A Parameter-free Hedging Algorithm

Kamalika Chaudhuri
ITA, UC San Diego
kamalika@soe.ucsd.edu

Yoav Freund
CSE, UC San Diego
yfreund@ucsd.edu

Daniel Hsu
CSE, UC San Diego
djhsu@cs.ucsd.edu

Abstract

We study the problem of decision-theoretic online learning (DTOL). Motivated by practical applications, we focus on DTOL when the number of actions is very large. Previous algorithms for learning in this framework have a tunable learning rate parameter, and a barrier to using online-learning in practical applications is that it is not understood how to set this parameter optimally, particularly when the number of actions is large.

In this paper, we offer a clean solution by proposing a novel and completely parameter-free algorithm for DTOL. We introduce a new notion of regret, which is more natural for applications with a large number of actions. We show that our algorithm achieves good performance with respect to this new notion of regret; in addition, it also achieves performance close to that of the best bounds achieved by previous algorithms with optimally-tuned parameters, according to previous notions of regret.

1 Introduction

In this paper, we consider the problem of decision-theoretic online learning (DTOL), proposed by Freund and Schapire [1]. DTOL is a variant of the problem of prediction with expert advice [2, 3]. In this problem, a learner must assign probabilities to a fixed set of actions in a sequence of rounds. After each assignment, each action incurs a loss (a value in $[0, 1]$); the learner incurs a loss equal to the expected loss of actions for that round, where the expectation is computed according to the learner's current probability assignment. The *regret* (of the learner) to an action is the difference between the learner's cumulative loss and the cumulative loss of that action. The goal of the learner is to achieve, on any sequence of losses, low regret to the action with the lowest cumulative loss (the best action).

DTOL is a general framework that captures many learning problems of interest. For example, consider tracking the hidden state of an object in a continuous state space from noisy observations [4]. To look at tracking in a DTOL framework, we set each action to be a path (sequence of states) over the state space. The loss of an action at time t is the distance between the observation at time t and the state of the action at time t, and the goal of the learner is to predict a path which has loss close to that of the action with the lowest cumulative loss.

The most popular solution to the DTOL problem is the Hedge algorithm [1, 5]. In Hedge, each action is assigned a probability, which depends on the cumulative loss of this action and a parameter η, also called the *learning rate*. By appropriately setting the learning rate as a function of the iteration [6, 7] and the number of actions, Hedge can achieve a regret upper-bounded by $O(\sqrt{T \ln N})$, for each iteration T, where N is the number of actions. This bound on the regret is optimal as there is a $\Omega(\sqrt{T \ln N})$ lower-bound [5].

In this paper, motivated by practical applications such as tracking, we consider DTOL in the regime where the number of actions N is very large. A major barrier to using online-learning for practical problems is that when N is large, it is not understood how to set the learning rate η. [7, 6] suggest

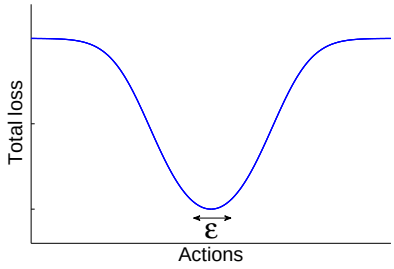

Figure 1: A new notion of regret. Suppose each action is a point on a line, and the total losses are as given in the plot. The regret to the top ϵ-quantile is the difference between the learner's total loss and the total loss of the worst action in the indicated interval of measure ϵ.

setting η as a fixed function of the number of actions N. However, this can lead to poor performance, as we illustrate by an example in Section 3, and the degradation in performance is particularly exacerbated as N grows larger. One way to address this is by simultaneously running multiple copies of Hedge with multiple values of the learning rate, and choosing the output of the copy that performs the best in an online way. However, this solution is impractical for real applications, particularly as N is already very large. (For more details about these solutions, please see Section 4.)

In this paper, we take a step towards making online learning more practical by proposing a novel, completely adaptive algorithm for DTOL. Our algorithm is called NormalHedge. NormalHedge is very simple and easy to implement, and in each round, it simply involves a single line search, followed by an updating of weights for all actions.

A second issue with using online-learning in problems such as tracking, where N is very large, is that the regret to the *best action* is not an effective measure of performance. For problems such as tracking, one expects to have a lot of actions that are close to the action with the lowest loss. As these actions also have low loss, measuring performance with respect to a small group of actions that perform well is extremely reasonable – see, for example, Figure 1.

