Learning what to remember

Robi Bhattacharjee RCBHATTA@ENG.UCSD.EDU
University of California, San Diego.
Gaurav Mahajan GMAHAJAN@ENG.UCSD.EDU
University of California, San Diego.

Editors: Sanjoy Dasgupta and Nika Haghtalab

Abstract

We consider a lifelong learning scenario in which a learner faces a neverending and arbitrary stream of facts and has to decide which ones to retain in its limited memory. We introduce a mathematical model based on the online learning framework, in which the learner measures itself against a collection of experts that are also memory-constrained and that reflect different policies for what to remember. Interspersed with the stream of facts are occasional questions, and on each of these the learner incurs a loss if it has not remembered the corresponding fact. Its goal is to do almost as well as the best expert in hindsight, while using roughly the same amount of memory. We identify difficulties with using the multiplicative weights update algorithm in this memory-constrained scenario, and design an alternative scheme whose regret guarantees are close to the best possible.

1. Introduction

A lifelong learning agent—a child learning language, or a robot exploring an environment, or a software program gathering information from the web—experiences a neverending stream of sensory or factual input. It cannot possibly retain all of this information and therefore has to decide what is important to remember. Ultimately, what should be remembered is information that will later be needed; thus the right memory policy might not be clear at the outset but will gradually be revealed by experience.

The problem of *learning what to remember* is a central challenge for lifelong learning systems with bounded memory (Thrun and Mitchell, 1995; Mitchell et al., 2018). In this paper, we present a simple mathematical formalism based on online learning (Cesa-Bianchi and Lugosi, 2006) that captures core aspects of this problem. In our model, at each time step, the learner receives either:

- a *fact*, which can be thought of as a (question, answer) pair, like ("What is the capital of France?", "Paris"), or
- a *question* ("What is the capital of France?").

In the first case, the learner must decide whether or not to store the fact. It is constrained by having only enough memory for M pieces of information. In the second case, the learner incurs a loss if it has not stored the corresponding fact.

The stream of facts and questions is arbitrary and neverending. In choosing a policy for what to remember, the learner has access to N *experts*. These are subject to the same memory bound and could reflect different priorities: for instance, one expert might favor geographical facts, while another might select financial information. The learner's goal is to do almost as well as the best expert in hindsight. That is, at any given time t, the learner should not have incurred too many more errors than the best expert at that time.

We think of an expert as an agent with enough memory for M facts. As each new fact arrives, the agent has the option to store it, but in doing so might need to jettison some previously stored fact.

Some difficulties: A central idea in online learning is to act according to a *weighted majority* of the experts (Littlestone and Warmuth, 1994) and to continually adjust these weights as the experts accumulate different losses. Each of the N experts, say expert e, is given a weight $w(e)$, and this weight is updated at each time step according to a rule of the form: "If expert e makes a mistake, then reduce $w(e)$ (in some manner)."

This general methodology immediately runs into two basic difficulties in our model.

1. There is no easy way for the learner to tell which experts incur a loss at the current time step, because it does not know which facts each expert has in memory. The most obvious way to know this would be to keep a running simulation of the memories of all the experts, but this would require storing MN facts.

2. Even if the learner somehow knows which experts incur a loss at each step, there is no easy way of maintaining the *memory* of the weighted majority of experts: the set of facts that are retained by the majority of the experts. The overall number of such facts can be shown to be at most $2M$, which is not bad, but the problem is that as the weights shift, the composition of this majority-memory also shifts. Tracking these shifts in memory might require suddenly storing facts that appeared in the past, which is not possible in our framework (Theorem 4 has an example).

1.1. Our Contributions

Our starting point towards dealing with these two difficulties is to postpone the first by assuming that the learner has access to an *oracle* that can tell it, for any expert and any fact, whether that expert currently has that fact in memory. We define this oracle formally in Definition 3. We will later do away with this requirement.

The second problem, about tracking the majority-memory, remains. To cope with it, we introduce a different online learning scheme that changes weights *very* infrequently, and in fact only uses two weights, 0 and 1. It uses $2M$ memory and has the following worst-case guarantee: If the loss of the best expert at time t is OPT, the loss of the algorithm at that time is $O(\text{OPT} \log N + M \log N)$.

(Informal) Theorem 1 *Let $\mathcal{E}$ be a set of N experts with M memory and OPT be the number of mistakes made by the best expert in $\mathcal{E}$ by time t. Then there exists an algorithm with access to above mentioned oracle which by time t makes at most $O(OPT \log N + M \log N)$ mistakes using at most $2M$ memory.*

With the second difficulty solved, we return to the first one and remove the need for an oracle. We consider experts of a particular form that we call *value-based*. Such an expert is fully specified by a value function $v : \mathcal{Q} \to \mathbb{N}$ that assigns a score to any given fact. The expert's memory always consists of the M highest-valued facts it has seen so far; when a new fact (q, a) arrives, it decides whether to store this information by simply comparing $v(q)$ to the lowest-valued fact in its memory.

We show that if experts are of this type and the learner is only evaluated on questions taught before (Assumption 4.1), then the learner can very coarsely keep track of the contents of all N of their memories while using just $O(M)$ memory (which would otherwise naively require storing MN questions or their corresponding values), and that this can be combined with the modified online learning algorithm introduced earlier to give similar mistake bounds.

(Informal) Theorem 2 *Let $\mathcal{E}$ be a set of N value-based experts with M memory and OPT denote the number of mistakes made by the best expert in $\mathcal{E}$ by time t. Assume the learner is only evaluated on questions taught before. Then, there exists an algorithm which by time t makes at most $O(OPT \log N + M \log N)$ mistakes using at most $4M$ memory.*

Finally, we demonstrate that an additive term of $O(M \log N)$ in the regret is inevitable in this setting. Our lower bound applies also if the experts are value-based and the adversary satisfies Assumption 4.1.

(Informal) Theorem 3 *There exists a set $\mathcal{E}$ of N value-based experts using M memory with the best expert making at most OPT mistakes such that any algorithm using $O(M)$ memory, with access to above mentioned oracle and only evaluated on questions taught before, makes at least $\Omega(OPT + M \log N)$ mistakes.*

Note that this lower bound in conjunction with our upper bounds show that for algorithms using $O(M)$ memory and with access to experts where the best expert makes no mistake, the mistake bound of $\Theta(M \log N)$ is actually tight.

1.2. Related Work

Our model is a natural extension of online learning (Cesa-Bianchi and Lugosi, 2006). Closest to our setting is Lu and Lu (2011) which considers restricting the memory used by an algorithm to store the weights. To the best of our knowledge, none of the previous work consider the setting where the experts store information. In contrast, our setting is concerned with experts which store "facts" and in turn how many facts (which is different than memory used to store weights considered in previous works) does a algorithm need to store for a reasonable regret. Gramacy et al. (2002) also identifies issues with implementing exponential weights in similar setting but does not provide any performance guarantees.

