

Reinforcement Learning with Immediate Rewards and Linear Hypotheses¹

Naoki Abe,² Alan W. Biermann,³ and Philip M. Long⁴

Abstract. We consider the design and analysis of algorithms that learn from the consequences of their actions with the goal of maximizing their cumulative reward, when

- the consequence of a given action is felt immediately, and
- a linear function, which is unknown a priori, (approximately) relates a feature vector for each action/state pair to the (expected) associated reward.

We focus on two cases, one in which a continuous-valued reward is (approximately) given by applying the unknown linear function, and another in which the probability of receiving the larger of binary-valued rewards is obtained. For these cases we provide bounds on the per-trial regret for our algorithms that go to zero as the number of trials approaches infinity. We also provide lower bounds that show that the rate of convergence is nearly optimal.

Key Words. Computational learning theory, Reinforcement learning, Immediate rewards, Online learning, Online algorithms, Decision theory, Dialogue systems.

1. Introduction. Many important applied problems can be accurately modeled as that of reinforcement learning from immediate rewards, where a function (approximately) relates features associated with a state/action pair and the expectation of the resulting reward. Furthermore, with appropriate features, it is often the case that the reward function can be accurately approximated by a linear function. In this paper we propose and analyze the performance of algorithms for maximizing the cumulative reward in such cases.

One possible application is the task of choosing internet banner advertisements. An interested user can click on an internet banner ad and obtain more information. Since this is evidence of interest in the ad, one goal of an internet ad server might be to display those ads that are likely to yield clicks. (For a more detailed formulation of the ad server problem, we refer the reader to [2].) It is reasonable to suppose that the click probability can be approximated by a linear function of logical combinations of various attributes associated with the users, the environment, and the ads (such as keywords

¹ Naoki Abe was supported in part by the Grant-in-Aid for Scientific Research on Priority Areas (Discovery Science) 1998 of the Ministry of Education, Science, Sports and Culture, Japan. Alan Biermann was supported by ONR Grant N00014-94-1-0938 and NSF Grant IRI9221842. Phil Long was supported by ONR Grant N00014-94-1-0938, National University of Singapore Academic Research Fund Grant RP960625.

² IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA. nabe@us.ibm.com.

³ Department of Computer Science, Duke University, P.O. Box 90129, Durham, NC 27708, USA. awb@cs.duke.edu.

⁴ Genome Institute of Singapore, 1 Science Park Road, #05-01, Singapore 117528, Republic of Singapore. gislongp@nus.edu.sg.

searched by the user, age, sex, the domain, ad genres, etc.); of course, a number of such logical combinations can be thought of as additional attributes [22], [24]. Whenever the server has a slot to display a banner ad, it is to select one from among a number of alternative ads, each of which is associated with an attribute vector. Importantly, these attribute values for the candidate ads can change over time depending on the user and environment attributes. Furthermore, the assumption of immediate reward is justified by the fact that most often the ad placed during a given visit to a web page will have little effect on the consequences of future ad placements, since future ads will often be presented to different users who were unaware of the earlier choice.

A second application is multimodal message generation. Here, a module of a dialogue system takes as input an abstract representation of an idea to convey to the user and must output a way to express that idea. A reasonable goal is to construct messages that are easy for the user to understand. In Appendix A we describe a grammar-based method for generating multimodal messages that we have implemented and integrated into an existing tutoring system [13]. The system adapts to the preferences of the user using reinforcement learning from immediate rewards and a linear model that is motivated by a coarse user model. The features used are extracted from the rules used to generate the message, and our model of the user implies that the time for the user to understand the message is linear in these features. Most of what determines how easy a statement is to understand concerns that statement and not previous statements—again, this motivates the assumption of immediate rewards.

Our analysis considers a worst-case sequence of states; independence assumptions are often not approximately valid for applications, including those that we focus on in this paper. The absence of independence is especially critical in the presence of an exploration/exploitation tradeoff like that encountered in reinforcement learning settings, since an algorithm cannot get relevant information whenever it likes. We measure the quality of algorithms by bounding the worst-case value of regret, which is defined by formalizing in different ways how much less cumulative reward the algorithm obtains than it would have obtained if it did not have to learn.

We prove upper bounds on the total regret of the algorithms we propose, and nearly matching lower bounds on the worst-case regret for any algorithm. In the case in which continuous-valued rewards are exactly determined by an unknown linear function, our upper and lower bounds match to within a small constant factor. In all cases our analysis shows that the per-trial regret goes to zero as the number of trials approaches infinity, and we provide nearly matching upper and lower bounds on the rate of convergence.

Each of our algorithms can be decomposed into a method for estimating the reward of different possible actions, and a method for using these estimates to choose a probability distribution over actions. The rewards are estimated using standard techniques: we maintain a hypothesis for the linear function relating state information with expected reward, and update this hypothesis using the Widrow–Hoff rule [28]. The main novelty in our algorithms is in the relationship between the estimates of the rewards resulting from different actions and the probabilities of choosing them.

An introduction to reinforcement learning from immediate rewards can be found in Chapter 2 of [27]. Worst-case theoretical analysis of this problem was initiated in the apple-tasting model [19], in which an algorithm learned whether to take a specified action given an attribute vector associated with it, and the binary-valued reward for the action

was assumed to be determined by an unknown function of those attributes. Analysis of a worst-case formulation of the bandit problem (see [12]), in which an algorithm must repeatedly choose from a row of slot machines, was carried out in [8] and [3]. A theoretical analysis using independence assumptions of an algorithm for learning a policy computed by a decision list is described in [17]. For other work on reinforcement learning with immediate rewards, we refer the reader to [11], [10], [26], [29], [21], [20], and [27]. Many of these algorithms make incremental adjustments of one sort or another, including to functions for estimating reward as ours does—again, the novelty in our algorithms is in how these estimated rewards are used to decide the probabilities of taking different actions. However, perhaps the main contribution in this work is in the analysis.

The most closely related previous theoretical work that we are aware of is the work on the bandit problem mentioned above. In the bandit problem the algorithm is faced with a row of slot machines (“one-armed bandits”), and must repeatedly decide which slot machine to play. This problem has been heavily studied (see [12]). Auer et al. [7] and Allenberg [3] studied it in a worst-case framework, where it is assumed that an adversary has fixed a sequence of payoffs for each of the slot machines, and where the rewards of those slot machines that are not played at a given time are never seen. They proved bounds that hold in the worst case on the difference between the expected total reward of their algorithm, which does not know the sequences of payoffs ahead of time, and the best total payoff that can be obtained by repeatedly playing a single slot machine.

Reinforcement learning from immediate rewards is like the bandit problem except that the learner is able, before each play, to observe information relevant to the payoff of each slot machine. In return for this assumption, we are able in some cases to bound the expected sum of the differences between the best reward available *during each trial* and the reward obtained by the learning algorithm.

Since the initial publication of this work in preliminary form [14], [25], [1], Auer has made further progress in this line of research [4]–[6]. His work is described in Sections 2.3 and 3.4.

2. Continuous-Valued Rewards. In this section we look at the case in which each alternative has a continuous-valued reward. We begin by examining the case in which the reward for some alternative is obtained by applying a fixed, unknown linear function to its feature vector (the “realizable” [18] case). Then we look at the case in which a linear function only approximately maps feature vectors to rewards.

2.1. The Realizable Case. We assume learning proceeds in trials. In each trial t the learning algorithm must choose from among n alternatives. Before making this choice, it is given feature vectors $\vec{x}_{t,1}, \dots, \vec{x}_{t,n}$, one for each alternative. We refer to the number of features as d . Then, possibly using randomization, it outputs its choice, which will formally be a number between 1 and n , and which we refer to as a_t . To finish the trial, the algorithm receives the reward y_{t,a_t} for its choice. Note that it does not find out the rewards $y_{t,i}$ for alternatives that it did not choose.

We assume that the total number of trials m in the learning process is finite and known to the algorithm ahead of time. This is just to simplify the analysis: as was the

case in [19], if our algorithms replace each dependence of a parameter on m with the same dependence on the trial number t , nearly the same analysis yields almost the same bounds. Details are omitted from this paper.

We assume that there is an unknown coefficient vector $\vec{v} \in \mathbf{R}^d$ such that, for all trials t and alternatives i , $y_{t,i} = \vec{v} \cdot \vec{x}_{t,i}$.

If one designs algorithms and analyzes them using the assumption that the algorithm knows a priori that the length of the feature vectors and the length of the coefficient vector are at most 1, one can apply known techniques to modify the algorithms and their analysis to cope with the case in which these lengths are greater than 1 and they are unknown. Briefly, if these parameters are known, the analysis goes through with a few small and straightforward modifications; as expected, scaling either up by a constant factor results in an increase in the optimal regret by the same constant. One can get around knowledge of these parameters using the usual doubling technique (see, e.g. [16] and [15]): the idea is to behave as if guesses at upper bounds on the parameters hold until those guesses are proved incorrect, and increase those guesses by constant factors when that happens. To avoid uninteresting clutter in our analysis, we assume that the algorithms know a priori that the length of these are at most 1.

We say that $\langle \vec{x}_{t,i} \rangle_{t,i}$ (i.e. the collection of all feature vectors encountered during some run of a learning algorithm) and \vec{v} are *admissible* if all of their lengths are at most 1. For some algorithm A , we refer to $\langle \vec{x}_{t,i} \rangle_{t,i}$ and \vec{v} collectively as a *run* of A .

Our first algorithm is distinguished by the property that each alternative *not* estimated to be the best by the current hypothesis is picked with probability roughly inversely proportional to how much worse it is predicted to be compared with the alternative that appears to be best. We call this algorithm CRW, since it is for optimizing *continuous*-valued rewards in the *realizable* case using how much *worse* alternatives appear to be. Algorithm CRW maintains a hypothesis for the coefficients of the linear function mapping feature vectors to rewards; we refer to the hypothesis on the t th trial by \vec{w}_t .

Algorithm CRW sets $\kappa = \sqrt{mn/2}$, and $\vec{w}_1 = (0, \dots, 0)$. On the t th trial,

- for each alternative i , it uses its current weight vector \vec{w}_t to calculate its estimate $\hat{y}_{t,i} = \vec{w}_t \cdot \vec{x}_{t,i}$ of the reward for alternative i on this trial,
- it sets $g_t = \operatorname{argmax}_i \hat{y}_{t,i}$ to be some alternative that its current hypothesis suggests yields the greatest reward,
- for each alternative i other than the alternative g_t that appears to be the best, it sets the probability $p_{t,i}$ of choosing alternative i to be

$$p_{t,i} = \frac{1}{n + 4\kappa(\hat{y}_{t,g_t} - \hat{y}_{t,i})},$$

- it gives the rest of the probability to g_t , i.e. it sets $p_{t,g_t} = 1 - \sum_{i \neq g_t} p_{t,i}$,
- it chooses a_t randomly according to $p_{t,1}, \dots, p_{t,n}$,
- it receives $y_{t,a_t} \in [-1, 1]$ from the environment (where for all i , $y_{t,i} = \vec{v} \cdot \vec{x}_{t,i}$), and
- it sets $\vec{w}_{t+1} = \vec{w}_t + (y_{t,a_t} - \vec{w}_t \cdot \vec{x}_{t,a_t})\vec{x}_{t,a_t}$.

Following [15], our analysis proceeds by using the squared distance between the hypothesis coefficient vector \vec{w}_t and the target coefficient vector \vec{v} as a “measure of progress”.

Our first progress lemma is known; it follows for example directly from the analysis in [15].

LEMMA 1 [15]. *For any \vec{x} , \vec{w}_{old} , $\vec{x} \in \mathbf{R}^n$, for which $\|\vec{x}\| \leq 1$, if $\vec{w}_{\text{new}} = \vec{w}_{\text{old}} + (\vec{v} \cdot \vec{x} - \vec{w}_{\text{old}} \cdot \vec{x})\vec{x}$, then*

$$\|\vec{w}_{\text{new}} - \vec{v}\|^2 - \|\vec{w}_{\text{old}} - \vec{v}\|^2 \leq -(\vec{w}_{\text{old}} \cdot \vec{x} - \vec{v} \cdot \vec{x})^2.$$

The following theorem is our main result about algorithm CRW.