In this paper, we address this issue by introducing a new notion of regret, which is more natural for practical applications. We order the cumulative losses of all actions from lowest to highest and define the *regret of the learner to the top ϵ-quantile* to be the difference between the cumulative loss of the learner and the $\lfloor \epsilon N \rfloor$-th element in the sorted list.

We prove that for NormalHedge, the regret to the top ϵ-quantile of actions is at most

$$O\left(\sqrt{T \ln \frac{1}{\epsilon}} + \ln^2 N\right),$$

which holds *simultaneously for all T and ϵ*. If we set $\epsilon = 1/N$, we get that the regret to the best action is upper-bounded by $O\left(\sqrt{T \ln N} + \ln^2 N\right)$, which is only slightly worse than the bound achieved by Hedge with optimally-tuned parameters. Notice that in our regret bound, the term involving T has no dependence on N. In contrast, Hedge cannot achieve a regret-bound of this nature uniformly for all ϵ. (For details on how Hedge can be modified to perform with our new notion of regret, see Section 4).

NormalHedge works by assigning each action i a potential; actions which have lower cumulative loss than the algorithm are assigned a potential $\exp(R_{i,t}^2/2c_t)$, where $R_{i,t}$ is the regret of action i and c_t is an adaptive scale parameter, which is adjusted from one round to the next, depending on the loss-sequences. Actions which have higher cumulative loss than the algorithm are assigned potential 1. The weight assigned to an action in each round is then proportional to the derivative of its potential. One can also interpret Hedge as a potential-based algorithm, and under this interpretation, the potential assigned by Hedge to action i is proportional to $\exp(\eta R_{i,t})$. This potential used by Hedge differs significantly from the one we use. Although other potential-based methods have been considered in the context of online learning [8], our potential function is very novel, and to the best

Initially: Set $R_{i,0} = 0$, $p_{i,1} = 1/N$ for each i.
For $t = 1, 2, \ldots$

1. Each action i incurs loss $\ell_{i,t}$.
2. Learner incurs loss $\ell_{A,t} = \sum_{i=1}^{N} p_{i,t}\ell_{i,t}$.
3. Update cumulative regrets: $R_{i,t} = R_{i,t-1} + (\ell_{A,t} - \ell_{i,t})$ for each i.
4. Find $c_t > 0$ satisfying $\frac{1}{N}\sum_{i=1}^{N} \exp\left(\frac{([R_{i,t}]_+)^2}{2c_t}\right) = e$.
5. Update distribution for round $t+1$: $p_{i,t+1} \propto \frac{[R_{i,t}]_+}{c_t} \exp\left(\frac{([R_{i,t}]_+)^2}{2c_t}\right)$ for each i.

Figure 2: The Normal-Hedge algorithm.

of our knowledge, has not been studied in prior work. Our proof techniques are also different from previous potential-based methods.

Another useful property of NormalHedge, which Hedge does not possess, is that it assigns zero weight to any action whose cumulative loss is larger than the cumulative loss of the algorithm itself. In other words, non-zero weights are assigned only to actions which perform better than the algorithm. In most applications, we expect a small set of the actions to perform significantly better than most of the actions. The regret of the algorithm is guaranteed to be small, which means that the algorithm will perform better than most of the actions and thus assign them zero probability.

[9, 10] have proposed more recent solutions to DTOL in which the regret of Hedge to the best action is upper bounded by a function of L, the loss of the best action, or by a function of the variations in the losses. These bounds can be sharper than the bounds with respect to T. Our analysis (and in fact, to our knowledge, any analysis based on potential functions in the style of [11, 8]) do not directly yield these kinds of bounds. We therefore leave open the question of finding an adaptive algorithm for DTOL which has regret upper-bounded by a function that depends on the loss of the best action.

The rest of the paper is organized as follows. In Section 2, we provide NormalHedge. In Section 3, we provide an example that illustrates the suboptimality of standard online learning algorithms, when the parameter is not set properly. In Section 4, we discuss Related Work. In Section 5, we present some outlines of the proof. The proof details are in the Supplementary Materials.