Previous works Steinhardt et al. (2016); Gonen et al. (2020) consider PAC learning (or SQ learning) with memory constraint where the adversary is restricted to sample from an unknown distribution and a fixed class of hypothesis. There is also a line of work Raz (2016); Garg et al. (2018); Dagan et al. (2019) which prove lower bounds under memory constraints. Our work in comparison is in the more general online learning setting where the adversary is not restricted to a fixed distribution or hypothesis class.

Another line of work considers a non-adversarial sequence prediction setting where the sequence of facts is generated by a underlying model. (Hsu et al., 2012; Anandkumar et al., 2012) uses spectral and tensor methods to learn the distribution generated by Hidden Markov Models by basically learning the parameters for the underlying model. Another strategy proposed in Sharan et al. (2018) is to just remember the last few facts. In contrast, in our framework, we consider the more general setting where the sequences can be adversarially chosen and the above strategies do not work well.

Our setting can be abstractly viewed as a combination of sketching and online learning. In sketching algorithms Morris (1978); Greenwald and Khanna (2001); Charikar et al. (2004), the goal is to compress data to approximately evaluate functions on it using small amount of memory. However naive use of sketching algorithms would lead to suboptimal memory use (quite often a dependence on time T).

Many learning models with explicit notions of memory are used in practice, e.g. recurrent neural networks (Elman, 1990), long short-term memory networks (Hochreiter and Schmidhuber, 1997),

neural Turing machines (Graves et al., 2014), memory networks (Weston et al., 2015), and others. Our work is motivated by these memory-based architectures which basically use their long term memory as a *dynamic* knowledge base and interact with (access or forget) this knowledge selectively using different mechanisms (including forgetting (Sukhbaatar et al., 2021) and attention (Bahdanau et al., 2015)).

2. Setting

Consider a set $\mathcal{Q}$ of all questions and set $\mathcal{A}$ of all answers. Let $\Phi : \mathcal{Q} \to \mathcal{A}$ be an arbitrary function which maps questions to answers. Throughout the paper, we will work with the corresponding set of facts $\mathcal{F} = \{(q, \Phi(q)) : q \in \mathcal{Q}\}$.

We now introduce our learning framework: *Online Question-Answering with Expert Advice*. Our framework can be thought of as a game between a learner and an adversary. The learner is allowed access to a *memory* $\mathcal{M}$ to store facts. It also receives advice from a group of N experts $\mathcal{E}$, each of which has access to its own memory $\mathcal{M}_e$ of size M.

At each time $t = 1, 2, \ldots$

1. An adversary chooses either to teach or evaluate.
2. If adversary chooses to teach,
   (a) the adversary shows a question answer pair $(q^{(t)}, \Phi(q^{(t)}))$ to the learner.
3. Otherwise, if adversary choose to evaluate,
   (a) the adversary picks a question $q^{(t)}$.
   (b) each expert e incurs a cost $c_e^{(t)} = \mathbb{1}\{(q^{(t)}, \Phi(q^{(t)})) \notin \mathcal{M}_e\}$ and the learner incurs a cost $c^{(t)} = \mathbb{1}\{(q^{(t)}, \Phi(q^{(t)})) \notin \mathcal{M}\}$.
4. Each expert e (and learner) updates its memory $\mathcal{M}_e$ (and memory $\mathcal{M}$) by either choosing to store $(q^{(t)}, \Phi(q^{(t)}))$ or ignoring this information. They can also choose to remove any question answer pairs already stored in $\mathcal{M}_e$.

Figure 1: Online Question-Answering with Expert Advice

We now explain the framework (Figure 1) at a high level. At each time step t, the adversary chooses either to teach or evaluate. If the adversary choose to *teach*, it shows a fact $(q^{(t)}, \Phi(q^{(t)}))$ to the learner and all experts. On the other hand, if the adversary chooses to *evaluate*, it chooses a question $q^{(t)}$. If the corresponding fact $(q^{(t)}, \Phi(q^{(t)}))$ is not stored in its memory $\mathcal{M}$ (expert memory $\mathcal{M}_e$), the learner (expert e) incurs a cost of $+1$. In both the scenarios, at this point, the learner (and any expert e) can either choose to store it in the memory $\mathcal{M}$ (and expert memory $\mathcal{M}_e$) or ignore this information. Note that both learner and experts can also choose to *not* persist /remove information from its memory so it can store more important information later on.

Our goal is to find a decision-making algorithm which uses *reasonable* amount of memory and makes least possible mistakes. We now formally define memory and mistakes for an algorithm.

Definition 1 (Memory and Mistakes) *We say a decision-making algorithm uses B memory and makes E mistakes by time T if*

$$|\mathcal{M}| \leq B \text{ for all time } t \in [T] \quad \text{and} \quad \sum_{t=1}^{T} c^{(t)} = E$$

Remark 2 (Auxiliary state) *Note that we primarily care about the number of facts stored by the algorithm and we allow algorithms to use auxiliary state to store some state i.e. weights for different experts etc (for example, multiplicative weights update algorithm uses this auxiliary state to store the number of errors made by each expert). We do not allow storing facts in auxiliary state and all the algorithms in this work use at most $O(N)$ auxillary state.*

One approach for the learner could be to store all the MN facts stored by the N experts at any time t. Even though this approach will make at most the number of mistakes made by the best expert, it uses unreasonable amount of memory.

On the other side of the spectrum, another approach for the learner could be to use multiplicative weights update algorithm (Littlestone and Warmuth, 1994). This approach uses only $2M$ memory! But as discussed in the introduction, there are two main difficulties with designing weighted majority style algorithms in this framework. In the next section, we will use an oracle to circumvent the *first* problem of keeping track of experts memory and solve the *second* problem of tracking the majority-memory.

3. Upper Bound: Unrestricted Expert Access Oracle

Recall that the first difficulty with weighted majority style algorithms is that its unclear how a learner can tell which facts are currently stored in the expert's memory without using MN memory. In this section, we will assume access to an oracle, *Unrestricted Expert Access Oracle*, which allows the learner to check for *any* question if the corresponding fact is stored in an expert's memory.

Definition 3 (Unrestricted Expert Access Oracle) *At any time t, on inputting any expert e and question q, Unrestricted Expert Access Oracle, denoted by $\mathcal{O}_u(e, q)$ returns True if the corresponding fact $(q, \Phi(q))$ is stored in the experts e's memory $\mathcal{M}_e$ and False otherwise.*

This oracle allows us to ignore the first issue for now (which we will solve later in Section 4). We still need to figure out the second difficulty: how to track the ever changing majority-memory. To illustrate this issue, we consider multiplicative weights update algorithm (Algorithm 1) with Unrestricted Expert Access Oracle $\mathcal{O}_u$ and decompose it into two fundamental steps. In the *weight update* step, the algorithm updates the weight assigned to each expert w_e based upon the number of mistakes made by the expert E_e (setting $w_e = (1-\gamma)^{E_e}$ for each expert). Then, in the *memory update* step, it memorizes a fact if and only if the *weighted majority* of experts memories the fact.