THEOREM 2. *On any admissible run of algorithm CRW, if $\mathbf{E}(\cdot)$ represents the expectation with respect to all the randomization in the learning process,*

$$\sum_{t=1}^m \max_i y_{t,i} - \mathbf{E} \left(\sum_{t=1}^m y_{t,a_t} \right) \leq \sqrt{2mn}.$$

Note that this provides a bound on the rate at which the average regret per trial goes to zero as the number of trials goes to infinity, since it implies

$$\left(\frac{1}{m} \sum_{t=1}^m \max_i y_{t,i} \right) - \mathbf{E} \left(\frac{1}{m} \sum_{t=1}^m y_{t,a_t} \right) \leq \sqrt{\frac{2n}{m}}.$$

All of the bounds of this paper have similar consequences.

The proof of Theorem 2 makes use of the following lemma.

LEMMA 3. *On any admissible run of algorithm CRW, on any trial t , if $\mathbf{E}(\cdot)$ represents the expectation with respect to the randomization of the algorithm,*

$$\max_i y_{t,i} - \mathbf{E}(y_{t,a_t}) \leq \kappa \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) + \frac{n}{2\kappa}.$$

PROOF. Choose an admissible run of algorithm CRW, and fix some trial t . Let

$$\text{progress} = \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2)$$

and $\text{best} = \max_i y_{t,i}$ and drop the subscript t from all notation. Choose b so that $y_b = \max_i y_i$.

Expanding the definition of progress, and applying Lemma 1 to each term, we get

$$\text{progress} \geq \sum_{i=1}^n p_i (\hat{y}_i - y_i)^2.$$

Thus

$$\begin{aligned} \text{best} - \mathbf{E}(y_a) - \kappa \text{progress} &\leq y_b - \left(\sum_{i=1}^n p_i y_i \right) - \kappa \left(\sum_{i=1}^n p_i (\hat{y}_i - y_i)^2 \right) \\ &= \left(\sum_{i=1}^n p_i (y_b - y_i) \right) - \kappa \left(\sum_{i=1}^n p_i (\hat{y}_i - y_i)^2 \right). \end{aligned}$$

Using calculus, one can see that, for each $i \neq b$, $\sum_{i=1}^n p_i (y_b - y_i - \kappa(\hat{y}_i - y_i))^2$ is maximized, as a function of y_i , when $y_i = \hat{y}_i - 1/2\kappa$. Substituting and simplifying, we get

$$\text{best} - \mathbf{E}(y_a) - \kappa \text{ progress} \leq \left(\sum_{i \neq b} p_i (y_b - \hat{y}_i) \right) + \frac{1 - p_b}{4\kappa} - p_b \kappa (\hat{y}_b - y_b)^2.$$

Again using calculus, one can see that the bound above, as a function of y_b , is maximized when $y_b = \hat{y}_b + (1 - p_b)/(2p_b\kappa)$. Once again substituting and simplifying, we get

$$(1) \quad \text{best} - \mathbf{E}(y_a) - \kappa \text{ progress} \leq \left(\sum_{i \neq b} p_i (\hat{y}_b - \hat{y}_i) \right) + \frac{1 - p_b}{4\kappa} + \frac{(1 - p_b)^2}{4p_b\kappa}.$$

For all $i \in \{1, \dots, n\}$, let $u_i = \hat{y}_g - \hat{y}_i$. From here we divide our analysis into cases, based on whether the greedy alternative is in fact the best alternative. We dispose with the easier case first.

Case 1: $g = b$. Rewriting (1), we get

$$\text{best} - \mathbf{E}(y_a) - \kappa \text{ progress} \leq \left(\sum_{i \neq g} p_i u_i \right) + \frac{1 - p_g}{4\kappa} + \frac{(1 - p_g)^2}{4p_g\kappa}.$$

Since for all $i \neq g$, $p_i \leq 1/n$, we have $p_g \geq 1/n$, which implies

$$\text{best} - \mathbf{E}(y_a) - \kappa \text{ progress} \leq \left(\sum_{i \neq g} p_i u_i \right) + \frac{n - 1}{4\kappa}.$$

Substituting in the definitions of the other p_i 's, we get

$$\begin{aligned} \text{best} - \mathbf{E}(y_a) - \kappa \text{ progress} &\leq \left(\sum_{i \neq g} \frac{u_i}{n + 4\kappa u_i} \right) + \frac{n - 1}{4\kappa} \\ &\leq \left(\sum_{i \neq g} \frac{1}{4\kappa} \right) + \frac{n - 1}{4\kappa} \\ &\leq \frac{n}{2\kappa}, \end{aligned}$$

completing the proof in this case.

Case 2: $g \neq b$. Here (1) implies

$$\begin{aligned} \text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} &\leq \left(\sum_{i \neq b} p_i ((\hat{y}_g - u_b) - \hat{y}_i) \right) + \frac{1 - p_b}{4\kappa} + \frac{(1 - p_b)^2}{4p_b\kappa} \\ &= \left(\sum_{i \neq b} p_i (\hat{y}_g - \hat{y}_i) \right) - (1 - p_b)u_b + \frac{1 - p_b}{4\kappa} \end{aligned}$$

$$\begin{aligned}
& + \frac{(1-p_b)^2}{4p_b\kappa} \\
& = \left(\sum_{i \notin \{b,g\}} p_i u_i \right) - (1-p_b)u_b + \frac{1-p_b}{4\kappa} + \frac{(1-p_b)^2}{4p_b\kappa} \\
& = \left(\sum_{i \neq g} p_i u_i \right) - u_b + \frac{1-p_b}{4\kappa} + \frac{(1-p_b)^2}{4p_b\kappa} \\
& \leq \left(\sum_{i \neq g} p_i u_i \right) - u_b + \frac{1}{4\kappa} + \frac{1}{4p_b\kappa}.
\end{aligned}$$

Substituting into the p_b in the denominator, we get

$$\begin{aligned}
\text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} & \leq \left(\sum_{i \neq g} p_i u_i \right) - u_b + \frac{1}{4\kappa} + \frac{n+4\kappa u_b}{4\kappa} \\
& = \left(\sum_{i \neq g} p_i u_i \right) + \frac{n+1}{4\kappa}.
\end{aligned}$$

Substituting into the remaining p_i 's as in Case 1 completes the proof. \square

PROOF OF THEOREM 2. For each t , let $\text{best}_t = \max_i y_{t,i}$. Applying Lemma 3, on each trial t ,

$$\text{best}_t - \mathbf{E}(y_{t,a_t}) \leq \kappa \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) + \frac{n}{2\kappa},$$

and therefore

$$\begin{aligned}
\sum_{t=1}^m (\text{best}_t - \mathbf{E}(y_{t,a_t})) & \leq \kappa \mathbf{E} \left(\sum_{t=1}^m (\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) \right) + \frac{nm}{2\kappa} \\
& = \kappa \mathbf{E}(\|\vec{w}_1 - \vec{v}\|^2 - \|\vec{w}_{m+1} - \vec{v}\|^2) + \frac{nm}{2\kappa} \\
& \leq \kappa + \frac{nm}{2\kappa}.
\end{aligned}$$

Substituting the value of κ and simplifying completes the proof. \square

Our next algorithm is optimistic when it is ignorant. We call it CRO (since it is for optimizing *continuous*-valued rewards in the *realizable* case by being *optimistic*). On each trial t , CRO behaves as follows:

- If all of $\vec{x}_{t,1}, \dots, \vec{x}_{t,n}$ are in the span of previously seen feature vectors (i.e. $\vec{x}_{1,a_1}, \dots, \vec{x}_{t-1,a_{t-1}}$), then since each reward is linear in the corresponding feature vector, each $y_{t,i}$ can be calculated exactly. CRO does this and chooses the action yielding the largest reward.

- If one of $\vec{x}_{t,1}, \dots, \vec{x}_{t,n}$ is not in the span of previously seen feature vectors, then CRO chooses an arbitrary action corresponding to some such feature vector.

THEOREM 4. *On any admissible run of algorithm CRO,*

$$\sum_{t=1}^m \max_i y_{t,i} - \sum_{t=1}^m y_{t,a_t} \leq 2d.$$

PROOF. On any trial in which all the feature vectors are in the span of previously seen feature vectors, algorithm CRO chooses the best alternative. On all other trials, the regret is at most 2, and the dimension of the space spanned by the chosen feature vectors increases by 1. Since it can only increase d times, this completes the proof. \square

At present, algorithm CRO seems to be only of theoretical interest, as we have not successfully analyzed a noise-tolerant variant of it.

The following lower bound shows that the $O(\min\{m, \sqrt{mn}, d\})$ upper bound obtained by combining Theorems 2 and 4 and observing that the total regret of any algorithm is at most $2m$ is within a constant factor of the best possible. The proof uses ideas from [19].

THEOREM 5. *There is a constant $c > 0$ such that for any algorithm A , any $n, d, m \geq 2$, there is an admissible $\langle \langle \vec{x}_{t,i} \rangle \rangle$ and \vec{v} such that if a_1, \dots, a_m are the (possibly random) actions taken by A ,*

$$\sum_{t=1}^m \max_i y_{t,i} - \mathbf{E} \left(\sum_{t=1}^m y_{t,a_t} \right) \geq c \min\{m, \sqrt{mn}, d\}.$$

The key lemma in proving Theorem 5 is the following. We have optimized for a simple proof at the expense of a slightly weaker bound.

LEMMA 6. *For any algorithm A , for any $n, d, m \geq 2$ such that $2n \leq d \leq mn$, there is an admissible $\langle \langle \vec{x}_{t,i} \rangle \rangle$ and \vec{v} such that if a_1, \dots, a_m are the (possibly random) actions taken by A ,*

$$\sum_{t=1}^m \max_i y_{t,i} - \mathbf{E} \left(\sum_{t=1}^m y_{t,a_t} \right) \geq \frac{1}{2\sqrt{5}} \min \left\{ \frac{mn}{d}, \frac{d}{8} \right\}.$$

PROOF. For each $j \in \{1, \dots, d\}$, let \vec{e}_j be the element of $\{0, 1\}^d$ with a 1 in the j th component, and 0's everywhere else.

Let $\ell = \lceil mn/d \rceil$. We divide the first $\ell \lfloor m/\ell \rfloor$ trials into $\lfloor m/\ell \rfloor$ stages of ℓ trials each. We do not use the remaining $m - \ell \lfloor m/\ell \rfloor$ trials, if any.

For all t , let the feature vector for the first action on each trial be \vec{e}_1 . Set the remaining feature vectors as follows: during each trial of the k th stage, the feature vectors for actions 2 through n are repeatedly $\vec{e}_{(k-1)n+2}, \dots, \vec{e}_{kn}$. (Since $k \leq \lfloor m/\ell \rfloor$, we have $kn \leq \lfloor m/\lceil mn/d \rceil \rfloor n \leq d$.)

Set the reward for the first action on each trial to be $\sqrt{\frac{1}{5}}$. Set the remaining rewards as follows. For each $i \neq 1$, and each stage k , let $q_{i,k}$ be the probability, if the reward for the first action on each trial is $\sqrt{\frac{1}{5}}$ and all other rewards on all other trials are 0, that Algorithm A ever tries action i during stage k . If there is an action i and a stage k for which $q_{i,k} \leq \frac{1}{2}$, then for the least such k , choose such an i , and set the reward for action i to be $2\sqrt{\frac{1}{5}}$ throughout stage k , and set all other rewards to be 0 (we refer to this as “Case 1”). If $q_{i,k} > \frac{1}{2}$ for all actions i and stages k , then set all rewards for actions other than the first to be 0 (this is “Case 2”).

In Case 1, since until algorithm A takes an action other than the first with nonzero reward, its behavior is the same as if there is no such action, with probability at least $\frac{1}{2}$, the action with reward $2\sqrt{\frac{1}{5}}$ is never taken during the stage in which it has that reward. Thus, the total regret during that stage is at least $\sqrt{\frac{1}{5}}\ell$.