2 Algorithm

2.1 Setting

We consider the decision-theoretic framework for online learning. In this setting, the learner is given access to a set of N actions, where $N \geq 2$. In round t, the learner chooses a weight distribution $p_t = (p_{1,t}, \ldots, p_{N,t})$ over the actions $1, 2, \ldots, N$. Each action i incurs a loss $\ell_{i,t}$, and the learner incurs the expected loss under this distribution:

$$\ell_{A,t} = \sum_{i=1}^{N} p_{i,t}\ell_{i,t}.$$

The learner's instantaneous regret to an action i in round t is $r_{i,t} = \ell_{A,t} - \ell_{i,t}$, and its (cumulative) regret to an action i in the first t rounds is

$$R_{i,t} = \sum_{\tau=1}^{t} r_{i,\tau}.$$

We assume that the losses $\ell_{i,t}$ lie in an interval of length 1 (*e.g.* $[0, 1]$ or $[-1/2, 1/2]$; the sign of the loss does not matter). The goal of the learner is to minimize this cumulative regret $R_{i,t}$ to any action i (in particular, the best action), for any value of t.

2.2 Normal-Hedge

Our algorithm, Normal-Hedge, is based on a potential function reminiscent of the half-normal distribution, specifically

$$\phi(x,c) \;=\; \exp\left(\frac{([x]_+)^2}{2c}\right) \quad \text{for } x \in \mathbb{R}, c > 0 \tag{1}$$

where $[x]_+$ denotes $\max\{0, x\}$. It is easy to check that this function is separately convex in x and c, differentiable, and twice-differentiable except at $x = 0$.

In addition to tracking the cumulative regrets $R_{i,t}$ to each action i after each round t, the algorithm also maintains a scale parameter c_t. This is chosen so that the average of the potential, over all actions i, evaluated at $R_{i,t}$ and c_t, remains constant at e:

$$\frac{1}{N}\sum_{i=1}^{N} \exp\left(\frac{([R_{i,t}]_+)^2}{2c_t}\right) \;=\; e. \tag{2}$$

We observe that since $\phi(x,c)$ is convex in $c > 0$, we can determine c_t with a line search.

The weight assigned to i in round t is set proportional to the first-derivative of the potential, evaluated at $R_{i,t-1}$ and c_{t-1}:

$$p_{i,t} \quad \propto \quad \frac{\partial}{\partial x}\phi(x,c)\bigg|_{x=R_{i,t-1},c=c_{t-1}} \;=\; \frac{[R_{i,t-1}]_+}{c_{t-1}} \exp\left(\frac{([R_{i,t-1}]_+)^2}{2c_{t-1}}\right).$$

Notice that the actions for which $R_{i,t-1} \leq 0$ receive zero weight in round t.

We summarize the learning algorithm in Figure 2.

3 An Illustrative Example

In this section, we present an example to illustrate that setting the parameters of DTOL algorithms as a function of N, the total number of actions, is suboptimal. To do this, we compare the performance of NormalHedge with two representative algorithms: a version of Hedge due to [7], and the Polynomial Weights algorithm, due to [12, 11]. Our experiments with this example indicate that the performance of both these algorithms suffer because of the suboptimal setting of the parameters; on the other hand, NormalHedge automatically adapts to the loss-sequences of the actions.

The main feature of our example is that the effective number of actions n (*i.e.* the number of distinct actions) is smaller than the total number of actions N. Notice that without prior knowledge of the actions and their loss-sequences, one cannot determine the effective number actions in advance; as a result, there is no direct method by which Hedge and Polynomial Weights could set their parameters as a function of n.

Our example attempts to model a practical scenario where one often finds multiple actions with loss-sequences which are almost identical. For example, in the tracking problem, groups of paths which are very close together in the state space, will have very close loss-sequences. Our example indicates that in this case, the performance of Hedge and the Polynomial Weights will depend on the discretization of the state space, however, NormalHedge will comparatively unaffected by such discretization.

Our example has four parameters: N, the total number of actions; n, the effective number of actions (the number of distinct actions); k, the (effective) number of good actions; and ϵ, which indicates how much better the good actions are compared to the rest. Finally, T is the number of rounds.