In essence, it constructs an implicit expert, the "majority" expert, that always contains in memory precisely the pairs $(q, \Phi(q))$ that at least half of the weighted experts know. Algorithm 1's goal is to then maintain memory that approximates the memory of this implicit expert.

Remark 4 (Difficulty in maintaining majority-memory) *The difficulty with this approach is that changes in the weights $w(e)$ can result in a dramatic change in the majority expert. For example,*

Algorithm 1: Multiplicative Weights Update algorithm

Initialize: $E_e = 0$ for all experts $e \in \mathcal{E}$
for time $t = 1, 2, \ldots$ **do**
  if the adversary chooses to evaluate **then**
    for all expert $e \in \mathcal{E}$ **do**
      Set $E_e = E_e + \mathbb{1}\{(q^{(t)}, \Phi(q^{(t)})) \notin \mathcal{M}_e\}$
      Set $w_e = (1-\gamma)^{E_e}$.
  Experts update their memory $\mathcal{M}_e$ according to their rules.
  Set $\mathcal{M} = \{(q^{(t)}, \Phi(q^{(t)}))\} \cup \mathcal{M}$
  for fact (q, a) in $\mathcal{M}$ **do**
    Let SaveExperts $= \{e : \mathcal{O}_u(e, q) = \text{True}\}$
    if $\sum_{e \in \text{SaveExperts}} w_e < \frac{1}{2} \sum_{e \in \mathcal{E}} w_e$ **then**
      Remove $\{(q, a)\}$ from $\mathcal{M}$.

Weight Update

Memory Update

consider the simple case in which there are only two experts e_1 and e_2 (this is easily extended to N experts) which hold completely different memory. If the majority expert shifts from e_1 to e_2 (say because e_1 made $O(1)$ more mistakes than e_2), this will result in a completely different majority-memory. Although we have the ability to immediately remove all unnecessary facts (corresponding to e_1's memory) from memory, we do not *have the ability to add all necessary facts (facts corresponding to e_2's memory) to memory; we may only memorize a fact when it is presented by the adversary (which causes M mistakes in our example to catch up every time majority changes).*

This difficulty is compounded by the fact that our majority expert can potentially frequently change, as the weights are constantly updated at every time step. Because of this, it is plausible that our algorithm will always be "behind" the majority expert by a significant lag, and this will cause problems for the standard analysis of the algorithm.

To resolve this issue, we propose a modified algorithm which we call the Lazy Weights algorithm (Algorithm 2). The main idea is to modify the *weight update* step to update the weights less often to avoid the problem of a frequently changing majority expert. We do not make any changes to the *memory update* step. Given access to Unrestricted Expert Access Oracle $\mathcal{O}_u$, we show that Algorithm 2 needs to store only $O(M)$ facts to achieve close to best possible mistake bound.

Theorem 1 (Upper Bound: Online learning with infrequent weight changes) *Let $\mathcal{E}$ be a set of N experts with M memory. Let OPT be the number of mistakes of the best expert in $\mathcal{E}$ by time t. Then, by time t, Algorithm 2 with access to Unrestricted Expert Access Oracle $\mathcal{O}_u$ makes at most $6OPT\lceil \log N \rceil + 6M\lceil \log N \rceil$ mistakes using at most $2M$ memory and $O(N)$ auxiliary state.*

We present a proof in Section 6.1. Next we provide a detailed description of Algorithm 2.

3.1. Lazy Weights Update algorithm

Algorithm 2 follows the overall structure of multiplicative weights update algorithm (Algorithm 1): it only modifies the *weight update* step while keeping the rest of the algorithm intact.

It maintains a set of ActiveExperts, which correspond to a "weight" of 1. All other experts are given a weight of 0. Initially, it gives "weight" of 1 to each expert i.e. each expert is in the set

Algorithm 2: Lazy Weights Update algorithm

Set $E_e = 0$ for all experts $e \in \mathcal{E}$
Set ActiveExperts $= \mathcal{E}$
for time $t = 1, 2, \ldots$ **do**
  if the adversary chooses to evaluate **then**
    Set $E_e = E_e + \mathbb{1}\{\mathcal{O}_u(e, q) = \text{False}\}$ for all expert $e \in \mathcal{E}$.
    Set BadExperts $= \{e \in \text{ActiveExperts} : E_e \geq M\}$.
    if $|\text{ActiveExperts}| \leq 3|\text{BadExperts}|$ **then**
      Set ActiveExperts $=$ ActiveExperts $\setminus$ BadExperts
    if ActiveExperts $= \emptyset$ **then**
      Set $E_e = 0$ for all experts $e \in \mathcal{E}$.
      Set ActiveExperts $= \mathcal{E}$
  Experts update their memory $\mathcal{M}_e$ according to their rules.
  Set $\mathcal{M} = \{(q^{(t)}, \Phi(q^{(t)}))\} \cup \mathcal{M}$
  for fact (q, a) in $\mathcal{M}$ **do**
    Let SaveExperts $= \{e : \mathcal{O}_u(e, q) = \text{True}\}$
    if $2|\text{SaveExperts}| < |\text{ActiveExperts}|$ **then**
      Remove $\{(q, a)\}$ from $\mathcal{M}$.

Weight Update

Memory Update

ActiveExperts. At each time t, it updates the number of mistakes made by each expert denoted by E_e (it can track mistakes made by each expert using Unrestricted Expert Access Oracle $\mathcal{O}_u$). Next, it defines the set of candidate experts likely to be removed/de-weighted:

$$\text{BadExperts} = \{e \in \text{ActiveExperts} : E_e \geq M\}$$

If $3|\text{BadExperts}| \geq |\text{ActiveExperts}|$, then it removes BadExperts from ActiveExperts. However, if ActiveExperts ends up being empty, it resets $E_e = 0$ for all experts $e \in \mathcal{E}$ and ActiveExperts $= \mathcal{E}$. Finally, it memorizes a fact if and only if the weighted majority of ActiveExperts memorize the fact (just like the multiplicative weights update algorithm).

4. Upper Bound: Value-based Experts

In this section, we build upon the algorithmic ideas from Section 3 to show how to cope with the first difficulty: *how a learner can tell which facts are currently stored in the expert's memory without using MN memory?*, discussed in the introduction. For this, we consider a particular class of experts, which we call value-based experts.

Definition 5 (Value-based Expert) *We say an expert e is value-based with M memory if there exists an injective function $v_e : \mathcal{Q} \to \mathbb{N}$ such that when shown a sequence of facts $(q_1, \Phi(q_1)), \ldots (q_n, \Phi(q_n))$, it stores the M facts with the largest value $v_e(q_i)$ i.e.*

$$(q_i, \Phi(q_i)) \in \mathcal{M}_e \iff v_e(q_i) \geq \max_M(\{v_e(q_1), v_e(q_2), \ldots, v_e(q_n)\})$$

where $\max_M$ represents the M^{th} largest element in a set. We will also refer to this Mth largest element as the threshold of expert e, denoted $T(e)$.

