counter: Instead of stopping immediately at biased "very-bad edges", we introduce a counter $\mathrm{cnt}_{\mathrm{bias}}$ to keep track of the accumulated biases: In the second pass, initially, when we were at the starting vertex v_1, we let $\mathrm{cnt}_{\mathrm{bias}} \leftarrow 0$. For any high probability edge (a, b) from current vertex v (satisfying $a \in \mathrm{High}(v)$), we say it is Δ-biased if

$$\sum_{\substack{x' \in X \\ M(a,x')=b}} \mathbb{P}_{x|v_1 \rightrightarrows v}(x') \in [2^{-\Delta-1}, 2^{-\Delta}).$$

Whenever we traverse a Δ-biased edge, we will update our counter (roughly) by

$$\mathrm{cnt}_{\mathrm{bias}} \leftarrow \mathrm{cnt}_{\mathrm{bias}} + \Delta.$$

Note a Δ-biased edge has to be a high probability edge. There can be at most $O(n)$ high probability edges when $S \le n^2/16$. Hence we will make at most $O(n)$ such updates.

- On one hand, when the counter $\mathrm{cnt}_{\mathrm{bias}} \le \epsilon \cdot n$, the high probability edges we have traversed can blow up the similarity by at most $2^{\epsilon n}$.
  This follows almost directly from the definition of Δ: We increase the counter by Δ if and only if we traversed a Δ-biased edge. As we have discussed, such an edge would blow up the similarity by at most 2^{Δ}. Hence in total, they can blow up the similarity by at most $2^{\mathrm{cnt}_{\mathrm{bias}}}$.
- On the other hand, the overall probability that $\mathrm{cnt}_{\mathrm{bias}} > \epsilon \cdot n$ is small. If we stop whenever the counter exceeds the threshold $\epsilon \cdot n$, we can show the overall stopping probability will be small.
  This is because we traverse a Δ-biased edge (a, b) instead of the other edge $(a, -b)$ with probability at most $2^{-\Delta}$ over the randomness of x. To gain some intuition, let us think about two extreme cases:
  - Case 1: Each time, the counter increases a little, e.g., $\Delta \approx 100$. In this case, w.p. $1 - (2^{-100})^{(\epsilon/100)\cdot n} = 1 - 2^{-\epsilon n}$, the counter $\mathrm{cnt}_{\mathrm{bias}}$ is going to increase like this for less than $(\epsilon/100) \cdot n$ times.
    (For $\Delta \ll 100$, since there are at most $O(n)$ updates, as long as the constant hiding by this big-O is much smaller than ϵ, we can ignore these updates, as the total increase due to them is negligible compared to $\epsilon \cdot n$.)
  - Case 2: Each time, the counter increases a lot, i.e., $\Delta = \delta n$ for $0 < \delta < 1$. In this case, w.p. $1 - (2^{-\delta n})^{(\epsilon/\delta)} = 1 - 2^{-\epsilon n}$, the counter $\mathrm{cnt}_{\mathrm{bias}}$ will increase like this for less than $\frac{\epsilon}{\delta}$ times.

  In both cases, the counter $\mathrm{cnt}_{\mathrm{bias}}$ will not overflow with high probability. To make $\mathrm{cnt}_{\mathrm{bias}}$ larger than $\epsilon \cdot n$, it is necessary to have a sufficiently large number of (correspondingly) sufficiently large Δ, and this is very unlikely. This resembles a famous quote:

  > *"You can fool some of the people all of the time, and all of the people some of the time, but you can not fool all of the people all of the time."— Abraham Lincoln*

  Formally, we can show that for each layer $i \in [T]$, the counter overflows at layer i with probability at most $2^{-\epsilon n}$. Together with a union bound over $T \ll 2^{\epsilon n}$ layers, we prove that the overall stopping probability for counter overflow is small.

This allows us to prove that for some constant $\epsilon' > 0$, any two-pass algorithm with $S \le \epsilon' n^2$ memory and $T \le 2^{\epsilon' n}$ samples cannot success with constant probability. This new idea is the key to proving a tight lower bound for two-pass learning.

In our actual proof, we will modify the program so that each vertex v will "remember" a unique counter value $\mathrm{cnt}_{\mathrm{bias}}(v)$.[3]

[3]More details about this modification, the bias counter, and related stopping rules are presented in Section 4 of the full version.