The instantaneous losses of the N actions are represented by a $N \times T$ matrix $B_N^{\varepsilon,k}$; the loss of action i in round t is the (i,t)-th entry in the matrix. The construction of the matrix is as follows. First, we construct a (preliminary) $n \times T$ matrix A_n based on the $2^d \times 2^d$ Hadamard matrix, where $n = 2^{d+1} - 2$. This matrix A_n is obtained from the $2^d \times 2^d$ Hadamard matrix by (1) deleting the constant row, (2) stacking the remaining rows on top of their negations, (3) repeating each row

horizontally $T/2^d$ times, and finally, (4) halving the first column. We show A_6 for concreteness:

$$A_6 = \left[\begin{array}{cccc|cccc|cccc|c} -1/2 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & \dots \\ -1/2 & -1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 & \dots \\ -1/2 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & \dots \\ +1/2 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & \dots \\ +1/2 & +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 & \dots \\ +1/2 & -1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 & \dots \end{array}\right]$$

If the rows of A_n give the losses for n actions over time, then it is clear that on average, no action is better than any other. Therefore for large enough T, for these losses, a typical algorithm will eventually assign all actions the same weight. Now, let $A_n^{\varepsilon,k}$ be the same as A_n except that ε is subtracted from each entry of the first k rows, *e.g.*

$$A_6^{\varepsilon,2} = \left[\begin{array}{cccc|cccc|c} -1/2-\varepsilon & +1-\varepsilon & -1-\varepsilon & +1-\varepsilon & -1-\varepsilon & +1-\varepsilon & -1-\varepsilon & +1-\varepsilon & \dots \\ -1/2-\varepsilon & -1-\varepsilon & +1-\varepsilon & +1-\varepsilon & -1-\varepsilon & -1-\varepsilon & +1-\varepsilon & +1-\varepsilon & \dots \\ -1/2 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & \dots \\ +1/2 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & \dots \\ +1/2 & +1 & -1 & -1 & +1 & +1 & -1 & -1 & \dots \\ +1/2 & -1 & -1 & +1 & +1 & -1 & -1 & +1 & \dots \end{array}\right].$$

Now, when losses are given by $A_n^{\varepsilon,k}$, the first k actions (the good actions) perform better than the remaining $n-k$; so, for large enough T, a typical algorithm will eventually recognize this and assign the first k actions equal weights (giving little or no weight to the remaining $n-k$). Finally, we artificially replicate each action (each row) N/n times to yield the final loss matrix $B_N^{\varepsilon,k}$ for N actions:

$$B_N^{\varepsilon,k} = \left.\begin{bmatrix} A_n^{\varepsilon,k} \\ A_n^{\varepsilon,k} \\ \vdots \\ A_n^{\varepsilon,k} \end{bmatrix}\right\} N/n \text{ replicates of } A_n^{\varepsilon,k}.$$

The replication of actions significantly affects the behavior of algorithms that set parameters with respect to the number of actions N, which is inflated compared to the effective number of actions n. NormalHedge, having no such parameters, is completely unaffected by the replication of actions.

We compare the performance of NormalHedge to two other representative algorithms, which we call "Exp" and "Poly". Exp is a time/variation-adaptive version of Hedge (exponential weights) due to [7] (roughly, $\eta_t = O(\sqrt{(\log N)/\mathrm{Var}_t})$, where Var_t is the cumulative loss variance). Poly is polynomial weights [12, 11], which has a parameter p that is typically set as a function of the number of actions; we set $p = 2\ln N$ as is recommended to guarantee a regret bound comparable to that of Hedge.

Figure 3 shows the regrets to the best action versus the replication factor N/n, where the effective number of actions n is held fixed. Recall that Exp and Poly have parameters set with respect to the number of actions N.

We see from the figures that NormalHedge is completely unaffected by the replication of actions; no matter how many times the actions may be replicated, the performance of NormalHedge stays exactly the same. In contrast, increasing the replication factor affects the performance of Exp and Poly: Exp and Poly become more sensitive to the changes in the total losses of the actions (*e.g.* the base of the exponent in the weights assigned by Exp increases with N); so when there are multiple good actions (*i.e.* $k > 1$), Exp and Poly are slower to stabilize their weights over these good actions. When $k = 1$, Exp and Poly actually perform better using the inflated value N (as opposed to n), as this causes the slight advantage of the single best action to be magnified. However, this particular case is an anomaly; this does not happen even for $k = 2$. We note that if the parameters of Exp and Poly were set to be a function of n, instead of N, then, then their performance would also not depend on the replication factor (the peformance would be the same as the $N/n = 1$ case). Therefore, the degradation in performance of Exp and Poly is solely due to the suboptimality in setting their parameters.