An example for such an expert is one which priorities geographical facts to be remembered and therefore sets v_e for such facts to be large. Note however that *temporal* experts of the form "store last M facts" can not be represented as a value-based experts.

How does assuming experts to be value-based help with the first difficulty? Recall that for each value-based expert e and value function v_e, there exists a threshold T_e^* such that

$$(q, \Phi(q)) \in \mathcal{M}_e \iff v_e(q) \geq T_e^*.$$

While we do not have access to the thresholds T_e^* (which change over time), we *do* have access to the value function v_e. Thus, if we can *estimate* T_e^* for all experts $e \in \mathcal{E}$. We will denote our estimate as T_e which we will then use to simulate the oracle $\mathcal{O}_u$, which in turn allows us to apply the same methods we used in Algorithm 2. However, the naive way of maintaining thresholds by storing the M largest v_e values for each expert e requires MN auxiliary state. We will show in Algorithm 5, how to use $O(M)$ memory (to store questions) and $O(N)$ auxiliary state (to store thresholds) to maintain approximate lower bounds for the thresholds.

In addition to our assumption about value-based experts, we will need another assumption regarding our adversary, which we will call the *sequential adversary* assumption.

Assumption 4.1 (Sequential Adversary) *We assume that the adversary only evaluates on question q if the fact $(q, \Phi(q))$ has been previously taught. In particular, if the adversary chooses to evaluate on $q^{(t)}$ then there exists $s < t$ with $q^{(s)} = q^{(t)}$ such that time s was a teaching instance.*

We note that this assumption is pretty natural (for example, most exams only evaluate on facts taught before). In fact, it only restricts the adversary from evaluating on facts never shown before, in which case all experts and the learner anyways always make a mistake.

In summary, we make the following two assumptions: (1) all experts $e \in \mathcal{E}$ are value based experts, and we have access to the value functions v_e for all e, (2) the adversary is sequential (Assumption 4.1). Under these assumptions, we will show that Algorithm 5, a modification of Algorithm 2, uses $O(M)$ memory, $O(N)$ auxillary state and makes same number of mistakes.

Theorem 2 (Upper Bound: Value-based Experts) *Let $\mathcal{E}$ be a set of N value based experts with M memory and the adversary satisfies Assumption 4.1. Let OPT denote the number of mistakes made by the best expert in $\mathcal{E}$ by time t. Then, by time t, Algorithm 5 makes at most $6OPT\lceil \log N \rceil + 6M\lceil \log N \rceil$ mistakes using at most $4M$ memory and $O(N)$ auxillary state.*

We provide a complete proof in Section 6.2. Next, we give a detailed description of Algorithm 5 in Section 4.1.

4.1. Value Based Lazy Weights Update algorithm

Algorithm 5 uses the same core ideas as Algorithm 2: we maintain a set of ActiveExperts, which correspond to a "weight" of 1. All other experts are given a weight of 0. These sets are maintained in the same lazy fashion: we remove experts from ActiveExperts only when a significant portion of ActiveExperts have made at least M mistakes.

The main difference in this setting that we do not have access to Unrestricted Expert Access Oracle $\mathcal{O}_u$, and consequently need a way to determine when an expert makes a mistake. We do this

by maintaining estimates T_e of the true thresholds T_e^*. These threshold, T_e are then used to simulate the Unrestricted Expert Access Oracle $\mathcal{O}_u$ by checking if $v_e(q) \geq T_e$ in Line 19.

One significant challenge with this strategy is that T_e can significantly underestimate the threshold. To account for this, we will also require an *additional* estimated threshold, T_e^{pre} which estimates the value of T_e^*, at the latest time before the current time during which the set ActiveExperts was changed.

We first describe the subroutine–Algorithm 3 for maintaining T_e^{pre}. Here, we store a set, MinorMistakes, which maintains an estimate of the set of "minor mistakes" , i.e. questions where (1) the algorithm makes a mistake and (2) strictly less than half of ActiveExperts make a mistake (according to our estimated thresholds T_e). We then set T_e^{pre} to be $\max_M\{v_e(q) : q \in \text{MinorMistakes}\}$ and correspondingly update the mistakes count for experts.

Algorithm 3: UpdatePreThreshold($q^{(t)}$)

1 MinorMistakes = MinorMistakes $\cup\, q^{(t)}$
2 **for** e in ActiveExperts **do**
3 $\quad x \leftarrow \max_M\{v_e(q) : q \in \text{MinorMistakes}\}$.
4 $\quad T_e^{pre} = \max(x, T_e^{pre})$.
5 **for** q in MinorMistakes **do**
6 $\quad$ **if** $v_e(q) < T_e^{pre}$ for at least half of all $e \in$ ActiveExperts **then**
7 $\quad\quad$ Set $E_e = E_e + 1$ for all e with $v_e(q) < T_e^{pre}$.
8 $\quad\quad$ Remove q from MinorMistakes.

Next, we describe the subroutine (Algorithm 4) for maintaining T_e. In Algorithm 4 we set T_e as the Mth largest value of $v_e(q)$ for questions q in $\mathcal{M} \cup$ MinorMistakes.

Algorithm 4: UpdateThreshold

1 **if** $|\mathcal{M} \cup \text{MinorMistakes}| \geq M$ **then**
2 $\quad$ **for** $e \in$ ActiveExperts **do**
3 $\quad\quad x \leftarrow \max_M\{v_e(q) : q \in \mathcal{M} \cup \text{MinorMistakes}\}$;
4 $\quad\quad T_e \leftarrow \max(x, T_e)$;

Other than these subroutines, Algorithm 5 is almost same as Algorithm 2, except that the error counts E_e are managed somewhat differently. Algorithm 2 immediately incremented the error counter E_e by 1 upon realizing that expert e makes a mistake on question $q^{(t)}$. In Algorithm 5, due to the inherent uncertainty in identifying mistakes we implement a more deferred strategy. In Line 10, we increment error counts when at least half of ActiveExperts make a mistake (according to our estimates T_e). Otherwise, we first move the question to MinorMistakes, and only upon removing the question from MinorMistakes do we increment the counters (Line 7 of Algorithm 3).