In Case 2, since for all stages, all actions other than the first have probability at least $\frac{1}{2}$ to be taken at some time during the stage, the expected total number of times in which actions other than the first are taken is at least $\lfloor m/\ell \rfloor (n-1)/2$, and each of those times the regret is $1/\sqrt{5}$. We have

$$\begin{aligned} \lfloor m/\ell \rfloor (n-1) &= \lfloor m/\lceil mn/d \rceil \rfloor (n-1) \\ &\geq \lfloor d/(2n) \rfloor (n-1) \quad (\text{since } d \leq mn) \\ &\geq d(n-1)/(4n) \quad (\text{since } d \geq 2n) \\ &\geq d/8, \end{aligned}$$

since $n \geq 2$. This completes the proof. \square

PROOF OF THEOREM 5. We divide our analysis into cases.

Case 1: $\min\{m, \sqrt{mn}, d\} = d$ and $d \geq 2n$. Since $d \leq \sqrt{mn}$, we have $mn/d \geq d$. Applying Lemma 6 completes the proof in this case.

Case 2: $\min\{m, \sqrt{mn}, d\} = \sqrt{mn}$. Since $\sqrt{mn} \leq d$, an adversary can set all but $\lceil \sqrt{mn} \rceil$ components of all feature vectors to 0, effectively setting $d = \lceil \sqrt{mn} \rceil$. In that case,

$$\frac{mn}{d} = \frac{mn}{\lceil \sqrt{mn} \rceil} \geq \frac{\sqrt{mn}}{2},$$

since $m, n \geq 2$. Since in this case $d \geq \sqrt{mn}$, we can apply Lemma 6 to complete the proof.

Case 3: $\min\{m, \sqrt{mn}, d\} = m$. In this case $m \leq \sqrt{mn}$ implies $m \leq n$. Also $m \leq d$. The idea is to reduce effectively both n and d to m , and to apply the argument from Lemma 6. Suppose that for all actions $i < m$ and for all trials t , $\vec{x}_{t,i} = \vec{e}_i$, and for all $i \geq m$ and for all trials t , $\vec{x}_{t,i} = \vec{e}_m$. For each $1 < i < m$, let q_i be the probability that algorithm A ever chooses action i if $v_1 = \sqrt{\frac{1}{5}}$ and $v_j = 0$ for all $j \neq 1$; let q_m be the probability that any action $i \geq m$ is ever chosen.

- *Case 3a:* $\min_i q_i \leq \frac{1}{2}$. Choose some j with $1 < j \leq m$ for which $q_j \leq \frac{1}{2}$ and suppose $v_j = 2\sqrt{\frac{1}{5}}$, $v_1 = \sqrt{\frac{1}{5}}$, and all other components of \vec{v} are 0. Since with probability at least $\frac{1}{2}$, no action with reward $2\sqrt{\frac{1}{5}}$ is ever chosen, and at least one such action is available on each trial, the expected total regret in this case is at least $(m/2)(2\sqrt{\frac{1}{5}} - \sqrt{\frac{1}{5}}) = m/(2\sqrt{5})$.
- *Case 3b:* $\min_i q_i > \frac{1}{2}$. Suppose that $v_1 = \sqrt{\frac{1}{5}}$, and for all $j > 1$, $v_j = 0$. In this case the expected number of times that actions other than the first are chosen is at least $(n-1)/2 \geq (m-1)/2$, and therefore the expected total regret is at least $(m-1)/(2\sqrt{5}) \geq m/(4\sqrt{5})$.

Case 4: $d < 2n$. We handle this case in a similar manner as we did Case 3. Here we will effectively reduce n to $\lfloor d/2 \rfloor$ by setting the feature vectors for alternatives $\lfloor d/2 \rfloor$ through n to $\vec{e}_{\lfloor d/2 \rfloor}$. A nearly identical proof to that for Case 3 yields a $\min\{m, \lfloor d/2 \rfloor\}/(4\sqrt{5}) \geq \min\{m, d\}/(16\sqrt{5})$ lower bound for this case, completing the proof. \square

2.2. The Agnostic Case. While examination of the case in which feature vectors and rewards are exactly linearly related can be a useful starting point, a practical algorithm must be able to cope with data in which the relationship is not perfectly linear, either due to the fact that a linear model is only an approximation, or due to noise in the data. In this subsection we theoretically examine algorithms that tolerate nonlinearity. The model of this subsection is the same as the model of Section 2.1, except that instead of assuming that there is a coefficient vector \vec{v} such that $\vec{v} \cdot \vec{x}_{t,i} = y_{t,i}$ for all trials t and actions i , we instead assume that the algorithm is given a parameter η such that there is a coefficient vector \vec{v} of length at most 1 for which

$$(2) \quad \sum_{t=1}^m \sum_{i=1}^n |\vec{v} \cdot \vec{x}_{t,i} - y_{t,i}| \leq \eta.$$

We then bound the expected regret of the algorithm in terms of η as well as m and n . Recall from Section 2.1 that we are assuming that the length of each feature vector $\vec{x}_{t,i}$ is at most 1. We also assume in this section that each reward $y_{t,i}$ is between -1 and 1 (this followed from the other assumptions in Section 2.1). When an algorithm is run on a sequence of feature vectors and rewards satisfying these assumptions, we call it an η -admissible run of the algorithm. Following [23], we refer to this as the agnostic case.

We examine two algorithms for the agnostic case. The first is a modification of the algorithm CRW studied in the realizable case. For obvious reasons, we call the modified algorithm CAW.

It sets $\kappa = \sqrt{2mn/(3+2\eta)}$ and $\vec{w}_1 = (0, \dots, 0)$. On the t th trial, the algorithm

- for each alternative i , sets $\hat{y}_{t,i} = \vec{w}_t \cdot \vec{x}_{t,i}$,
- sets $g_t \in \{1, \dots, n\}$ to be some alternative that maximizes \hat{y}_{t,g_t} ,
- for each alternative i other than g_t , sets

$$p_{t,i} = \frac{1}{n + 3\kappa(\hat{y}_{t,g_t} - \hat{y}_{t,i})/2},$$

- sets $p_{t,g_t} = 1 - \sum_{i \neq g_t} p_{t,i}$,
- chooses a_t randomly according to $p_{t,1}, \dots, p_{t,n}$,
- receives $y_{t,a_t} \in \{0, 1\}$ from the environment, and
- sets $\vec{w}_{t+1} = \vec{w}_t + (y_{t,a_t} - \vec{w}_t \cdot \vec{x}_{t,a_t})\vec{x}_{t,a_t}/2$.

The following refinement of Lemma 1 follows directly⁵ from the analysis in [15]. For future reference, we state it in more generality than we need for the present analysis.

LEMMA 7 [15]. *For any $\vec{v}, \vec{w}_{\text{old}}, \vec{x} \in \mathbf{R}^n, z, \alpha \in \mathbf{R}$ for which $\|\vec{x}\| \leq 1$, and $0 < \alpha \leq \frac{1}{2}$, if $y = \vec{v} \cdot \vec{x}$, and $\vec{w}_{\text{new}} = \vec{w}_{\text{old}} + \alpha(z - \hat{y})\vec{x}$, then*

$$(3) \quad \begin{aligned} \|\vec{w}_{\text{new}} - \vec{v}\|^2 - \|\vec{w}_{\text{old}} - \vec{v}\|^2 \\ \leq -(\alpha - \alpha^2/2)(\vec{w}_{\text{old}} \cdot \vec{x} - z)^2 + (\alpha + \alpha^2/2 + \alpha^3/3)(y - z)^2. \end{aligned}$$

The following theorem is our main result about algorithm CAW.

THEOREM 8. *On any η -admissible run of algorithm CAW, if $\mathbf{E}(\cdot)$ represents the expectation with respect to the randomization of CAW, then*

$$\sum_{t=1}^m \max_i y_{t,i} - \mathbf{E} \left(\sum_{t=1}^m y_{t,a_t} \right) \leq \frac{4}{3} \sqrt{mn(3 + 2\eta)}.$$

The proof of Theorem 8 makes use of the following lemma.

LEMMA 9. *On any η -admissible run of algorithm CAW, on any trial t , if $\mathbf{E}(\cdot)$ represents the expectation with respect to the randomization of the algorithm,*

$$\max_i y_{t,i} - \mathbf{E}(y_{t,a_t}) \leq \kappa \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) + \frac{4n}{3\kappa} + \frac{2}{3}\kappa \sum_{i=1}^n |\vec{v} \cdot x_{t,i} - y_{t,i}|.$$

PROOF. Choose an η -admissible run of algorithm CAW, and fix some trial t . As in the proof of Lemma 3, let progress = $\mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2)$ and best = $\max_i y_{t,i}$ and drop the subscript t from all notation. Choose b so that $y_b = \max_i y_i$.

Applying Lemma 7 with $\alpha = \frac{1}{2}$,

$$\begin{aligned} \text{best} - \mathbf{E}(y_a) - \kappa \text{progress} \\ \leq y_b - \left(\sum_{i=1}^n p_i y_i \right) - \kappa \left(\sum_{i=1}^n p_i (3(\hat{y}_i - y_i)^2/8 - 2(\vec{v} \cdot \vec{x}_i - y_i)^2/3) \right) \end{aligned}$$

⁵ To get Lemma 7 from [15], first apply Lemma IV.3 from that paper with $c = \frac{1}{2}$, $X = 1$, $\beta = \alpha$, $\mathbf{w}_1 = \vec{w}_{\text{old}}$, $\mathbf{w}_2 = \vec{w}_{\text{new}}$, $(\mathbf{w}, \mathbf{x}) = y$, $y = z$, and $\hat{y} = \hat{y}$. Then simplify and exploit the fact that $1/(1-x) = 1+x+x^2 \dots$ for all $0 < x < 1$.

$$\begin{aligned}
&= \left(\sum_{i=1}^n p_i (y_b - y_i) \right) - \kappa \left(\sum_{i=1}^n p_i (3(\hat{y}_i - y_i)^2/8 - 2(\vec{v} \cdot \vec{x}_i - y_i)^2/3) \right) \\
&\leq \left(\sum_{i=1}^n p_i (y_b - y_i) \right) - \kappa \left(\sum_{i=1}^n p_i (3(\hat{y}_i - y_i)^2/8) \right) + (2\kappa/3) \sum_{i=1}^n |\vec{v} \cdot \vec{x}_i - y_i|.
\end{aligned}$$

From here, applying an argument almost identical to that for Lemma 3 to bound the first two terms completes the proof. \square

PROOF OF THEOREM 8. For each t , let $\text{best}_t = \max_i y_{t,i}$ and let $\eta_t = \sum_{i=1}^n |\vec{v} \cdot \vec{x}_{t,i} - y_{t,i}|$. Applying Lemma 9, on each trial t ,

$$\text{best}_t - \mathbf{E}(y_{t,a_t}) \leq \kappa \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) + \frac{4n}{3\kappa} + \frac{2\kappa\eta_t}{3},$$

and therefore

$$\begin{aligned}
\sum_{t=1}^m (\text{best}_t - \mathbf{E}(y_{t,a_t})) &\leq \kappa \mathbf{E} \left(\sum_{t=1}^m (\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) \right) + \frac{4nm}{3\kappa} + \frac{2\kappa\eta}{3} \\
&= \kappa \mathbf{E}(\|\vec{w}_1 - \vec{v}\|^2 - \|\vec{w}_{m+1} - \vec{v}\|^2) + \frac{4nm}{3\kappa} + \frac{2\kappa\eta}{3} \\
&\leq \kappa \left(1 + \frac{2\eta}{3} \right) + \frac{4nm}{3\kappa}.
\end{aligned}$$

Substituting the value of κ and simplifying completes the proof. \square

Next, we describe another algorithm for which we can prove better bounds when the “nonlinearity” η of the data is relatively large. We call this algorithm CAU, since it is for optimizing continuous-valued rewards in the agnostic case while doing its exploration by sampling uniformly. Other differences are that the stepsize of the algorithm’s update to its hypothesis does not depend on the magnitude of its error, but does depend on whether the alternative chosen was the greedy one (a larger stepsize is taken in the less likely case of exploration). This algorithm will make use of the following three parameters:

$$\begin{aligned}
p &= \min \left\{ 1, \frac{n^{2/3}}{m^{1/3}} \right\}, \\
\beta &= \frac{\sqrt{p}}{n\sqrt{m}}, \quad \text{and} \\
\alpha &= \frac{(n-1)\beta(1-p)}{p}.
\end{aligned}$$

It sets $\vec{w}_1 = \vec{0}$. On trial t , algorithm CAU

- after getting $\vec{x}_{t,1}, \dots, \vec{x}_{t,n}$ from the environment, chooses g_t from among $\{1, \dots, n\}$ to maximize $\vec{w}_t \cdot \vec{x}_{t,g_t}$,

- flips a coin with probability p of coming up heads, and
 - if the coin comes up heads, algorithm CAU picks a_t uniformly at random from among $\{1, \dots, n\} - \{g_t\}$, gets y_{t,a_t} from the environment, and sets

$$\vec{w}_{t+1} = \vec{w}_t + \alpha \text{sign}(y_{t,a_t} - \vec{w}_t \cdot \vec{x}_{t,a_t}) \vec{x}_{t,a_t},$$

- if the coin comes up tails, algorithm CAU sets a_t to be g_t , gets y_{t,a_t} from the environment, and sets

$$\vec{w}_{t+1} = \vec{w}_t + \beta \text{sign}(y_{t,a_t} - \vec{w}_t \cdot \vec{x}_{t,a_t}) \vec{x}_{t,a_t}.$$

The proof the following lemma is similar to that of Lemma 7.