Potential Argument: To implement the idea, we will introduce a new stopping rule: We stop whenever the counter $\mathrm{cnt}_{\mathrm{bias}}$ exceeds $\epsilon \cdot n$. We have to prove that overall, we stop due to this new rule with a small probability.

Unlike previous stopping rules, this new stopping rule is "soft", in the sense that when a rare event of probability $2^{-\Delta}$ happens, it does not stop right away. Instead, it accumulates such rare events and only stops when enough rare events have occurred. Previously, we only need to analyze stopping probability based on the randomness at the current step. But now, we have to look at the computation history and exploit the fact that we stop only when a lot of rare events have happened in history.

This makes it harder to analyze the stopping probability. Our main technical contribution to two-pass is to come up with a potential analysis resolving the issue. Roughly speaking, our potential function is defined as $\Phi \approx 2^{\mathrm{cnt}_{\mathrm{bias}}}$. Initially, the expectation of Φ is 1 (as we have $\mathrm{cnt}_{\mathrm{bias}} = 0$ at the starting vertex). Since in each step, $\mathrm{cnt}_{\mathrm{bias}}$ increases by Δ with probability roughly $2^{-\Delta}$, the expectation of Φ (of the current state v) will be (almost) non-increasing. At the end of the computation, we can use Markov's inequality to bound the probability of $\mathbf{Pr}[\mathrm{cnt}_{\mathrm{bias}} > \epsilon \cdot n]$ by $\frac{1}{2^{\epsilon n}}$.[4]

D. Transfer Lemma

However, the proof still has the last important missing piece. In Section II-C, we argued intuitively that for a Δ-biased edge (a, b), with only $2^{-\Delta}$ probability over the randomness of x, we traverse (a, b) instead of $(a, -b)$. But there is one important subtlety.

a) Informal discussion.: Note we defined the Δ-biased edges w.r.t. the posterior distribution $\mathbb{P}_{x|v_1 \rightleftharpoons v}$. So we actually showed is the following:

> For all starting vertex v_1 of the second pass, after $v_1 \rightleftharpoons v$, the computational path will then traverse a Δ-biased edge from v w.p. at most $2^{-\Delta}$.

But ideally, we want to argue about the two-pass branching program, starting from v_0. The ideal statement will be:

> For the starting vertex v_0 of the first pass, after $v_0 \rightleftharpoons v$, the computational path will then traverse a Δ-biased edge from v w.p. at most $2^{-\Delta}$.

If $v = v_2$ is a vertex in the last layer of the second pass, as explained in Section II-B, we know that $v_0 \rightleftharpoons v_2$ is equivalent to $v_1 \rightleftharpoons v_2$. This is because the second pass remembers the first pass, so $v_1 \rightleftharpoons v_2$ can "certify" that the first pass indeed reaches v_1.

But for stopping rules, it is crucial that we stop or update counters in the middle of the program. So v is not always going to be in the last layer. Suppose v is in the middle of the second layer and v' is its corresponding vertex in the first layer. As explained in Section II-B, we have

$$(v_0 \rightleftharpoons v) = (v_1 \rightleftharpoons v) \wedge (v' \rightleftharpoons v_1).$$

[4]We will give a more detailed overview and more intuitions in Section 5.1 of the full version.

Hence the statement we showed differs from the ideal one. Intuitively, this is because $v_1 \rightleftharpoons v$ can only certify that $v_0 \rightleftharpoons v'$. We need extra arguments for controlling $v' \rightleftharpoons v_1$. This is why we need the transfer lemma, to transfer the statement we showed into the ideal statement we want.

This is the most technical part of [12]. In Section 6 of the full version, we will frame this subtlety as an adaptive issue and give a slightly simplified proof together with a clean interpretation. This perspective enables us to generalize this lemma to multiple passes.

Distribution Mismatch: To explain this subtlety in full clarity, let us carefully examine the definition of being Δ-biased. We say that a high probability edge (a, b) is Δ-biased if,

$$\sum_{\substack{x' \in X \\ M(a,x')=b}} \mathbb{P}_{x|v_1 \rightleftharpoons v}(x') \leq [2^{-\Delta-1}, 2^{-\Delta}).$$

Note that the vector x is sampled conditioning on the event $v_1 \rightleftharpoons v$. Hence it corresponds to the following random process:

- First, sample a uniformly random $x \in X$ and let $v = v_1$.
- At the i-th step uniformly sample $a_i \in A$ and move v along the edge $(a_i, M(a_i, x))$.