Algorithm 5: Value Based Lazy Weights Update algorithm

Set $E_e, T_e^{pre}, T_e = 0$ for all experts $e \in \mathcal{E}$
Set ActiveExperts $= \mathcal{E}$
Set MinorMistakes $= \emptyset$
for time $t = 1, 2, \ldots$ **do**
  if the adversary evaluated and $q^{(t)} \notin \mathcal{M}$ **then**
    Set FailedExperts $= \{e \in \text{ActiveExperts} : v_e(q^{(t)}) < T_e\}$.
    if $2|\text{FailedExperts}| < |\text{ActiveExperts}|$ **then**
      Run UpdatePreThreshold($q^{(t)}$)
    else
      Set $E_e = E_e + 1$ for all $e \in$ FailedExperts
    Set BadExperts $= \{e \in \text{ActiveExperts} : E_e \geq M\}$
    if $|\text{ActiveExperts}| \leq 3|\text{BadExperts}|$ **then**
      ActiveExperts $=$ ActiveExperts $\setminus$ BadExperts
    if ActiveExperts $= \emptyset$ **then**
      ActiveExperts $= \mathcal{E}$, $E_e = 0$, for all experts $e \in \mathcal{E}$.
  Run UpdateThreshold
  Set $\mathcal{M} = \{(q^{(t)}, \Phi(q^{(t)}))\} \cup \mathcal{M}$
  for fact (q, a) in $\mathcal{M}$ **do**
    Set SaveExperts $= \{e \in \text{ActiveExperts} : v_e(q) \geq T_e\}$
    if $2|\text{SaveExperts}| < |\text{ActiveExperts}|$ **then**
      Remove $\{(q, a)\}$ from $\mathcal{M}$.

Weight Update

Memory Update

5. Lower Bound

We now discuss the minimum memory any algorithm needs to behave competitively with respect to the best expert. We show that for any algorithm with $O(M)$ memory and arbitrary amount of auxillary state, the additive $\Omega(M \log N)$ mistakes are inevitable.

Theorem 3 (Lower Bound) *Fix c, N, M and OPT to be positive natural numbers. There exists a set $\mathcal{E}$ of N value based experts using M memory with the best expert making at most OPT mistakes and adversary satisfying Assumption 4.1 such that any algorithm A using cM memory and with access to Unrestricted Expert Access Oracle $\mathcal{O}_u$, in the worst case, makes at least $\Omega(OPT + M \log N)$ mistakes.*

We provide a complete proof in Section 6.3. On a high level, the proof basically repeats the following simple strategy (for M memory algorithms): Divide the set of N experts into two groups which remember two different set of M facts. Irrespective of what algorithm chooses to remember, we can always choose a set of questions to evaluate it on, such that algorithm makes $\approx M/2$ mistakes and half of the experts make 0 mistake. Repeating this $\log(N)$ times (recursively on the set of experts which make 0 mistakes) gives the required bound.

6. Proofs

In this section, we provide the proofs for our upper bounds: Theorem 1-2 and lower bound: Theorem 3. Our proof for Theorem 2 builds on the proof of Theorem 1, so we first present its proof.

6.1. Proof for Theorem 1

In this subsection, we will provide a proof and intuition for memory and mistake bound of Theorem 1. We start with proving that Algorithm 2 uses at most $2M$ memory. To prove this, we will use the following helper lemma which helps us analyze the behavior of "majority" expert.

Lemma 6 (Helper Lemma for Majority Expert) *Consider an arbitrary weighting function $w : \mathcal{E} \to \{0, 1\}$. Let D be a set of facts and $b : \mathcal{E} \times \mathcal{F} \to \{0, 1\}$ be a binary function such that for all experts $e \in \mathcal{E}$*

$$\sum_{f \in D} b(e, f) \leq M \tag{1}$$

Then, the following is true

$$\Big|\Big\{f \in D : \sum_{e \in \mathcal{E}} w(e) b(e, f) \geq \frac{1}{2} \sum_{e \in \mathcal{E}} w(e)\Big\}\Big| \leq 2M$$

Proof Let's denote the set above by D' and suppose $|D'| = k$. Then, using Equation (1), we get

$$\begin{aligned}
\frac{k}{2} \sum_{e \in \mathcal{E}} w(e) &\leq \sum_{f \in D'} \sum_{e \in \mathcal{E}} w(e) b(e, f) \\
&= \sum_{e \in \mathcal{E}} w(e) \Big(\sum_{f \in D} b(e, f) \Big) \\
&\leq M \sum_{e \in \mathcal{E}} w(e)
\end{aligned}$$

where the first step follows from definition of D', the second step follows from $D' \subset D$ and the last step follows from Equation (1). ■

Invoking the above lemma for $b(e, f)$ defined as whether expert e stored fact f or not, and weighting $w(e)$ defined as whether expert e is in ActiveExperts or not proves our memory bound. This argument also shows that multiplicative weights update algorithm uses at most $2M$ memory.

Lemma 7 (Memory Bound) *For all time t, $|\mathcal{M}| \leq 2M$.*

Proof Let $b(e, (q', \Phi(q'))) = 1$ if and only if $\mathcal{O}_u(e, q') = True$ and 0 otherwise. Then, because each expert stores at most M facts from $\mathcal{M} \cup \{(q, a)\}$, we get that for all experts $e \in \mathcal{E}$

$$\sum_{f \in \mathcal{M} \cup \{(q,a)\}} b(e, f) \leq M$$

Define weighting $w : \mathcal{E} \to \{0, 1\}$ given by $w(e) = \mathbb{1}\{e \in \text{ActiveExperts}\}$. Then, by the last step in the algorithm at each time t, (q, a) is *not* removed from (persisted in) $\mathcal{M}$ if and only if

$$\sum_{e \in \mathcal{E}} w^{(t)}(e) b(e, f) \geq \frac{1}{2} \sum_{e \in \mathcal{E}} w^{(t)}(e)$$

Therefore, by Lemma 6, the claim follows. ■

Now, we need to show that Algorithm 2 does not make too many mistakes compared to the best expert. We would like to distinguish between the mistakes made by Algorithm 2 based on if majority of the ActiveExperts also made a mistake or not. We define such mistakes as being minor or major mistakes.

Definition 8 (Minor and Major Mistakes) *We partition the mistakes made by the algorithm into:*

1. *A **minor mistake** is a question-time pair (q, t) in which the algorithm makes a mistake and strictly less than half of ActiveExperts make a mistake. This can be thought of a question where algorithm make a mistake, but the implicit majority expert succeeds.*
2. *A **major mistake** is a question-time pair (q, t) in which the algorithm makes a mistake and at least half of ActiveExperts make a mistake. This can be thought of a question where both algorithm and the majority expert make mistakes.*

As we shall see later, number of major mistakes made by Algorithm 2 are much easier to control. Therefore, we first prove our main lemma for controlling minor mistakes made by Algorithm 2. Here we show that Algorithm 2 can only make at most $2M$ minor mistakes between two consecutive ActiveExperts update using our helper lemma (Lemma 6) for analysing majority expert.