LEMMA 10. *Choose $d \in \mathbf{N}$, $\vec{x}, \vec{w}_{\text{old}} \in \mathbf{R}^d$, $\gamma > 0$, and $y \in \mathbf{R}$. Assume $\|\vec{x}\| \leq 1$. Let $\hat{y} = \vec{w}_{\text{old}} \cdot \vec{x}$ and $\vec{w}_{\text{new}} = \vec{w}_{\text{old}} + \gamma \text{sign}(y - \hat{y})\vec{x}$. Then, for any $\vec{v} \in \mathbf{R}^d$,*

$$\|\vec{w}_{\text{new}} - \vec{v}\|_2^2 - \|\vec{w}_{\text{old}} - \vec{v}\|_2^2 \leq -2\gamma|\hat{y} - y| + 2\gamma|\vec{v} \cdot \vec{x} - y| + \gamma^2.$$

PROOF. We have

$$\begin{aligned} & \|\vec{w}_{\text{new}} - \vec{v}\|_2^2 - \|\vec{w}_{\text{old}} - \vec{v}\|_2^2 \\ &= \|\vec{w}_{\text{old}} + \gamma \text{sign}(y - \hat{y})\vec{x} - \vec{v}\|_2^2 - \|\vec{w}_{\text{old}} - \vec{v}\|_2^2 \\ &= ((\vec{w}_{\text{old}} - \vec{v}) + \gamma \text{sign}(y - \hat{y})\vec{x}) \cdot ((\vec{w}_{\text{old}} - \vec{v}) + \gamma \text{sign}(y - \hat{y})\vec{x}) \\ &\quad - \|\vec{w}_{\text{old}} - \vec{v}\|_2^2 \\ &= (\vec{w}_{\text{old}} - \vec{v}) \cdot (\vec{w}_{\text{old}} - \vec{v}) + 2(\vec{w}_{\text{old}} - \vec{v}) \cdot (\gamma \text{sign}(y - \hat{y})\vec{x}) \\ &\quad + (\gamma \text{sign}(y - \hat{y})\vec{x}) \cdot (\gamma \text{sign}(y - \hat{y})\vec{x}) - \|\vec{w}_{\text{old}} - \vec{v}\|_2^2 \\ &= 2\gamma(\hat{y} - \vec{v} \cdot \vec{x}) \text{sign}(y - \hat{y}) + \gamma^2\|\vec{x}\|_2^2 \\ &= 2\gamma(\hat{y} - y + y - \vec{v} \cdot \vec{x}) \text{sign}(y - \hat{y}) + \gamma^2\|\vec{x}\|_2^2 \\ &= -2\gamma|\hat{y} - y| + 2\gamma \text{sign}(y - \hat{y})(y - \vec{v} \cdot \vec{x}) + \gamma^2\|\vec{x}\|_2^2. \end{aligned}$$

Overestimating each of the last two terms completes the proof. \square

THEOREM 11. *On any η -admissible run of algorithm CAU, if $\mathbf{E}(\cdot)$ represents the expectation with respect to the randomization of CAU, then*

$$\sum_{t=1}^m \max_i y_{t,i} - \mathbf{E} \left(\sum_{t=1}^m y_{t,a_t} \right) \leq \eta + 3(nm)^{2/3}.$$

PROOF. For each t , let $\eta_t = \sum_{i=1}^n |\vec{v} \cdot \vec{x}_{t,i} - y_{t,i}|$.

For each $t \in \{1, \dots, m\}$, let $b_t = \operatorname{argmax}_i y_{t,i}$, recall that $g_t = \operatorname{argmax}_i \hat{y}_{t,i}$, and let a_t be the choice made by algorithm CAU on trial t . Then

$$\begin{aligned} \mathbf{E} \left(\sum_{t=1}^m y_{t,b_t} - y_{t,a_t} \right) &= \sum_{t=1}^m \left(\mathbf{E}((1-p)(y_{t,b_t} - y_{t,g_t})) + \frac{p}{n-1} \left(\sum_{i \neq g_t} y_{t,b_t} - y_i \right) \right) \\ &\leq (1-p) \left(\sum_{t=1}^m \mathbf{E}(y_{t,b_t} - y_{t,g_t}) \right) + 2pm. \end{aligned}$$

Since, by the definition of g_t , it is always the case that $\hat{y}_{t,g_t} \geq \hat{y}_{t,b_t}$, we have

$$\begin{aligned} (4) \quad \mathbf{E} \left(\sum_{t=1}^m y_{t,b_t} - y_{t,a_t} \right) &\leq (1-p) \sum_{t=1}^m \mathbf{E}(y_{t,b_t} - \hat{y}_{t,b_t} + \hat{y}_{t,g_t} - y_{t,g_t}) + 2pm \\ &\leq (1-p) \mathbf{E} \left(\sum_{t=1}^m |y_{t,b_t} - \hat{y}_{t,b_t}| + |\hat{y}_{t,g_t} - y_{t,g_t}| \right) + 2pm. \end{aligned}$$

Fix t , and fix the values of the randomization from trials before the t th. By Lemma 10,

$$\begin{aligned} &\mathbf{E}(\|\bar{w}_{t+1} - \bar{v}\|_2^2 - \|\bar{w}_t - \bar{v}\|_2^2) \\ &\leq (1-p)(-2\beta|\hat{y}_{t,g_t} - y_{t,g_t}| + 2\beta|\bar{v} \cdot \bar{x}_{t,g_t} - y_{t,g_t}| + \beta^2) \\ &\quad + \sum_{i \neq g_t} \frac{p}{n-1} (-2\alpha|\hat{y}_{t,i} - y_{t,i}| + 2\alpha|\bar{v} \cdot \bar{x}_{t,i} - y_{t,i}| + \alpha^2) \\ &= (1-p)(-2\beta|\hat{y}_{t,g_t} - y_{t,g_t}| + 2\beta|\bar{v} \cdot \bar{x}_{t,g_t} - y_{t,g_t}| + \beta^2) \\ &\quad - \frac{2\alpha p}{n-1} \left(\sum_{i \neq g_t} |\hat{y}_{t,i} - y_{t,i}| \right) + \frac{2\alpha p}{n-1} \left(\sum_{i \neq g_t} |\bar{v} \cdot \bar{x}_{t,i} - y_{t,i}| \right) + p\alpha^2. \end{aligned}$$

Substituting the definition of α , we have

$$\begin{aligned} &\mathbf{E}(\|\bar{w}_{t+1} - \bar{v}\|_2^2 - \|\bar{w}_t - \bar{v}\|_2^2) \\ &\leq (1-p)(-2\beta|\hat{y}_{t,g_t} - y_{t,g_t}| + 2\beta|\bar{v} \cdot \bar{x}_{t,g_t} - y_{t,g_t}| + \beta^2) \\ &\quad - 2(1-p)\beta \left(\sum_{i \neq g_t} |\hat{y}_{t,i} - y_{t,i}| \right) \\ &\quad + 2(1-p)\beta \left(\sum_{i \neq g_t} |\bar{v} \cdot \bar{x}_{t,i} - y_{t,i}| \right) + (n-1)^2 \beta^2 (1-p)^2 / p. \end{aligned}$$

Collecting terms, we get

$$\begin{aligned} &\mathbf{E}(\|\bar{w}_{t+1} - \bar{v}\|_2^2 - \|\bar{w}_t - \bar{v}\|_2^2) \\ &\leq -2\beta(1-p) \sum_{i=1}^n |\hat{y}_{t,i} - y_{t,i}| + 2\beta(1-p)\eta_t + (1-p)\beta^2 \end{aligned}$$

$$\begin{aligned}
& + (n-1)^2\beta^2(1-p)^2/p \\
& \leq -2\beta(1-p)(|\hat{y}_{t,g_t} - y_{t,g_t}| + |\hat{y}_{t,b_t} - y_{t,b_t}|) + 2\beta(1-p)\eta_t \\
& \quad + (1-p)\beta^2 + (n-1)^2\beta^2(1-p)^2/p.
\end{aligned}$$

Averaging over all of the algorithm's randomization, we get

$$\begin{aligned}
& \mathbf{E}(\|\bar{w}_{t+1} - \bar{v}\|_2^2 - \|\bar{w}_t - \bar{v}\|_2^2) \\
& \leq -2\beta(1-p)\mathbf{E}(|\hat{y}_{t,g_t} - y_{t,g_t}| + |\hat{y}_{t,b_t} - y_{t,b_t}|) + 2\beta(1-p)\eta_t \\
& \quad + (1-p)\beta^2 + (n-1)^2\beta^2(1-p)^2/p.
\end{aligned}$$

Summing over t and telescoping, we get

$$\begin{aligned}
& \sum_{t=1}^m -2\beta(1-p)\mathbf{E}(|\hat{y}_{t,g_t} - y_{t,g_t}| + |\hat{y}_{t,b_t} - y_{t,b_t}|) + (1-p)2\beta\eta_t \\
& \quad + (1-p)\beta^2 + \beta^2(n-1)^2(1-p)^2/p \\
& \geq \mathbf{E}(\|\bar{w}_{m+1} - \bar{v}\|_2^2 - \|\bar{w}_1 - \bar{v}\|_2^2).
\end{aligned}$$

Applying the facts that $\|\bar{v}\|_2 \leq 1$ and $\bar{w}_1 = \bar{0}$, we get

$$\begin{aligned}
& \sum_{t=1}^m -2\beta(1-p)\mathbf{E}(|\hat{y}_{t,g_t} - y_{t,g_t}| + |\hat{y}_{t,b_t} - y_{t,b_t}|) + (1-p)2\beta\eta_t \\
& \quad + (1-p)\beta^2 + \beta^2(n-1)^2(1-p)^2/p \geq -1.
\end{aligned}$$

Solving, we get

$$\begin{aligned}
& \mathbf{E}\left(\sum_{t=1}^m (|\hat{y}_{t,g_t} - y_{t,g_t}| + |\hat{y}_{t,b_t} - y_{t,b_t}|)\right) \\
& \leq \frac{1 + (1-p)2\beta\eta + m((1-p)\beta^2 + \beta^2(n-1)^2(1-p)^2/p)}{2\beta(1-p)}.
\end{aligned}$$

This means, by (4), that

$$\begin{aligned}
& \mathbf{E}\left(\sum_{t=1}^m y_{t,b_t} - y_{t,a_t}\right) \\
& \leq \frac{1 + (1-p)2\beta\eta + m((1-p)\beta^2 + \beta^2(n-1)^2(1-p)^2/p)}{2\beta} + 2pm.
\end{aligned}$$

Simplifying and overapproximating, we get

$$\mathbf{E}\left(\sum_{t=1}^m y_{t,b_t} - y_{t,a_t}\right) \leq \frac{1}{2\beta} + \eta + \frac{\beta mn^2}{2p} + 2pm.$$

Substituting the definition of β yields

$$\mathbf{E} \left(\sum_{t=1}^m y_{t,b_t} - y_{t,a_t} \right) \leq n \sqrt{\frac{m}{p}} + 2pm + \eta.$$

If $m \geq n^2$, then substituting $n^{2/3}/m^{1/3}$ for p we get

$$\mathbf{E} \left(\sum_{t=1}^m y_{t,a_t} - y_{t,b_t} \right) \leq 3(nm)^{2/3} + \eta.$$

If $m \leq n^2$, then, since trivially $\mathbf{E}(\sum_{t=1}^m y_{t,a_t} - y_{t,b_t}) \leq 2m$, we have

$$\mathbf{E} \left(\sum_{t=1}^m y_{t,a_t} - y_{t,b_t} \right) \leq 2m^{1/3}m^{2/3} \leq 2(nm)^{2/3}.$$

This completes the proof. \square

Note that there is a trivial lower bound of η on the best regret guarantee possible for an η -admissible run of any algorithm. Putting this together with Theorem 5, we get a lower bound of $\Omega(\sqrt{mn} + \eta)$ which implies that the $O(\sqrt{mn}(1 + \eta))$ bound of Theorem 8 and the $O((mn)^{2/3} + \eta)$ bound of Theorem 11 cannot be improved much.