The definition of Δ-biased edges is saying that conditioning on this process reaches v, the probability that $M(a, x) = b$ is at most $2^{-\Delta}$. However, when we talk about our overall stopping probability, we are referring to a different random process:

- First, sample a random $x \sim \mathbb{P}_{x|v_0 \to v_1}$ and let $v = v_1$.
- At the i-th step uniformly sample $a_i \sim \mathbf{Pr}[a_i \mid v_0 \to v]$ and move v along edge $(a_i, M(a_i, x))$.

We can see there is a distribution mismatch between these two. Conditioning on we reached v, in the first process, the posterior distribution of x is $\mathbb{P}_{x|v_1 \rightleftharpoons v}$, while in the second process, the posterior distribution of x is $\mathbb{P}_{x|v_0 \rightleftharpoons v}$. As explained in Section II-B, in general for a vertex v, $(v_0 \rightleftharpoons v) = (v_1 \rightleftharpoons v) \wedge (v' \rightleftharpoons v_1)$. The posterior distribution at the end of the second process

$$\mathbb{P}_{x|v_0 \rightleftharpoons v} = \mathbb{P}_{x|(v_1 \rightleftharpoons v) \wedge (v' \rightleftharpoons v_1)}$$

which is not only conditioning on (1) the event that the path from v_1 reaches v but also on (2) the event that in the first pass, the path from v' picks v_1 as the end vertex.

The issue is that we ultimately want to prove that the stopping probability is small in the second process $v_0 \rightleftharpoons v$. But as Δ-bias has been defined w.r.t. the first process, our previous argument only shows that the stopping probability is small in the first process $v_1 \rightleftharpoons v$.

Transfer Lemma: To resolve this, [12] introduced the following lemma, "transferring" an upper bound on the stopping probability of the first process $v_1 \rightleftharpoons v$ to an upper bound on the stopping probability of the second process $v_0 \rightleftharpoons v$.

Informal Statement: For any two-pass algorithm, suppose for all fixed starting vertex v_1, the computational path from v_1 stops with probability less than 2^{-cn} for some $0 < c < 1$. The computational path from v_0 will stop in the second pass with probability less than $2^{-(c-\epsilon)n}$ for some $0 < \epsilon < 1$.

Our Perspective: Note the assumption of this lemma says that *for all fixed* v_1, over the randomness of x and $a_1, a_2, \ldots, a_T$, the probability of stopping is small. The reason there is this distribution mismatch is that the vertex v_1 is not fixed beforehand but picked by the first pass adaptively (depending on $x, a_1, a_2, \ldots, a_T$ as well). Hence this is really an issue caused by adaptivity.

We will then interpret the proof as exploiting the fact that the first pass (a memory-bounded one-pass algorithm) has a limited ability to adaptively choose the worst v_1. This perspective, which seems missing from the previous works, is crucial for us to generalize this lemma to multiple passes.[5].

E. Extending to Multiple Passes

To extend this to multiple passes, we first adapt our modification on the program and stop rules beyond two passes. This turns out to be simple.[6] Our new counter and its potential analysis extend to multi-pass smoothly as well.[7]

Main Challenge. Transfer Lemma: However, for the transfer lemma, the original proof crucially relies on the fact that the algorithm has only two passes. Subtle technical difficulties arise when generalizing it to multiple passes.[8]

Current Bottleneck. Transfer Lemma: Currently, our lower bound stops at $O(\log \log n)$ many passes. This is because of the transfer lemma. To prove a lower bound for q-passes, we will use our result for $(q-1)$-passes in a black-box way. This implicitly applies the transfer lemma for $q-1$ times. It turns out that each application of the transfer lemma is quite costly: In the informal statement in Section II-D, the original 2^{-cn} bound on stopping probability is demoted to a $2^{-(c-\epsilon)n}$ bound after applying transfer lemma. Roughly speaking, if $\epsilon > 0$ is a constant, we can at most apply this lemma a constant number of times. Then we can only prove a lower bound for a fixed constant number of passes.

By carefully choosing parameters, we can prove that for any algorithm with q passes, to succeed with constant probability, it necessarily requires either $\Omega\left(\frac{n^2}{c_q}\right)$ memory or $2^{\Omega(n/c_q)}$ samples where $c_q = 100^{3^q}$. In particular, this implies a $n^{2-o(1)}$ memory lower bound for $o(\log \log n)$-pass learning. The bottleneck in improving these results beyond $O(\log \log n)$ passes is the successive application of the transfer lemma. It remains an interesting open problem whether a better tradeoff such as $c_q = \exp(q)$ or even $c_q = \mathsf{poly}(q)$ can be proved. For example: can a $n^{o(1)}$-pass algorithm learn the hidden vector x using $n^{2-\Omega(1)}$ memory and $2^{n^{1-\Omega(1)}}$ samples?