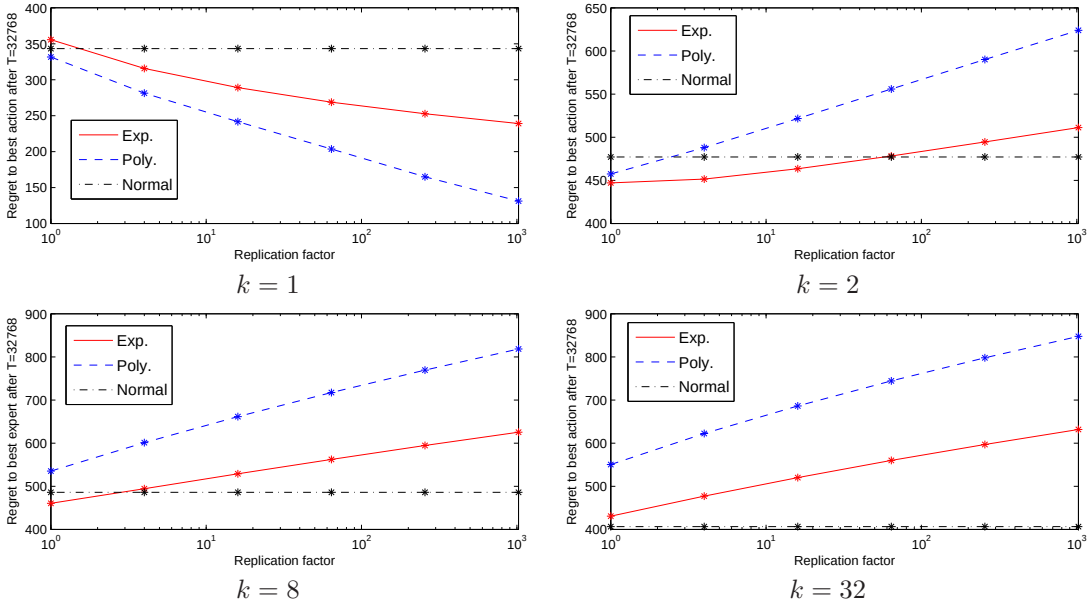

Figure 3: Regrets to the best action after $T = 32768$ rounds, versus replication factor N/n. Recall, k is the (effective) number of good actions. Here, we fix $n = 126$ and $\epsilon = 0.025$.

4 Related work

There has been a large amount of literature on various aspects of DTOL. The Hedge algorithm of [1] belongs to a more general family of algorithms, called the exponential weights algorithms; these are originally based on Littlestone and Warmuth's Weighted Majority algorithm [2], and they have been well-studied.

The standard measure of regret in most of these works is the regret to the best action. The original Hedge algorithm has a regret bound of $O(\sqrt{T \log N})$. Hedge uses a fixed learning rate η for all iterations, and requires one to set η as a function of the total number of iterations T. As a result, its regret bound also holds only for a fixed T. The algorithm of [13] guarantees a regret bound of $O(\sqrt{T \log N})$ to the best action uniformly for all T by using a doubling trick. Time-varying learning rates for exponential weights algorithms were considered in [6]; there, they show that if $\eta_t = \sqrt{8 \ln(N)/t}$, then using exponential weights with $\eta = \eta_t$ in round t guarantees regret bounds of $\sqrt{2T \ln N} + O(\ln N)$ for any T. This bound provides a better regret to the best action than we do. However, this method is still susceptible to poor performance, as illustrated in the example in Section 3. Moreover, they do not consider our notion of regret.

Though not explicitly considered in previous works, the exponential weights algorithms can be partly analyzed with respect to the regret to the top ϵ-quantile. For any *fixed* ϵ, Hedge can be modified by setting η as a function of this ϵ such that the regret to the top ϵ-quantile is at most $O(\sqrt{T \log(1/\epsilon)})$. The problem with this solution is that it requires that *the learning rate to be set as a function of that particular* ϵ (roughly $\eta = \sqrt{(\log 1/\epsilon)/T}$). Therefore, unlike our bound, this bound does not hold uniformly for all ϵ. One way to ensure a bound for all ϵ uniformly is to run $\log N$ copies of Hedge, each with a learning rate set as a function of a different value of ϵ. A final master copy of the Hedge algorithm then looks at the probabilities given by these subordinate copiesto give the final probabilities. However, this procedure adds an additive $O(\sqrt{T \log \log N})$ factor to the regret to the ϵ quantile of actions, for *any* ϵ. More importantly, this procedure is also impractical for real applications, where one might be already working with a large set of actions. In contrast, our solution NormalHedge is clean and simple, and we guarantee a regret bound for all values of ϵ uniformly, without any extra overhead.