2.3. *Improvements.* Putting Theorems 8 and 11 together results in a bound of

$$\min\left\{\frac{4}{3}\sqrt{mn(3 + 2\eta)}/3, \eta + 3(nm)^{2/3}\right\}.$$

Since the publication of preliminary versions of this work, Auer [4] has improved on this bound, obtaining a bound of

$$\min\{4\sqrt{mn}, \eta + 3(3\eta nm)^{1/3}\}$$

for an algorithm that takes $O(dn)$ time per trial, as ours do. By reducing to a problem studied in [9], he established a bound of

$$\eta + c\sqrt{dmn \ln(dm)}$$

for an algorithm that uses $O(m^d)$ time.

3. Random Binary-Valued Rewards. Now we turn to the latter of the two cases studied in this paper: the case in which the *probability* of obtaining the larger of two rewards is given by an unknown linear function, rather than the continuous-valued rewards themselves.

As in Section 2, we assume that learning proceeds in trials, where in each trial t the learning algorithm must choose from among n alternatives. Also, the algorithm is given

feature vectors $\vec{x}_{t,1}, \dots, \vec{x}_{t,n} \in \mathbf{R}^d$, one for each alternative, before making its choice, which we refer to as a_t . At the end of the trial, the algorithm receives z_{t,a_t} , a $\{0, 1\}$ -valued quantity indicating whether this choice resulted in failure (0) or a success (1).

We assume that there is an unknown coefficient vector $\vec{v} \in \mathbf{R}^d$ such that, for all trials t and alternatives i , $\Pr(z_{t,i} = 1) = \vec{v} \cdot \vec{x}_{t,i}$. We make the benign assumption that all $\vec{x}_{t,i}$'s encountered during the learning process have the property that $\vec{v} \cdot \vec{x}_{t,i} \in [0, 1]$. This assumption would be satisfied for example if there was a default probability of success (which can be represented in our framework using a feature that always has the same value) that was adjusted somewhat by the specifics of the feature vectors.

As in Section 2, we assume that m and n are known ahead of time, and that the lengths of the coefficient vector and feature vectors are at most 1. We say that $\langle \vec{x}_{t,i} \rangle_{t,i}$ (i.e. the collection of all feature vectors encountered during some run of a learning algorithm) and \vec{v} are *admissible* if all of their lengths are at most 1 and $\vec{v} \cdot \vec{x}_{t,i}$ is always in $[0, 1]$. In this case we also say that any run of an algorithm with these is admissible.

3.1. Algorithms. Define the clipping function π by letting $\pi(x)$ be the element of $[0, 1]$ that is closest to x .

3.1.1. Algorithm BW. As did algorithms CRW and CAW, algorithm BW chooses each alternative *not* estimated to be the best by the current hypothesis with probability roughly inversely proportional to how much worse it is predicted to be compared with the alternative that appears to be best. It sets $\kappa = m^{3/4}\sqrt{n}/2$, $\alpha = 1/\sqrt{m}$, and $\vec{w}_1 = (0, \dots, 0)$. On the t th trial, the algorithm

- for each alternative i , sets $\hat{y}_{t,i} = \pi(\vec{w}_t \cdot \vec{x}_{t,i})$,
- sets $g_t \in \{1, \dots, n\}$ to be some alternative that maximizes \hat{y}_{t,g_t} ,
- for each alternative i other than g_t , sets

$$p_{t,i} = \frac{1}{n + 4\kappa(\alpha - \alpha^2)(\hat{y}_{t,g_t} - \hat{y}_{t,i})},$$

- sets $p_{t,g_t} = 1 - \sum_{i \neq g_t} p_{t,i}$,
- chooses a_t randomly according to $p_{t,1}, \dots, p_{t,n}$,
- receives $z_{t,a_t} \in \{0, 1\}$ from the environment (where for all i , $\Pr(z_{t,i} = 1) = \vec{v} \cdot \vec{x}_{t,i}$; we refer to $\vec{v} \cdot \vec{x}_{t,i}$ as $y_{t,i}$),
- sets $\vec{w}_{t+1} = \vec{w}_t + \alpha(z_{t,a_t} - \vec{w}_t \cdot \vec{x}_{t,a_t})\vec{x}_{t,a_t}$.

3.1.2. Algorithm BU. As did CAU, algorithm BU picks the not-apparently-best alternatives with equal probability and it takes a bigger step when a not-apparently-best alternative is chosen. In particular, it sets $\kappa = m^{4/5}n^{2/5}$, $p = \sqrt{n}/(3\kappa)^{1/4}$, $\alpha = \frac{1}{2}n^{1/3}/(p\kappa)^{2/3}$, $\beta = 1/(2\kappa^{2/3})$, and $\vec{w}_1 = (0, \dots, 0)$. On the t th trial, algorithm BU

- for each alternative i , calculates its estimate $\hat{y}_{t,i} = \pi(\vec{w}_t \cdot \vec{x}_{t,i})$ of the probability of success for alternative i on this trial,
- sets $g_t \in \{1, \dots, n\}$ to be some alternative that maximizes \hat{y}_{t,g_t} ,
- flips a biased coin, and
 - with probability p ,
 - * chooses a_t uniformly at random from $\{1, \dots, n\}$,

- * receives $z_{t,a_t} \in \{0, 1\}$ from the environment, and
- * sets $\vec{w}_{t+1} = \vec{w}_t + \alpha(z_{t,a_t} - \vec{w}_t \cdot \vec{x}_{t,a_t})\vec{x}_{t,a_t}$, and
- with probability $1 - p$,
- * sets $a_t = g_t$,
- * receives $z_{t,a_t} \in \{0, 1\}$ from the environment, and
- * sets $\vec{w}_{t+1} = \vec{w}_t + \beta(z_{t,a_t} - \vec{w}_t \cdot \vec{x}_{t,a_t})\vec{x}_{t,a_t}$.

3.2. *Upper Bounds.* In this section we analyze the algorithms presented in Section 3.1.

3.2.1. *Preliminaries.* Straightforward application of calculus leads to the following variant of Lemma 7.

LEMMA 12. *For any $\vec{v}, \vec{w}_{\text{old}}, \vec{x} \in \mathbf{R}^n$, $z, \alpha \in \mathbf{R}$ for which $\|\vec{x}\| \leq 1$, $0 < \alpha < \frac{1}{2}$, and $z \in [0, 1]$, if $y = \vec{v} \cdot \vec{x}$ and $\vec{w}_{\text{new}} = \vec{w}_{\text{old}} + \alpha(y - \hat{y})\vec{x}$, then*

$$(5) \quad \begin{aligned} & \|\vec{w}_{\text{new}} - \vec{v}\|^2 - \|\vec{w}_{\text{old}} - \vec{v}\|^2 \\ & \leq -(\alpha - \alpha^2/2)(\pi(\vec{w}_{\text{old}} \cdot \vec{x}) - z)^2 + (\alpha + \alpha^2/2 + \alpha^3/3)(y - z)^2. \end{aligned}$$

PROOF. Given in Appendix A.5. □

In our next lemma we assume that z is generated randomly according to $\vec{v} \cdot \vec{x}$, and the progress is given in terms of how well $\vec{w}_{\text{old}} \cdot \vec{x}$ approximates this probability.

LEMMA 13. *For any $\vec{v}, \vec{w}_{\text{old}}, \vec{x} \in \mathbf{R}^n$, $\alpha \in \mathbf{R}$ for which $\|\vec{x}\| \leq 1$, $0 < \alpha < \frac{1}{2}$, if $y = \vec{v} \cdot \vec{x} \in [0, 1]$, $\hat{y} = \pi(\vec{w}_{\text{old}} \cdot \vec{x})$, and if $z \in \{0, 1\}$ is chosen randomly so that $\Pr(z = 1) = y$ and $\vec{w}_{\text{new}} = \vec{w}_{\text{old}} + \alpha(z - \hat{y})\vec{x}$, then*

$$\mathbf{E}(\|\vec{w}_{\text{new}} - \vec{v}\|^2 - \|\vec{w}_{\text{old}} - \vec{v}\|^2) \leq -(\alpha - \alpha^2/2)(\hat{y} - y)^2 + \alpha^2 + \alpha^3/3.$$

PROOF. Applying Lemma 12,

$$\begin{aligned} & \mathbf{E}(\|\vec{w}_{\text{new}} - \vec{v}\|^2 - \|\vec{w}_{\text{old}} - \vec{v}\|^2) \\ & \leq y(-(\alpha - \alpha^2/2)(1 - \hat{y})^2 + (\alpha + \alpha^2/2 + \alpha^3/3)(1 - y)^2) \\ & \quad + (1 - y)(-(\alpha - \alpha^2/2)\hat{y}^2 + (\alpha + \alpha^2/2 + \alpha^3/3)y^2). \end{aligned}$$

Let $r = \hat{y} - y$. Then

$$\begin{aligned} & \mathbf{E}(\|\vec{w}_{\text{new}} - \vec{v}\|^2 - \|\vec{w}_{\text{old}} - \vec{v}\|^2) \\ & \leq y(-(\alpha - \alpha^2/2)(1 - (y + r))^2 + (\alpha + \alpha^2/2 + \alpha^3/3)(1 - y)^2) \\ & \quad + (1 - y)(-(\alpha - \alpha^2/2)(y + r)^2 + (\alpha + \alpha^2/2 + \alpha^3/3)y^2). \end{aligned}$$

Simplifying yields

$$\mathbf{E}(\|\vec{w}_{\text{new}} - \vec{v}\|^2 - \|\vec{w}_{\text{old}} - \vec{v}\|^2) \leq -(\alpha - \alpha^2/2)r^2 + (\alpha^2 + \alpha^3/3)y(1 - y),$$

which, since $y \in [0, 1]$, completes the proof. □

3.2.2. *Analysis of Algorithm BW.* The following theorem is our main result about algorithm BW.

THEOREM 14. *On any admissible run of algorithm BW, if $\mathbf{E}(\cdot)$ represents the expectation with respect to all the randomization in the learning process,*

$$\sum_{t=1}^m \max_i \mathbf{E}(z_{t,i}) - \sum_{t=1}^m \mathbf{E}(z_{t,a_t}) \leq (2 + o(1))n^{1/2}m^{3/4},$$

where $o(1)$ denotes a quantity whose limit as m goes to infinity is 0.

The proof of Theorem 14 makes use of the following lemma.