Lemma 9 (Minor Mistakes Between Updates) *Let time $t < t'$ be such that ActiveExperts were not updated i.e. Line 8 was not executed between t and t'. Then there are at most $2M$ minor mistakes between times t and t' (inclusive).*

Proof We start by considering the state at time t. Let $S^{(t)}$ denote the subset of facts D shown till time t that at least half of all active experts know and the algorithm does *not* have in memory. First, we have that $|S^{(t)}| \leq 2M$. To see why this is true, let $b(e, (q', \Phi(q'))) = 1$ if and only if $\mathcal{O}_u(e, q') = True \wedge e \in$ ActiveExperts; and 0 otherwise. Then, because each expert stores at most M facts from D, we get that for all experts $e \in \mathcal{E}$

$$\sum_{f \in D} b(e, f) \leq M$$

Define weighting $w : \mathcal{E} \to \{0, 1\}$ given by $w(e) = \mathbb{1}\{e \in \text{ActiveExperts}\}$. Then, by definition of $S^{(t)}$, $(q, a) \in S^{(t)}$ if and only if

$$\sum_{e \in \mathcal{E}} w(e) b(e, f) \geq \frac{1}{2} \sum_{e \in \mathcal{E}} w(e)$$

Therefore, by Lemma 6, $|S^{(t)}| \leq 2M$.

Next, we look at what happens between time t and t'. Observe that every minor fail reduces $S^{(t)}$ by exactly one. This is true, because for every minor mistake, (1) we see a fact f from $S^{(t)}$ by definition and (2) by our algorithm, we memorize the fact when we make a mistake since f is the memory of at least half of the active experts. This completes the proof. ■

Using the bound on minor mistakes between updates of ActiveExperts, we can easily bound the number of total mistakes: minor and major, made by Algorithm 2 in comparison to best expert.

Lemma 10 (Mistake Bound Between Updates) *Let $L^{(t)}$ be the number of mistakes made by the algorithm by time t. Consider times $t < t'$ such that $L^{(t)} + 6M < L^{(t')}$. Then ActiveExperts were updated i.e. Line 8 was executed between time t and t'.*

Proof Assume towards a contradiction that ActiveExperts were **not** updated.. Let A denote the number of ActiveExperts at time t, R denote the number of major mistakes betwen t and t', and X denote the number of BadExperts at time t' i.e. ActiveExperts with $E_e \geq M$ at time t'. Our strategy will be to double count the number of pairs (e, s) where $t \leq s \leq t'$ is a *major mistake*, and e is an expert that got the question at time s incorrect.

We first upper bound this count. For each expert with $E_e < M$, it can clearly be part of at most M pairs. Since, each expert is part of at most R pairs, we get that there are most $XR + (A - X)M$ such pairs.

Next, we can easily lower bound the count. Each major mistake (e, s) has at least $A/2$ experts that get the corresponding question wrong (by definition). Thus, there are at least $AR/2$ such pairs.

Together, the upper and lower bound imply that

$$XR + (A - X)M \geq AR/2.$$

Since, by Lemma 9, we make at most $2M$ minor mistakes, and we must have at least $4M$ major mistakes (since we make at least $6M$ mistakes in total), i.e. $R \geq 4M$. Substituting this above implies $X \geq A/3$, which is in contradiction with algorithm description (Lines 11 to 13) since then ActiveExperts will be updated.

■

Lemma 10 basically means that whenever the algorithm makes $6M$ mistakes, $1/3$rd of the ActiveExperts make M mistakes. Note that this can only happen at most $\log N$ times before every expert must have made at least M mistakes. This immediately proves our main result–Theorem 1.

Proof [Proof of Theorem 1] We first define some notation. We say time t is a **hard reset** if Line 11 of Algorithm 2 is executed at time t i.e. all errors are set again to 0, and ActiveExperts is set to $\mathcal{E}$ at time t.

We claim that: if $t < t'$ are 2 consecutive hard resets, then

$$L^{(t)} + 6M\lceil \log N \rceil \geq L^{(t')} \quad \text{and} \quad \text{OPT}^{(t')} \geq \text{OPT}^{(t)} + M\,.$$

We now prove our theorem using this claim. Let t be any time, and let $0 = t_0 < t_1 < \cdots < t_r < t$ be all the hard resets smaller than t. Applying our first claim, we see that OPT grows by at least M between every t_i, t_{i+1}, implying that

$$\text{OPT}^{(t)} \geq rM\,.$$

On the other hand, the number of mistakes made by the algorithm grows by at most $6M\lceil \log N \rceil$ between every t_i, t_{i+1}. Thus

$$L^{(t)} \leq 6rM\lceil \log N \rceil + 6M\lceil \log N \rceil\,.$$

Substituting our bound on $\text{OPT}^{(t)}$, proves the theorem.

We now prove the claim. Directly before a hard reset, by definition every expert satisfies $E_e \geq M$. This immediately implies that

$$\text{OPT}^{(t')} \geq \text{OPT}^{(t)} + M\,,$$

since E_e was reset to 0 at time t' for all experts $e \in \mathcal{E}$ (by definition of a hard reset) and only incremented by 1 when expert e makes a mistake. Next, by Lemma 10, every $6M$ mistakes by the algorithm corresponds to removing at least $1/3$ of all ActiveExperts. Thus executing this process at most $\lceil \log N \rceil$ times results in the hard reset, meaning that we can have at most $6M\lceil \log N \rceil$ mistakes incurred by the algorithm between t, t' which proves our claim. ■

6.2. Proof for Theorem 2

In this subsection, we provide a proof and intuition for memory and mistake bound of Algorithm 5. The general proof structure of Theorem 2 closely follows that of Theorem 1. We will have the same main steps: bounding the total memory used by the algorithm (Lemma 7), defining minor and major mistakes (Definition 8), bounding the number of minor mistakes between updates (Lemma 9), and bounding the number of total mistakes between updates (Lemma 10).

The key difference is that Algorithm 5 does not get full information of when experts actually make mistakes; it instead has to use its maintained thresholds, T_e, T_e^{pre} to estimate when this happens. This means that the error counts, E_e are not true measures of the number of mistakes each expert makes. To account for this, we will first apply the arguments from Section 4 to show that Algorithm 2 has a similar performance to Algorithm 5 *with respect to its error counts*, E_e. We will then show that the error counts E_e are indeed underestimates of the true error counts, E_e^*, which will then imply the theorem.

Lemma 11 (Memory Bound) *For all times t, $|\mathcal{M}| \leq 2M$.*

Proof We closely adapt the proof of Lemma 7. The only difference here is that we define the binary function b by using the thresholds T_e instead of applying the oracle $\mathcal{O}_u$.

To that end, let $b(e, q) = 1$ if $v_e(q) \geq T_e$ and 0 otherwise. At the end of executing UpdateThreshold, by definition there exist at most M questions $(q, a) \in \mathcal{M}$ such that $v_e(q) \geq T_e$. It follows that for all $e \in \mathcal{E}$,

$$\sum_{(q,a)\in\mathcal{M}} b(e, q) \leq M.$$

Define weighting $w : \mathcal{E} \to \{0, 1\}$ given by $w(e) = \mathbb{1}\{e \in \text{ActiveExperts}\}$. Then, by the last step in the algorithm at each time t, (q, a) is *not* removed from (persisted in) $\mathcal{M}$ if and only if

$$\sum_{e\in\mathcal{E}} w^{(t)}(e)b(e, q) \geq \frac{1}{2}\sum_{e\in\mathcal{E}} w^{(t)}(e)$$

Therefore, by Lemma 6, the claim follows. ■

Next, we give an updated definition of major and minor mistakes.