[5]A more detailed overview will be given in Section 6 of the full version.

[6]The details are presented in Section 8 of the full version.

[7]This will be included in Section 10.1 of the full version.

[8]We give detailed intuitions and discussions in Section 9 of the full version.

REFERENCES

[1] O. Shamir, "Fundamental limits of online and distributed algorithms for statistical learning and estimation," in *NIPS*, 2014, pp. 163–171.

[2] J. Steinhardt, G. Valiant, and S. Wager, "Memory, communication, and statistical queries," in *COLT*, ser. JMLR Workshop and Conference Proceedings, vol. 49. JMLR.org, 2016, pp. 1490–1516.

[3] R. Raz, "Fast learning requires good memory: A time-space lower bound for parity learning," in *FOCS*. IEEE Computer Society, 2016, pp. 266–275.

[4] G. Kol, R. Raz, and A. Tal, "Time-space hardness of learning sparse parities," in *STOC*. ACM, 2017, pp. 1067–1080.

[5] M. Moshkovitz and N. Tishby, "Mixing complexity and its applications to neural networks," *CoRR*, vol. abs/1703.00729, 2017.

[6] D. Moshkovitz and M. Moshkovitz, "Mixing implies lower bounds for space bounded learning," in *COLT*, ser. Proceedings of Machine Learning Research, vol. 65. PMLR, 2017, pp. 1516–1566.

[7] R. 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)*. IEEE, 2017, pp. 732–742.

[8] D. Moshkovitz and M. Moshkovitz, "Entropy samplers and strong generic lower bounds for space bounded learning," in *ITCS*, ser. LIPIcs, vol. 94. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 28:1–28:20.

[9] S. Garg, R. Raz, and A. Tal, "Extractor-based time-space lower bounds for learning," in *STOC*. ACM, 2018, pp. 990–1002.

[10] P. Beame, S. O. Gharan, and X. Yang, "Time-space tradeoffs for learning finite functions from random evaluations, with applications to polynomials," in *COLT*, ser. Proceedings of Machine Learning Research, vol. 75. PMLR, 2018, pp. 843–856.

[11] Y. Dagan and O. Shamir, "Detecting correlations with little memory and communication," in *COLT*, ser. Proceedings of Machine Learning Research, vol. 75. PMLR, 2018, pp. 1145–1198.

[12] S. Garg, R. Raz, and A. Tal, "Time-space lower bounds for two-pass learning," in *34th Computational Complexity Conference (CCC)*, 2019.

[13] V. Sharan, A. Sidford, and G. Valiant, "Memory-sample tradeoffs for linear regression with small error," in *STOC*. ACM, 2019, pp. 890–901.

[14] Y. Dagan, G. Kur, and O. Shamir, "Space lower bounds for linear prediction in the streaming model," in *COLT*, ser. Proceedings of Machine Learning Research, vol. 99. PMLR, 2019, pp. 929–954.

[15] S. Garg, P. K. Kothari, and R. Raz, "Time-space tradeoffs for distinguishing distributions and applications to security of goldreich's PRG," in *APPROX-RANDOM*, ser. LIPIcs, vol. 176. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 21:1–21:18.

[16] S. Garg, P. K. Kothari, P. Liu, and R. Raz, "Memory-sample lower bounds for learning parity with noise," in *APPROX-RANDOM*, ser. LIPIcs, vol. 207. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 60:1–60:19.

[17] A. Marsden, V. Sharan, A. Sidford, and G. Valiant, "Efficient convex optimization requires superlinear memory," in *COLT*, ser. Proceedings of Machine Learning Research, vol. 178. PMLR, 2022, pp. 2390–2430.

[18] Q. Liu, R. Raz, and W. Zhan, "Memory-sample lower bounds for learning with classical-quantum hybrid memory," *arXiv preprint arXiv:2303.00209*, 2023.

[19] A. Gonen, S. Lovett, and M. Moshkovitz, "Towards a combinatorial characterization of bounded-memory learning," in *NeurIPS*, 2020.

[20] L. Csanky, "Fast parallel matrix inversion algorithms," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 618–623, 1976.

[21] D. A. M. Barrington, "Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1," in *STOC*. ACM, 1986, pp. 1–5.