LEMMA 15. *On any admissible run of algorithm BW, on any trial t , if $\mathbf{E}(\cdot)$ represents the expectation with respect to all the randomization in the learning process,*

$$\begin{aligned} \max_i y_{t,i} - \mathbf{E}(z_{t,a_t}) &\leq \kappa \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) + \frac{n}{2\kappa(\alpha - \alpha^2)} \\ &\quad + \kappa(\alpha^2 + \alpha^3/3). \end{aligned}$$

PROOF. Choose an admissible run of algorithm BW, and fix some trial t . Let $\text{progress} = \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2)$ and $\text{best} = \max_i y_{t,i}$ and drop the subscript t from all notation. Choose b so that $y_b = \max_i y_i$.

Applying Lemma 13,

$$\begin{aligned} \text{best} - \mathbf{E}(z_a) - \kappa \text{progress} &\leq y_b - \left(\sum_{i=1}^n p_i y_i \right) - \kappa \left(\sum_{i=1}^n p_i ((\alpha - \alpha^2/2)(\hat{y}_i - y_i)^2 - \alpha^2 - \alpha^3/3) \right) \\ &= \left(\sum_{i=1}^n p_i (y_b - y_i) \right) - \kappa \left(\sum_{i=1}^n p_i ((\alpha - \alpha^2/2)(\hat{y}_i - y_i)^2 - \alpha^2 - \alpha^3/3) \right) \\ &= \left(\sum_{i=1}^n p_i (y_b - y_i - \kappa(\alpha - \alpha^2/2)(\hat{y}_i - y_i)^2) \right) + \kappa(\alpha^2 + \alpha^3/3). \end{aligned}$$

Using calculus to find the worst-case values of the y_i 's as in the proof of Lemma 3, we have

$$\begin{aligned} (6) \quad \text{best} - \mathbf{E}(z_a) - \kappa \text{progress} &\leq \left(\sum_{i \neq b} p_i (\hat{y}_b - \hat{y}_i) \right) + \frac{1 - p_b}{4\kappa(\alpha - \alpha^2/2)} \\ &\quad + \frac{(1 - p_b)^2}{4p_b\kappa(\alpha - \alpha^2/2)} + \kappa(\alpha^2 + \alpha^3/3). \end{aligned}$$

For all $i \in \{1, \dots, n\}$, let $u_i = \hat{y}_b - \hat{y}_i$. □

Case 1: $g = b$. In this case as in Lemma 3, the lemma is a direct consequence of the fact that $p_g \geq 1/n$.

Case 2: $g \neq b$. In this case (6) implies

$$\begin{aligned}
& \text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} \\
& \leq \left(\sum_{i \neq b} p_i ((\hat{y}_g - u_b) - \hat{y}_i) \right) + \frac{1 - p_b}{4\kappa(\alpha - \alpha^2/2)} + \frac{(1 - p_b)^2}{4p_b\kappa(\alpha - \alpha^2/2)} \\
& \quad + \kappa(\alpha^2 + \alpha^3/3) \\
& = \left(\sum_{i \neq b} p_i (\hat{y}_g - \hat{y}_i) \right) - (1 - p_b)u_b + \frac{1 - p_b}{4\kappa(\alpha - \alpha^2/2)} + \frac{(1 - p_b)^2}{4p_b\kappa(\alpha - \alpha^2/2)} \\
& \quad + \kappa(\alpha^2 + \alpha^3/3) \\
& = \left(\sum_{i=1}^n p_i u_i \right) - u_b + \frac{1 - p_b}{4\kappa(\alpha - \alpha^2/2)} + \frac{(1 - p_b)^2}{4p_b\kappa(\alpha - \alpha^2/2)} \\
& \quad + \kappa(\alpha^2 + \alpha^3/3) \\
& \leq \left(\sum_{i \neq g} p_i u_i \right) - u_b + \frac{1}{4\kappa(\alpha - \alpha^2/2)} + \frac{1}{4p_b\kappa(\alpha - \alpha^2/2)} \\
& \quad + \kappa(\alpha^2 + \alpha^3/3),
\end{aligned}$$

as $u_g = 0$ and $p_b \geq 0$.

Substituting into the p_b in the denominator, we get

$$\begin{aligned}
\text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} & \leq \left(\sum_{i \neq g} p_i u_i \right) - u_b + \frac{1}{4\kappa(\alpha - \alpha^2/2)} \\
& \quad + \frac{n + 4\kappa(\alpha - \alpha^2/2)u_b}{4\kappa(\alpha - \alpha^2/2)} + \kappa(\alpha^2 + \alpha^3/3) \\
& = \left(\sum_{i \neq g} p_i u_i \right) + \frac{n + 1}{4\kappa(\alpha - \alpha^2/2)} + \kappa(\alpha^2 + \alpha^3/3).
\end{aligned}$$

Substituting into the remaining p_i 's completes the proof. \square

PROOF OF THEOREM 14. Assume without loss of generality that $m > 1$. For each t , let $\text{best}_t = \max_i y_{t,i}$. Applying Lemma 15, on each trial t ,

$$\text{best}_t - \mathbf{E}(z_{t,a_t}) \leq \kappa \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) + \frac{n}{2\kappa(\alpha - \alpha^2/2)} + \kappa(\alpha^2 + \alpha^3/3),$$

and therefore

$$\begin{aligned}
\sum_{t=1}^m (\text{best}_t - \mathbf{E}(z_{t,a_t})) &\leq \kappa \mathbf{E} \left(\sum_{t=1}^m (\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) \right) \\
&\quad + \frac{nm}{2\kappa(\alpha - \alpha^2/2)} + \kappa(\alpha^2 + \alpha^3/3)m \\
&= \kappa \mathbf{E}(\|\vec{w}_1 - \vec{v}\|^2 - \|\vec{w}_{m+1} - \vec{v}\|^2) \\
&\quad + \frac{nm}{2\kappa(\alpha - \alpha^2/2)} + \kappa(\alpha^2 + \alpha^3/3)m \\
&\leq \kappa(1 + (\alpha^2 + \alpha^3/3)m) + \frac{nm}{2\kappa(\alpha - \alpha^2/2)}.
\end{aligned}$$

Substituting the values of κ and α and simplifying yields

$$\sum_{t=1}^m (\text{best}_t - \mathbf{E}(z_{t,a_t})) \leq \left(\frac{24\sqrt{m} - 4 - 1/\sqrt{m}}{12\sqrt{m} - 6} \right) m^{3/4} \sqrt{n},$$

completing the proof. \square

3.2.3. *Analysis of Algorithm BU.* The following is our main result about BU.

THEOREM 16. *On any admissible run of algorithm BU, if $\mathbf{E}(\cdot)$ represents the expectation with respect to all the randomization in the learning process,*

$$\sum_{t=1}^m \max_i \mathbf{E}(z_{t,i}) - \sum_{t=1}^m \mathbf{E}(z_{t,a_t}) \leq 5n^{2/5}m^{4/5}.$$

The proof of Theorem 16 makes use of the following lemma.

LEMMA 17. *On any admissible run of algorithm BU, on any trial t , if $\mathbf{E}(\cdot)$ represents the expectation with respect to all the randomization in the learning process,*

$$\begin{aligned}
\max_i y_{t,i} - \mathbf{E}(z_{t,a_t}) &\leq \kappa \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) + p + \frac{n}{4(\alpha - \alpha^2/2)\kappa p} \\
&\quad + \frac{1}{4(\beta - \beta^2/2)\kappa} + (\alpha^2 + \alpha^3/3)p\kappa + (\beta^2 + \beta^3/3)\kappa.
\end{aligned}$$

PROOF. Choose an admissible run of BU, and fix some trial t . Let

$$\text{progress} = \mathbf{E}(\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2)$$

and $\text{best} = \max_i y_{t,i}$ and drop the subscript t from all other notation. Choose b so that $y_b = \max_i y_i$.

Clearly,

$$\text{best} - \mathbf{E}(z_a) \leq (1 - p)(y_b - y_g) + p,$$

and applying Lemma 13, we have

$$\begin{aligned} \text{progress} &\geq (p(\alpha - \alpha^2/2)/n)(y_b - \hat{y}_b)^2 + (1-p)(\beta - \beta^2/2)(y_g - \hat{y}_g)^2 \\ &\quad - (1-p)(\beta^2 + \beta^3/3) - p(\alpha^2 + \alpha^3/3) \end{aligned}$$

so

$$\begin{aligned} \text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} &\leq (1-p)(y_b - y_g) + p - (\kappa p(\alpha - \alpha^2/2)/n)(y_b - \hat{y}_b)^2 \\ &\quad - (1-p)\kappa(\beta - \beta^2/2)(y_g - \hat{y}_g)^2 \\ &\quad + (1-p)\kappa(\beta^2 + \beta^3/3) + p\kappa(\alpha^2 + \alpha^3/3). \end{aligned}$$

The right-hand side of this inequality is maximized, as a function of y_b , when $y_b = \hat{y}_b + (1-p)n/(2\kappa p(\alpha - \alpha^2/2))$, and so

$$\begin{aligned} \text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} &\leq (1-p)(\hat{y}_b - y_g) + p + \frac{(1-p)^2 n}{4\kappa p(\alpha - \alpha^2/2)} \\ &\quad - (1-p)\kappa(\beta - \beta^2/2)(y_g - \hat{y}_g)^2 \\ &\quad + (1-p)\kappa(\beta^2 + \beta^3/3) + p\kappa(\alpha^2 + \alpha^3/3). \end{aligned}$$

The right-hand side of this inequality is maximized, as a function of y_g , when $y_g = \hat{y}_g - 1/2\kappa(\beta - \beta^2/2)$, which implies

$$\begin{aligned} \text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} &\leq (1-p)(\hat{y}_b - \hat{y}_g) + p + \frac{(1-p)^2 n}{4\kappa p(\alpha - \alpha^2/2)} \\ &\quad + \frac{(1-p)}{4\kappa(\beta - \beta^2/2)} + (1-p)\kappa(\beta^2 + \beta^3/3) \\ &\quad + p\kappa(\alpha^2 + \alpha^3/3). \end{aligned}$$

Finally, the definition of \hat{y}_g implies that $\hat{y}_g \geq \hat{y}_b$, so

$$\begin{aligned} \text{best} - \mathbf{E}(z_a) - \kappa \text{ progress} &\leq p + \frac{(1-p)^2 n}{4\kappa p(\alpha - \alpha^2/2)} + \frac{(1-p)}{4\kappa(\beta - \beta^2/2)} \\ &\quad + (1-p)\kappa(\beta^2 + \beta^3/3) + p\kappa(\alpha^2 + \alpha^3/3), \end{aligned}$$

completing the proof. \square

PROOF OF THEOREM 16. Assume without loss of generality that $m > 1$. For each t , let $\text{best}_t = \max_i y_{t,i}$. Applying Lemma 17,

$$\begin{aligned} &\sum_{t=1}^m (\text{best}_t - \mathbf{E}(z_{t,a_t})) \\ &\leq \kappa \mathbf{E} \left(\sum_{t=1}^m (\|\vec{w}_t - \vec{v}\|^2 - \|\vec{w}_{t+1} - \vec{v}\|^2) \right) + pm + \frac{nm}{4(\alpha - \alpha^2/2)\kappa p} \end{aligned}$$

$$\begin{aligned}
& + \frac{m}{4(\beta - \beta^2/2)\kappa} + (\alpha^2 + \alpha^3/3)pkm + (\beta^2 + \beta^3/3)\kappa m \\
& = \kappa \mathbf{E}(\|\vec{w}_1 - \vec{v}\|^2 - \|\vec{w}_{m+1} - \vec{v}\|^2) + pm + \frac{nm}{4(\alpha - \alpha^2/2)\kappa p} \\
& \quad + \frac{m}{4(\beta - \beta^2/2)\kappa} + (\alpha^2 + \alpha^3/3)pkm + (\beta^2 + \beta^3/3)\kappa m \\
& \leq \kappa + pm + \frac{nm}{4(\alpha - \alpha^2/2)\kappa p} + \frac{m}{4(\beta - \beta^2/2)\kappa} + (\alpha^2 + \alpha^3/3)pkm \\
& \quad + (\beta^2 + \beta^3/3)\kappa m.
\end{aligned}$$

Substituting the values of α and β , and applying the fact that each is at most $\frac{1}{2}$, we get

$$\sum_{t=1}^m (\text{best}_t - \mathbf{E}(z_{t,a_t})) \leq \kappa + pm + \frac{mn^{2/3}}{(\kappa p)^{1/3}} + \frac{m}{\kappa^{1/3}}.$$

Substituting the value of p , we get

$$\sum_{t=1}^m (\text{best}_t - \mathbf{E}(z_{t,a_t})) \leq \kappa + \frac{2m\sqrt{n}}{\kappa^{1/4}} + \frac{2m}{\kappa^{1/3}}.$$

Substituting the value of κ completes the proof. \square

3.3. Lower Bounds. Our lower bound will be proved using a reduction from the *bandit problem* (see [12]). In the instance of the bandit problem that we need for our application, an algorithm is confronted with a row of K slot machines. Each time a slot machine is played it either pays off or doesn't. Each slot machine pays off with some probability that is unknown to the algorithm, and each time the algorithm plays some slot machine, that random outcome is independent of the other plays. The algorithm makes a sequence of T choices of which machine to play, and each time it plays some machine, it finds out whether that machine pays off. Randomized algorithms are allowed. The goal is to maximize the total number of the T plays that pay off. We make use of the following technical lemma.