Definition 12 (Major and Minor Mistakes) *We classify instances in which the algorithm makes mistakes as follows.*

1. *A **minor mistake** is a question q that is stored inside MinorMistakes.*

2. *A **major mistake** is a question q for which the error counters E_e are incremented for at least half of all experts in ActiveExperts. This can either occur in Line 10 of Algorithm 5, or Line 7 of Algorithm 3.*

The main differences in this definition are that they are no longer time specific (i.e. a question answer pair (q, a) is not necessarily classified as a major or minor mistake at the time it is streamed), and that they are defined with respect to actions the algorithm takes (rather than an oracle). Furthermore, minor mistakes are mutable: it is possible for a question q to be considered a minor mistakes at time t but later be converted to a major mistake (Line 7 of Algorithm 3). Thus, for any given time, we define the number of minor mistakes made between updates as the number of elements inside MinorMistakes, which we bound in the following lemma. Note that while the classification of q can potentially change from minor mistake to major mistake, the key idea is that at all times, all mistakes have a precise classification as to whether they are major or minor (which is crucial in the proof of Lemma 14).

Lemma 13 (Minor Mistakes) *$|MinorMistakes| \leq 2M$ at all times.*

Proof The idea here closely follows the proof of Lemma 11. The only difference is that we define b with respect T_e^{pre}, as $b(e, q) = 1$ if $v_e(q) \geq T_e^{pre}$ and 0 otherwise. Since executing UpdatePreThreshold enforces that $\sum_{q \in mistakes} b(e, q) \leq M$ for all experts, the same argument follows. ■

We now bound the total number of mistakes between updates.

Lemma 14 (Mistake Bound Between Updates) *Let $L^{(t)}$ be the number of mistakes made by the algorithm by time t. Consider times $t < t'$ such that $L^{(t)} + 6M < L^{(t')}$. Then ActiveExperts were updated i.e. Line 8 was executed between time t and t'.*

Proof This proof almost identically follows the proof of Lemma 10. Assume towards a contradiction that ActiveExperts were **not** updated. Let A denote the number of ActiveExperts at time t, R denote the number of major mistakes betwen t and t', and X denote the number of BadExperts at time t' i.e. ActiveExperts with $E_e \geq M$ at time t'. Note that we are using Definition 12 for major and minor mistakes.

Our strategy will be to double count the number of pairs (e, q) where q is a major mistake, and e is an expert whose error counter is incremented on behalf of q (i.e. in Line 10 of Algorithm 5, or Line 7 of Algorithm 3.). Similar to Lemma 10, this gives

$$XR + (A - X)M \geq AR/2.$$

Given this equation, we finish the proof by applying the same reasoning as Lemma 10. The only remaining argument is to prove that $R \geq 4M$.

Let q be an arbitrary question that our algorithm makes a mistake on. If at least half of all active experts also make a mistake (based on their estimated thresholds T_e), then q is a major mistake (Line 10 of Algorithm 5) and is *never* considered a minor mistake. If this does not occur, then q is appended to MinorMistakes (Line 1 of Algorithm 3) and is consequently considered a minor mistake during *all times it remains in MinorMistakes*. Finally, if q is removed from MinorMistakes, then it is necessarily considered a major mistake again (Line 7 of Algorithm 3). In summary, while the

classification of q can potentially change from minor mistake to major mistake, the key idea is that at all times, all mistakes have a precise classification as to whether they are major or minor.

Finally, since Lemma 13 implies that at all times the number of minor mistakes is at most $2M$, we must have that $R \geq 4M$ as all other mistakes must be classified as major. ■

By a direct adaptation of the proof of Theorem 1, we have the following corollary.

Corollary 15 *Define a perceived error of an expert e to be any instance in which E_e is incremented by 1. Let $\widetilde{OPT}$ denote the smallest perceived error of any expert. Then Algorithm 5 makes at most $6\ \widetilde{OPT}\ \lceil \log n \rceil + 6M \lceil \log N \rceil$ mistakes using at most $4M$ memory and $O(N)$ auxiliary state.*

Proof This directly follows the proof of Theorem 1. We use the same definition of **hard reset**. Like proof of Theorem 1, if $t < t'$ are 2 consecutive hard resets, then

$$L^{(t)} + 6M\lceil \log N \rceil \geq L^{(t')} \quad \text{and} \quad \widetilde{\text{OPT}}^{(t')} \geq \widetilde{\text{OPT}}^{(t)} + M\,.$$

Let t be any time, and let $0 = t_0 < t_1 < \cdots < t_r < t$ be all the hard resets smaller than t. Like Theorem 1, we get

$$\widetilde{\text{OPT}}^{(t)} \geq rM\,, \quad \text{and} \quad L^{(t)} \leq 6rM\lceil \log N \rceil + 6M\lceil \log N \rceil\,.$$

This proves the mistake bound. Moreover, the only additional memory this algorithm uses are: the set MinorMistakes, and the thresholds T_e, T_e^{pre}. Since there are $O(N)$ thresholds and since $|\text{MinorMistakes}| \leq 2M$ (Lemma 13), the memory bound follows. ■

Finally, to prove Theorem 2, we need to show that $\widetilde{\text{OPT}}$ is an underestimate of the true number of mistakes made by the best expert, OPT.

Proof [Proof of Theorem 2] As we stated above, it suffices to show that $\widetilde{\text{OPT}} \leq \text{OPT}$. For any expert e, let T_e and T_e^{pre} be the values of thresholds at time t while executing Line 10 or Line 7. Define T_e^* to be the true threshold of expert e at that time, and $T_e^{pre,*}$ to be the true threshold of expert e at time s where s was the last time that the set of active experts was updated by our algorithm. Note that such true thresholds must exist because all our experts are assumed to be value based experts (Definition 5).

We claim that $T_e \leq T_e^*$, and $T_e^{pre} \leq T_e^{pre,*}$. These claims finishes the proof as it implies that every increment to error counter E_e corresponds to a instance in which expert e actually made a mistake.

The core idea for both of these claims is that T_e^* or $T_e^{pre,*}$ is Mth largest value of a certain set of questions and every time we update either of T_e or $T_e^{pre,*}$, we are updating them to the Mth largest value of its subset.

First, we prove $T_e \leq T_e^*$. This follows since, T_e is updated in Line 3 of Algorithm 4 to be the Mth largest value of a *subset* of observed questions and T_e^* is defined to be the Mth largest value of *all* observed questions.

The proof for T_e^{pre} is slightly more involved. $T_e^{pre,*}$ is defined to be the Mth largest value of *all* observed questions before time s where s was the last time that the set of active experts was updated by our algorithm. To prove this we need to show that the set used in Line 3, MinorMistakes is a

subset of *all* observed questions before time s or equivalently MinorMistakes does not contain any question that was *first* streamed after time s.