More recent work in [14, 7, 10] provide algorithms with significantly improved bounds when the total loss of the best action is small, or when the total variation in the losses is small. These bounds do not explicitly depend on T, and thus can often be sharper than ones that do (including ours). We stress, however, that these methods use a different notion of regret, and their learning rates depend explicitly on N.

Besides exponential weights, another important class of online learning algorithms are the polynomial weights algorithms studied in [12, 11, 8]. These algorithms too require a parameter; this parameter does not depend on the number of rounds T, but depends crucially on the number of actions N. The weight assigned to action i in round t is proportional to $([R_{i,t-1}]_+)^{p-1}$ for some $p > 1$; setting $p = 2 \ln N$ yields regret bounds of the form $\sqrt{2eT(\ln N - 0.5)}$ for any T. Our algorithm and polynomial weights share the feature that zero weight is given to actions that are performing worse than the algorithm, although the degree of this weight sparsity is tied to the performance of the algorithm. Finally, [15] derive a time-adaptive variation of the follow-the-(perturbed) leader algorithm [16, 17] by scaling the perturbations by a parameter that depends on both t and N.

5 Analysis

5.1 Main results

Our main result is the following theorem.

Theorem 1. *If Normal-Hedge has access to N actions, then for all loss sequences, for all t, for all $0 < \epsilon \leq 1$ and for all $0 < \delta \leq 1/2$, the regret of the algorithm to the top ϵ-quantile of the actions is at most*

$$\sqrt{(1 + \ln(1/\epsilon)) \left(3(1 + 50\delta)t + \frac{16 \ln^2 N}{\delta} (\frac{10.2}{\delta^2} + \ln N) \right)} .$$

In particular, with $\epsilon = 1/N$, the regret to the best action is at most

$$\sqrt{(1 + \ln N) \left(3(1 + 50\delta)t + \frac{16 \ln^2 N}{\delta} (\frac{10.2}{\delta^2} + \ln N) \right)} .$$

The value δ in Theorem 1 appears to be an artifact of our analysis; we divide the sequence of rounds into two phases – the length of the first is controlled by the value of δ – and bound the behavior of the algorithm in each phase separately. The following corollary illustrates the performance of our algorithm for large values of t, in which case the effect of this first phase (and the δ in the bound) essentially goes away.

Corollary 2. *If Normal-Hedge has access to N actions, then, as $t \to \infty$, the regret of Normal-Hedge to the top ϵ-quantile of actions approaches an upper bound of*

$$\sqrt{3t(1 + \ln(1/\epsilon)) + o(t)} .$$

In particular, the regret of Normal-Hedge to the best action approaches an upper bound of of

$$\sqrt{3t(1 + \ln N) + o(t)} .$$

The proof of Theorem 1 follows from a combination of Lemmas 3, 4, and 5, and is presented in detail at the end of the current section.

5.2 Regret bounds from the potential equation

The following lemma relates the performance of the algorithm at time t to the scale c_t.

Lemma 3. *At any time t, the regret to the best action can be bounded as*

$$\max_i R_{i,t} \leq \sqrt{2c_t(\ln N + 1)} .$$

Moreover, for any $0 \leq \epsilon \leq 1$ and any t, the regret to the top ϵ-quantile of actions is at most

$$\sqrt{2c_t(\ln(1/\epsilon) + 1)} .$$

Proof. We use E_t to denote the actions that have non-zero weight on iteration t. The first part of the lemma follows from the fact that, for any action $i \in E_t$,

$$\exp\left(\frac{(R_{i,t})^2}{2c_t}\right) \;=\; \exp\left(\frac{([R_{i,t}]_+)^2}{2c_t}\right) \;\leq\; \sum_{i'=1}^{N} \exp\left(\frac{([R_{i',t}]_+)^2}{2c_t}\right) \;\leq\; Ne$$

which implies $R_{i,t} \leq \sqrt{2c_t(\ln N + 1)}$.

For the second part of the lemma, let $R_{i,t}$ denote the regret of our algorithm to the action with the ϵN-th highest regret. Then, the total potential of the actions with regrets greater than or equal to $R_{i,t}$ is at least

$$\epsilon N \exp\left(\frac{([R_{i,t}]_+)^2}{2c_t}\right) \leq Ne$$

from which the second part of the lemma follows. □

5.3 Bounds on the scale c_t and the proof of Theorem 1

In Lemmas 4 and 5, we bound the growth of the scale c_t as a function of the time t.