LEMMA 18 [8]. *There is a constant $\gamma > 0$ such that, for any algorithm B for the bandit problem, for any $T \geq K \geq 2$, if a slot machine $i \in \{1, \dots, K\}$ is chosen uniformly at random, and*

- *the probability that slot machine i pays off is set to $p_i = \frac{1}{2} + \frac{1}{4}\sqrt{K/T}$, and*
- *the probability that all other slot machines pay off is set to $\frac{1}{2}$, then*

if z_1, \dots, z_T is the random sequence of outcomes obtained by applying B to those slot machines,

$$\mathbf{E} \left(p_i T - \sum_{t=1}^T z_t \right) \geq \gamma \sqrt{KT}.$$

We apply this in our main lower bound argument, which shows that the bounds obtained by combining Theorems 14 and 16 with the trivial upper bound of m are nearly best possible bounds in terms of m and n .

THEOREM 19. *For any number $m \geq 2$ of trials and any number $n \geq 2$ of alternatives per trial and any algorithm L , there is a number d of features, a sequence $\langle \vec{x}_{t,i} \rangle_{t,i}$ of feature vectors, and a coefficient vector \vec{v} such that, if a_1, \dots, a_m are the (random) choices arising from L , $\langle \vec{x}_{t,i} \rangle_{t,i}$, and \vec{v} , and $z_{1,a_1}, \dots, z_{m,a_m}$ is the corresponding random sequence of success/failure events, then, if γ is the constant from Lemma 18, we have*

$$\sum_{t=1}^m (\max_i \vec{v} \cdot \vec{x}_{t,i}) - \mathbf{E}(z_{t,a_t}) \geq \gamma \min\{m^{3/4}n^{1/4}, m\}.$$

PROOF. We divide our analysis into cases, depending on the relative size of m and n . We start with the easy case.

Case 1: $m \leq 16n$. Suppose $d = n$ and $\vec{e}_1, \dots, \vec{e}_n$ are the feature vectors given on each of the m trials. Let $m' = \min\{m, \lfloor n/2 \rfloor\}$. For each i , let p_i be the probability that action i is chosen during the first m' trials if all choices result in failure. Since only m' actions can ever be taken during this time, $\sum_i p_i \leq m' \leq n/2$, so there exists i with $p_i \leq \frac{1}{2}$. If $\vec{v} = \vec{e}_i$, then with probability $\frac{1}{2}$, action i will not be chosen during the first m' trials, and the expected regret is therefore at least $m'/2 \geq m/64$.

Case 2: $m > 16n$. Let $r = \lfloor \sqrt{mn}/4 \rfloor$, and divide the first $r \lfloor m/r \rfloor$ trials into $\lfloor m/r \rfloor$ stages with r trials each. In each of these stages, we simulate an instance of the bandit problem as follows.

In this case, we set the number of features d to be $n \lfloor m/r \rfloor + 1$. For simplicity, we number features from 0. Feature 0 has a value of $\sqrt{\frac{1}{2}}$ for all alternatives on all trials. During the j th stage, the value of the $((j-1)r+i)$ th feature of the i th alternative is also $\sqrt{\frac{1}{2}}$, and all other features have a value of 0. For example, the sequence of trials (alternatives with their feature values) for $n = 2$ is shown in Figure 1.

Once we have the fixed feature vectors as above, any algorithm A for reinforcement learning of probabilistic linear functions with immediate rewards from m trials, gives rise to a sequence $B_1, \dots, B_{\lfloor m/r \rfloor}$ of algorithms for the bandit problem with r plays as follows. One views the state of the algorithm A at the beginning of the j th stage as a random input (i.e. as randomization), and then the decisions made by algorithm A during the j th stage as those of a randomized algorithm for solving the bandit problem.

Now we set the coefficients of the target linear function as follows. First, we set $v_0 = \sqrt{\frac{1}{2}}$. For each stage j , choose i_j uniformly at random from $\{1, \dots, n\}$. Then for each stage j , set $v_{(j-1)r+i_j} = (\sqrt{2}/4)\sqrt{n/r}$, and $v_{(j-1)r+i} = 0$ for all $i \neq i_j$.

Note that, for each stage j and alternative i , the probability of success when that alternative i is chosen is constant throughout stage j . Furthermore, the results during stages before stage j provide no information about the probabilities of success during stage j .

$$\begin{array}{l}
 \text{Stage 1} \left\{ \begin{array}{l}
 \text{Trial 1} \quad \left\{ \begin{array}{l} \sqrt{\frac{1}{2}} \\ \sqrt{\frac{1}{2}} \end{array} \right. \begin{array}{l} \sqrt{\frac{1}{2}} \\ 0 \end{array} \begin{array}{l} 0 \\ \sqrt{\frac{1}{2}} \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \dots \\
 \vdots \\
 \text{Trial } r \quad \left\{ \begin{array}{l} \sqrt{\frac{1}{2}} \\ \sqrt{\frac{1}{2}} \end{array} \right. \begin{array}{l} \sqrt{\frac{1}{2}} \\ 0 \end{array} \begin{array}{l} 0 \\ \sqrt{\frac{1}{2}} \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \dots
 \end{array} \right. \\
 \\
 \text{Stage 2} \left\{ \begin{array}{l}
 \text{Trial } r + 1 \quad \left\{ \begin{array}{l} \sqrt{\frac{1}{2}} \\ \sqrt{\frac{1}{2}} \end{array} \right. \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} \sqrt{\frac{1}{2}} \\ 0 \end{array} \begin{array}{l} 0 \\ \sqrt{\frac{1}{2}} \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \dots \\
 \vdots \\
 \text{Trial } 2r \quad \left\{ \begin{array}{l} \sqrt{\frac{1}{2}} \\ \sqrt{\frac{1}{2}} \end{array} \right. \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} \sqrt{\frac{1}{2}} \\ 0 \end{array} \begin{array}{l} 0 \\ \sqrt{\frac{1}{2}} \end{array} \begin{array}{l} 0 \\ 0 \end{array} \begin{array}{l} 0 \\ 0 \end{array} \dots
 \end{array} \right. \\
 \vdots
 \end{array}$$

Fig. 1. The feature vectors used for the sequence of trials in the proof of Theorem 19, in the case $n = 2$.

With this coefficient vector and the feature vectors as described above, the probability of success for i_j during the j th stage is $\frac{1}{2} + \sqrt{n/r}/4$, and for all other alternatives, this probability is $\frac{1}{2}$.

The length of the feature vectors and the coefficient vector are at most 1. Furthermore, applying Lemma 18, there is a constant $\gamma' > 0$ such that $\mathbf{E}(\sum_{t=1}^m \text{best}_t - z_{t,a_t}) \geq \gamma' \lfloor m/r \rfloor \sqrt{rn}$, where this expectation is with respect to the random choice of \vec{v} as well as the randomness of the learning process. This implies that there exists a choice for \vec{v} , such that, for that fixed \vec{v} , $\sum_{t=1}^m (\max_i \vec{v} \cdot \vec{x}_{t,i}) - \mathbf{E}(z_{t,a_t}) \geq \gamma' \lfloor m/r \rfloor \sqrt{rn}$. Substituting the value of r and simplifying completes the proof. \square

3.4. *Improvement.* Auer [6] obtained a result that improves significantly on Theorems 14 and 16 when the number of features d is small relative to m and n . His bound is $O(\sqrt{md} \log(mn))$.

4. Conclusions and Future Work. In this paper we have considered the problem of reinforcement learning with immediate rewards in a worst-case theoretical framework.

While the best known upper and lower bounds for the problems studied here are pretty close, there is obvious room for improvement. We hope that this line of theoretical research will generate some practically useful ideas.

Extending the analysis concerning probabilistically generated, binary-valued rewards to the agnostic case would be nice.

Adaptively ranking the results of a web search seems to be accurately modeled with immediate rewards, and this problem contains an exploration/exploitation tradeoff, since

pages ranked low enough are unlikely to be chosen, no matter how attractive they would have been to the user. A worst-case analysis seems particularly relevant to this problem, since designers of web pages have aims that are not necessarily compatible with those doing the searching, and therefore with the ranking algorithm. Ranking search results can be viewed as an extension of the problem of choosing a banner ad, in which the choice of a single item is replaced with the choice of a ranked list. Therefore, hopefully ideas from this paper can be applied in a worst-case theoretical study of the ranking problem. Such an analysis would of course be relevant to the more general information retrieval problem. This is but one of a wide variety of problems involving short-term interactions with users that seem to be accurately modeled by reinforcement learning with immediate rewards.

Acknowledgements. We thank Greg Keim and Atsu Nakamura for valuable conversations, and Wee Sun Lee and Atsu Nakamura for their comments on earlier versions of this paper. We also thank anonymous referees for their thoughtful reviews.

Appendix A. An Application—Multimodal Message Generation. In this Appendix we describe an application of the model of reinforcement learning with immediate continuous-valued rewards and linear hypotheses: an adaptive grammar-based method for generating multimodal messages.

One representative context in which the problem of message generation arises is in the design of an interactive tutoring system. One module in a typical architecture for such a system takes as input some abstract representation of an idea to convey to the user and must output a way to express that idea. The output module in the Pascal tutor developed by members of the Duke Voice and Natural Language Lab (see [13]) takes as input a prolog statement, and must output a message that is a mix of speech, text, and graphics. The most interesting subproblem faced by this module was identifying part of a program (for example, choosing between saying “there is something wrong with the fourth word on the third line”, saying “there is something wrong with the highlighted region” while highlighting, and so on), so we focus on this subproblem for the remainder of this section.

A.1. Operators. As mentioned above a key subproblem in the generation of output messages for the tutor is that of identifying a particular substring of the user’s program. We refer to this substring as the *target substring*.

The algorithm maintains a list of substrings that is updated through the application of rules. The list begins by consisting only of a single string: the entire program. The goal is to, through the application of certain rules, transform the list so it consists only of the target substring.

In our preliminary system, there are two kinds of rules:

- nouns (like `LINE`, `WORD`, and `SEMICOLON`), which describe a set of substrings of the “current” string, and
- selectors (like `FIRST` and `SECOND-FROM-LAST`), which choose a particular element from a list of substrings.

As mentioned above, an example of a noun operator is the `LINE` operator. A precondition for the application of the `LINE` operator is that the set of strings currently referred to consists of a single string. Applying the `LINE` operator results in this string being broken into several strings, one for each line in the original string.

An example of a selector operator is the `THIRD` operator. A precondition for the application of the `THIRD` operator is that the list of strings currently referred to contains no pair of strings that overlap. Applying the `THIRD` operator replaces the current list of strings with the list consisting only of the third string in the list.