If a question q is added to MinorMistakes, then two things must happen. First, at least half of all experts got the question correct according to the current estimated thresholds T_e. Second, our algorithm must get it wrong. The key observation is that if the question q were streamed for the first time *after* s, then our algorithm would have memorized it. This is because all maintained thresholds are non-decreasing. Thus, since our algorithm got it wrong, q must have been streamed *before* s. ■

6.3. Proof for Theorem 3

In this subsection, we provide the proof for our lower bound–Theorem 3. On a high level, the proof basically repeats the following simple strategy (for M memory algorithms): Divide the set of N experts into two groups which remember two different set of M facts. Irrespective of what algorithm chooses to remember, we can always choose a set of questions to evaluate it on, such that algorithm makes $\approx M/2$ mistakes and half of the experts make 0 mistake. Repeating this $\log(N)$ times (recursively on the set of experts which make 0 mistakes) gives the required bound.

Proof [Proof of Theorem 3] We will divide our sequence of question answer pairs into two parts. We first discuss the first part where any algorithm will make at least $\lfloor \log_{2c} N \rfloor \lfloor M/2 \rfloor$ mistakes and there exists an expert which makes 0 mistakes. In the second part, any algorithm will make at OPT mistakes and the best expert will make at most OPT mistakes.

First part: We will consider $\lfloor \log_{2c} N \rfloor$ collections $C_k = \{(q_{k,j}, \Phi(q_{k,j}))\}_{j=1}^{2cM}$. Note that there are $2cM$ unique question answer pairs in each collection. We will consider the following sequence:

$$C_1, \mathcal{Q}_1, C_2, \mathcal{Q}_2, \ldots, C_{\lfloor \log_{2c} N \rfloor}, \mathcal{Q}_{\lfloor \log_{2c} N \rfloor}$$

where each $\mathcal{Q}_i$ is a set of M questions chosen by adversary that we will define later. We will show that for any algorithm A, there exists a choice of $\mathcal{Q}_i$'s such that the algorithm A will make at least $\lfloor M/2 \rfloor$ mistakes on each $\mathcal{Q}_i$ and there exists a common expert which will make 0 mistakes. Since, we do this for $i = 1$ to $\lfloor \log_{2c} N \rfloor$, we get the desired result.

Next, we define our experts $\mathcal{E}$ which we further divide into $2c$ groups $\mathcal{E}_1, \ldots, \mathcal{E}_{2c}$ each containing $\lfloor N/2c \rfloor$ experts (we throw out the extra experts). Then, each $\mathcal{E}_i$ is further divided into $2c$ groups $\mathcal{E}_{i,1}, \ldots, \mathcal{E}_{i,2c}$. This tree like process is repeated $\lfloor \log_{2c} N \rfloor$ times. Since, all the collections have distinct question answers and each expert (which is not thrown out) is in at least one of the leaf groups, we define the value function for expert w in leaf group $\mathcal{E}_{i_1,i_2,\ldots,i_{\lfloor \log_{2c} N \rfloor}}$ for collection C_k as

$$v_e(q_{k,j}) = \begin{cases} k & \text{if } M(i_k - 1) + 1 \le j \le M i_k \\ 0 & \text{otherwise} \end{cases}$$

Essentially, the experts in $\mathcal{E}_{i_1,i_2,\ldots,i_{\lfloor \log_{2c} N \rfloor}}$ when presented with collection $\{C_1, \ldots, C_k\}$ remembers only M question answers from C_k, in particular $q_{k,j}$ for $j \in [M(i_k - 1), M i_k]$.

Since, the algorithm has only cM memory, and there are in total $2cM$ question answers in each C_k, by pigeonhole principle there exists an $i_k \in [2c]$ such that A remembers less than $\lfloor M/2 \rfloor$ question answers from $\{(q_{k,j}, \Phi(q_{k,j}))\}_{j=M(i_k-1)}^{Mi_k}$. Therefore, if we set $\mathcal{Q}_k = \{(q_{k,j}, \Phi(q_{k,j}))\}_{j=M(i_k-1)}^{Mi_k}$, we get that the algorithm A makes at least $\lfloor \log_{2c} N \rfloor \lfloor M/2 \rfloor$ mistakes. Also, by our setup, the experts in leaf group $\mathcal{E}_{i_1,\ldots,i_{\lfloor \log_{2c} N \rfloor}}$ will make 0 mistakes. This proves the claim.

Second part: We consider the following sequence:

$$(q_1, \Phi(q_1)), \ldots, (q_{cM+1}, \Phi(q_{cM+1})), q$$

where unique question answer pairs $(q_i, \Phi(q_i))$ are shown and then adversary chooses question q to evaluate the experts and algorithm A. Note that by pigeonhole principle, there exists a question answer pair $(q_i, \Phi(q_i))$ such that the algorithm has not stored the answer for question q_i. Choosing $q = q_i$, the algorithm will make 1 mistake and any expert will make at most 1 mistake. Repeating this OPT times proves our claim. ■

References

Animashree Anandkumar, Daniel Hsu, and Sham M. Kakade. A method of moments for mixture models and hidden markov models. In *Conference on Learning Theory (COLT)*, 2012.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 2004.

Yuval Dagan, Gil Kur, and Ohad Shamir. Space lower bounds for linear prediction in the streaming model. In *Conference on Learning Theory (COLT)*, 2019.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 1990.

Sumegha Garg, Ran Raz, and Avishay Tal. Extractor-based time-space lower bounds for learning. In *Symposium on Theory of Computing (STOC)*, 2018.

Alon Gonen, Shachar Lovett, and Michal Moshkovitz. Towards a combinatorial characterization of bounded memory learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Robert B Gramacy, Manfred K. K Warmuth, Scott Brandt, and Ismail Ari. Adaptive caching by refetching. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2002.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint*, 2014.

Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *International Conference on Management of Data (SIGMOD)*, 2001.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences (JCSS)*, 2012.

N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 1994.

Chi-Jen Lu and Wei-Fu Lu. Making online decisions with bounded memory. In *Algorithmic Learning Theory (ALT)*, 2011.

Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bishan Yang, Justin Betteridge, Andrew Carlson, Bhanava Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 2018.

Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 1978.

Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. In *Symposium on Theory of Computing (STOC)*, 2016.

Vatsal Sharan, Sham Kakade, Percy Liang, and Gregory Valiant. Prediction with a short memory. In *Symposium on Theory of Computing (STOC)*, 2018.

Jacob Steinhardt, Gregory Valiant, and Stefan Wager. Memory, communication, and statistical queries. In *Conference on Learning Theory (COLT)*, 2016.

Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. Not all memories are created equal: Learning to forget by expiring. In *International Conference on Machine Learning (ICML)*, 2021.

Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 1995.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015.