The main outline of the proof of Theorem 1 is as follows. As c_t increases monotonically with t, we can divide the rounds t into two phases, $t < t_0$ and $t \geq t_0$, where t_0 is the first time such that

$$c_{t_0} \geq \frac{4\ln^2 N}{\delta} + \frac{16\ln N}{\delta^3},$$

for some fixed $\delta \in (0, 1/2)$. We then show bounds on the growth of c_t for each phase separately. Lemma 4 shows that c_t is not too large at the end of the first phase, while Lemma 5 bounds the per-round growth of c_t in the second phase. The proofs of these two lemmas are quite involved, so we defer them to the supplementary appendix.

Lemma 4. *For any time t,*

$$c_{t+1} \leq 2c_t(1 + \ln N) + 3\,.$$

Lemma 5. *Suppose that at some time t_0, $c_{t_0} \geq \frac{4\ln^2 N}{\delta} + \frac{16\ln N}{\delta^3}$, where $0 \leq \delta \leq \frac{1}{2}$ is a constant. Then, for any time $t \geq t_0$,*

$$c_{t+1} - c_t \leq \frac{3}{2}(1 + 49.19\delta)\,.$$

We now combine Lemmas 4 and 5 together with Lemma 3 to prove the main theorem.

Proof of Theorem 1. Let t_0 be the first time at which $c_{t_0} \geq \frac{4\ln^2 N}{\delta} + \frac{16\ln N}{\delta^3}$. Then, from Lemma 4,

$$c_{t_0} \leq 2c_{t_0-1}(1 + \ln N) + 3,$$

which is at most

$$\frac{8\ln^3 N}{\delta} + \frac{34\ln^2 N}{\delta^3} + \frac{32\ln N}{\delta^3} + 3 \leq \frac{8\ln^3 N}{\delta} + \frac{81\ln^2 N}{\delta^3}.$$

The last inequality follows because $N \geq 2$ and $\delta \leq 1/2$. By Lemma 5, we have that for any $t \geq t_0$,

$$c_t \leq \frac{3}{2}(1 + 49.19\delta)(t - t_0) + c_{t_0}.$$

Combining these last two inequalities yields

$$c_t \leq \frac{3}{2}(1 + 49.19\delta)t + \frac{8\ln^3 N}{\delta} + \frac{81\ln^2 N}{\delta^3}.$$

Now the theorem follows by applying Lemma 3. □

References

[1] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[2] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

[3] V. Vovk. A game of prediction witih expert advice. *Journal of Computer and System Sciences*, 56(2):153–173, 1998.

[4] K. Chaudhuri, Y. Freund, and D. Hsu. Tracking using explanation-based modeling, 2009. arXiv:0903.2862.

[5] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[6] P. Auer, N. Cesa-Bianchi, and C. Gentile. Adaptive and self-confident on-line learning algorithms. *Journal of Computer and System Sciences*, 64(1), 2002.

[7] N. Cesa-Bianchi, Y. Mansour, and G. Stoltz. Improved second-order bounds for prediction with expert advice. *Machine Learning*, 66(2–3):321–352, 2007.

[8] N. Cesa-Bianchi and G. Lugosi. Potential-based algorithms in on-line prediction and game theory. *Machine Learning*, 51:239–261, 2003.

[9] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning and Games*. Cambridge University Press, 2006.

[10] E. Hazan and S. Kale. Extracting certainty from uncertainty: Regret bounded by variation in costs. In *COLT*, 2008.

[11] C. Gentile. The robustness of p-norm algorithms. *Machine Learning*, 53(3):265–299, 2003.

[12] A. J. Grove, N. Littlestone, and D. Schuurmans. General convergence results for linear discriminant updates. *Machine Learning*, 43(3):173–210, 2001.

[13] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Hembold, R. E. Schapire, and M. Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, 1997.

[14] R. Yaroshinsky, R. El-Yaniv, , and S. Seiden. How to better use expert advice. *Machine Learning*, 55(3):271–309, 2004.

[15] M. Hutter and J. Poland. Adaptive online prediction by following the perturbed leader. *Journal of Machine Learning Research*, 6:639–660, 2005.

[16] J. Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3:97–139, 1957.

[17] A. Kalai and S. Vempala. Efficient algorithms for the online optimization. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.