Another class of selector operators are the graphics operators, for example, the `HIGHLIGHT` operator. Formally, there is a separate `HIGHLIGHT` operator for each substring of the original program, but to apply a `HIGHLIGHT` operator, its corresponding string must be an element of the list of strings currently referred to. Applying a `HIGHLIGHT` operator replaces the list of strings referred to with the highlighted string.

The algorithm backtracks when the list of strings referred to does not contain any string which in turn has the target string as a substring. The existence of an `ALL` operator, which replaces a string with the list of all of its substrings, ensures that the procedure terminates.

As the algorithm proceeds, in addition to the list of strings currently referred to, it also maintains a stack of those operators which have been applied. When the list of strings referred to consists only of the desired string, it is then fairly straightforward to construct the message describing the string using this stack. For example, if the string was arrived at by first applying the `LINE` operator, then the `THIRD` operator, then the `WORD` operator, then a `HIGHLIGHT` operator, then this portion of the message would have the computer say “. . . the highlighted word of the third line . . .” while highlighting the given word.

A.2. An Example. Suppose the program is

```
program hw;
begin
  writeln('Hello, wolrd');
end.
```

and the system wants to identify the misspelled word. The original list of strings consists only of the single string “program hw;<ret>begin<ret> writeln('Hello, wolrd');<ret>end.<ret>”. Applying the `LINE` operator would result in replacing this string with the following list of strings: “program hw;”, “begin”, “writeln('Hello, wolrd');”, and “end.”. Next, applying the third operator of the `ORDINAL` class would pick out the third item in the list, making the current list consist only of the string “writeln('Hello, wolrd');”. From this point, applying the `WORD` operator would result in our current list of strings consisting of “writeln”, “Hello”, and “wolrd”. Finally, applying the appropriate `HIGHLIGHT` operator would select the string “wolrd”, and the system would be done, constructing the message “. . . the highlighted word of the third line . . .” together with the action of highlighting the given word.

A.3. Scoring. Clearly, the above rule-based process can generate a huge variety of ways of expressing a particular substring of the user's program. Some of them are obviously worse than others. For example, one would never want a program to say “there is

something wrong with the thirteenth character of the twentieth line”. Therefore, we need some mechanism for scoring proposed messages, and for choosing low-cost messages.

One scoring mechanism is motivated by the following idealized user model: we assume that the user works through the message from general to specific, maintaining a list of strings currently referred to in a similar manner to that done during generation, except without the backtracking. Our goal is to ensure that the total time taken by the user to understand the message is small.

Concretely, the cost of a message is constructed as follows. First, for each class of noun operators, there is a particular (adjustable) constant cost for applying operators from that class. For operators that count, for example, the operator which identifies the third item in a list and the operator which identifies say the fourth from last item in a list, there is a constant c such that the cost of identifying the k th item is kc . This corresponds to a crude user model in which the user actually counts “one, two, three . . .” until the desired object is reached. Finally, for each class of graphical operators (in the basic implementation there is one for highlighting, one for blinking, and one for pointing with an arrow), there is a constant, call it c' such that if there are ℓ items in the list before application of the graphical operator, the cost of applying it is $c' \ln \ell$. This corresponds to a user model where the user zeros in on the highlighted item “geometrically”, first locating the general area, then successively refining.

In a nonadaptive mode, constants are fixed for each of the operator classes, and the algorithm searches the space of lists of strings by applying the operators, keeping track at any time of the least cost solution found so far and the running cost of the current partial solution, and pruning as described above.

A.4. Adapting the Coefficients. Recall that our cost function (we can convert costs to rewards by negating them) was motivated by consideration of the time it took the user to understand the message. This leads us to use (the negation of) some measure of the time for the user to respond to the message as reinforcement for the learning algorithm. Our initial implementation used the total time between when a message was sent and when the user next queried the tutor as reinforcement for the learning algorithm. At first glance, this reinforcement appears impractically crude. However, algorithm CAU of this paper is highly robust with respect to noise. Furthermore, the step size of the update made by CAU to the coefficients does not depend on the size of the error, and therefore the algorithm is intuitively even more robust with respect to flagrant outliers than is captured by our analysis. Our initial experience using the system suggests that it indeed learns to output subjectively good messages using such crude, however objective and passively obtained, feedback.

To illustrate how we arrive at features for the learning algorithm, assume that there are only two classes of operators for selecting strings from a list: the class of `ORDINAL` operators and the class of `HIGHLIGHT` operators. As described above, our cost model would in this case have constants for each of the nouns (like `WORD` and `LINE`), together with constants c_{ord} and c_{high} such that, for example, the cost of applying the `ORDINAL` operator to pick the third item from the list would be $3c_{\text{ord}}$, and the cost to use a `HIGHLIGHT` operator to pick any item from a list of ten items would be $(\ln 10)c_{\text{high}}$.

Now, let us examine the form taken by the cost of one output considered by an algorithm using multimedia grammars in this way. Suppose the algorithm is considering

saying “The highlighted character in the second word of the third line.” Suppose further that the second word of the third line of the highlight paragraph has ℓ characters in it. Then the cost of the above method of referring to the given character would be, according to the model outlined above,

$$c_{\text{ord}}3 + c_{\text{line}} + c_{\text{ord}}2 + c_{\text{word}} + c_{\text{high}} \ln \ell + c_{\text{char}}.$$

However, rearranging, we get that this is equal to

$$c_{\text{high}}(\ln \ell) + c_{\text{ord}}(3 + 2) + c_{\text{line}} + c_{\text{word}} + c_{\text{char}}.$$

For any proposed message, suppose we identify the following features:

- $\sum_i k_i$, where k_1, k_2, \dots are the indices referred to by the various applications of ordinal rules,
- $\sum_i \log \ell_i$, where ℓ_1, ℓ_2, \dots are the size of the focus lists when highlight rules are applied,
- for each noun for which there is an operator, the number of times that noun was used (typically either 0 or 1).

Then, by collecting terms, we can see that the cost of any proposed output is a linear function of these features, with the coefficients given by c_{ord} , c_{high} and the various noun costs. This implies that a linear function of these features is an appropriate cost model, and therefore that the analyses of Section 2.2 are relevant to this setting. (Note that the feature vectors constructed in this way may not be unit length, but recall that it is known how to modify algorithms that work for unit length feature vectors to handle the general case.)

Algorithms CAU and CAW described in this paper have a nonzero probability of picking any message. Due to the large number of possible messages, it is difficult to sample from all of them while operating in real time. Instead, in our implementation, the output system first generates a list of the lowest cost alternatives (according to the current cost function), and this is in turn passed to the learning algorithm. By adjusting the threshold for membership in this list, we are able to trade between computational efficiency, and the ability of the learning algorithm to “explore” freely. Setting the threshold in terms of cost relative to the lowest cost alternative, rather than having a fixed number of lowest cost alternatives, facilitates pruning and therefore efficient implementation. This results in a nonconstant number of alternatives being presented at each “trial”, but the algorithms are easily modified to cope with this.

A.5. Comments. We have implemented this system, using a variant of CAU, and integrated it into an existing Pascal tutor [13]. The tutor as a whole works in real time, interacting with the user using speech recognition and generated speech, graphics, and text as described in this paper, as well as by observing the user’s current program. The system noticeably adapts to different user preferences, and generates messages that subjectively appear reasonable.

Appendix B. Proof of Lemma 12. Define $f: \mathbf{R} \rightarrow \mathbf{R}$ to be the right-hand side of (3), viewed as a function of $\vec{w}_{\text{old}} \cdot \vec{x}$; i.e. for all u ,

$$f(u) = -(\alpha - \alpha^2/2)(u - z)^2 + (\alpha + \alpha^2/2 + \alpha^3/3)(y - z)^2.$$

Then

$$f'(u) = -2(\alpha - \alpha^2/2)(u - z).$$

If $u \geq 1$, then since $\alpha - \alpha^2/2 > 0$,

$$f'(u) \leq -2(\alpha - \alpha^2/2)(1 - z) \leq 0,$$

since z is at most 1. Thus, f is maximized, subject to $u \geq 1$, when $u = 1$.

If $u \leq 0$, then

$$f'(u) \geq (\alpha - \alpha^2/2)z \geq 0$$

since z is at least 0. Thus, f is maximized, subject to $u \leq 0$, when $u = 0$.

Overall, we have that $f(\pi(u)) \geq f(u)$, and putting this together with Lemma 7 completes the proof. \square

References

- [1] N. Abe and P. M. Long. Associative reinforcement learning using linear probabilistic concepts. *Proceedings of the 16th International Conference on Machine Learning*, pages 3–11, 1999.
- [2] N. Abe and A. Nakamura. Learning to optimally schedule internet banner advertisements. *Proceedings of the 16th International Conference on Machine Learning*, 1999.
- [3] C. Allenberg. Individual sequence prediction – upper bounds and an application for complexity. *Proceedings of the 1999 Conference on Computational Learning Theory*, pages 233–242, 1999.
- [4] P. Auer. An improved algorithm for learning linear evaluation functions. *Proceedings of the 2000 Conference on Computational Learning Theory*, 2000.
- [5] P. Auer. Using upper confidence bounds for online learning. *Proceedings of the 41st Annual Symposium on the Foundations of Computer Science*, 2000.
- [6] P. Auer. Using confidence bounds for exploitation–exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002. A preliminary version has appeared in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*.
- [7] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. *Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, 1995.
- [8] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. Technical Report NC-TR-98-025, Neurocolt, 1998.
- [9] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002. A preliminary version has appeared in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*.
- [10] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 15:360–374, 1985.
- [11] A. G. Barto, R. S. Sutton, and P. S. Brouwer. Associative search network: a reinforcement learning associative memory. *Biological Cybernetics*, 40:201–211, 1981.
- [12] D. A. Berry and B. Fristedt. *Bandit Problems*. Chapman and Hall, New York, 1985.
- [13] A. W. Biermann, C. I. Guinn, M. Fulkerson, G. Keim, Z. Liang, D. Melamed, and K. Rajagopalan. Goal-oriented multimedia dialogue with variable initiative. In *Foundations of Intelligent Systems*, Z. W. Ras and A. Skowron (eds.), 1997.
- [14] A. W. Biermann and P. M. Long. The composition of messages in speech-graphics interactive systems. *Proceedings of the 1996 International Symposium on Spoken Dialogue*, 1996.
- [15] N. Cesa-Bianchi, P. M. Long, and M. K. Warmuth. Worst-case quadratic loss bounds for prediction using linear functions and gradient descent. *IEEE Transactions on Neural Networks*, 7(3):604–619, 1996.

- [16] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth. How to use expert advice. *Journal of the Association for Computing Machinery*, 44(3):427–485, May 1997.
- [17] C.-N. Fiechter. Design and Analysis of Efficient Reinforcement Learning Algorithms. Ph.D. thesis, University of Pittsburgh, 1997.
- [18] D. Haussler, M. Kearns, H. S. Seung, and N. Tishby. Rigorous learning curve bounds from statistical mechanics. *Machine Learning*, 25:195–236, 1996.
- [19] D. P. Helmbold, N. Littlestone, and P. M. Long. Apple tasting. *Information and Computation*, 161(2):85–139, 2000. Preliminary version in *FOCS '92*.
- [20] L. P. Kaelbling. Associative reinforcement learning: a generate and test algorithm. *Machine Learning*, 15(3):299–320, 1994.
- [21] L. P. Kaelbling. Associative reinforcement learning: functions in k-dnf. *Machine Learning*, 15(3):279–298, 1994.
- [22] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of Boolean formulae. *Proceedings of the 19th Annual Symposium on the Theory of Computation*, pages 285–295, 1987.
- [23] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17:115–141, 1994.
- [24] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
- [25] P. M. Long. On-line evaluation and prediction using linear functions. *Proceedings of the 1997 Conference on Computational Learning Theory*, 1997.
- [26] K. S. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction*. Prentice-Hall, Englewood Cliffs, NJ, 1989.
- [27] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [28] B. Widrow and M. E. Hoff. Adaptive switching circuits. 1960 *IRE WESCON Conv. Record*, pages 96–104, 1960.
- [29] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.