INFORMATION AND COMPUTATION 132, 1–63 (1997)
ARTICLE NO. IC962612

Exponentiated Gradient versus Gradient Descent for Linear Predictors*

Jyrki Kivinen†

Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland

and

Manfred K. Warmuth‡

Computer and Information Sciences, University of California, Santa Cruz, California 95064

We consider two algorithms for on-line prediction based on a linear model. The algorithms are the well-known gradient descent (GD) algorithm and a new algorithm, which we call EG$^{\pm}$. They both maintain a weight vector using simple updates. For the GD algorithm, the update is based on subtracting the gradient of the squared error made on a prediction. The EG$^{\pm}$ algorithm uses the components of the gradient in the exponents of factors that are used in updating the weight vector multiplicatively. We present worst-case loss bounds for EG$^{\pm}$ and compare them to previously known bounds for the GD algorithm. The bounds suggest that the losses of the algorithms are in general incomparable, but EG$^{\pm}$ has a much smaller loss if only few components of the input are relevant for the predictions. We have performed experiments which show that our worst-case upper bounds are quite tight already on simple artificial data.

1. INTRODUCTION

We consider a scenario in which the learner, or learning algorithm, tries to accurately predict real-valued outcomes in a sequence of trials. In the beginning of the tth trial, the learner receives an instance $\mathbf{x}_t$, which is an N-dimensional real vector. The components $x_{t,i}$ of the instances are also called input variables. Based

* An extended abstract appeared in "Proceedings of the 27th Annual ACM Symposium on the Theory of Computing," pp. 209–218, ACM Press, New York, May 1995.

† Supported by the Academy of Finland, Emil Aaltonen Foundation, and ESPRIT Project NeuroCOLT.

‡ Supported by NSF Grant IRI-9123692. E-mail: manfred@cse.ucsc.edu.

0890-5401/97 $25.00

on the instance $\mathbf{x}_t$ and information received in the previous trials, the learner makes its real-valued prediction $\hat{y}_t$. After this, the actual tth outcome y_t is observed, and the learner is charged for the possible discrepancy between the predicted outcome $\hat{y}_t$ and the actual outcome y_t. The discrepancy is measured by a loss function L, for example by the square loss function given by $L(y_t, \hat{y}_t) = (y_t - \hat{y}_t)^2$. Over a long sequence of trials, the learner tries to minimize its total loss, which is simply the sum of the losses incurred at the individual trials. A learning algorithm that follows this protocol is called an (*on-line*) *prediction algorithm.*

Obviously, if no assumptions are made concerning the relation between the instances and outcomes, there is not much a prediction algorithm can do. To set a reasonable goal, we measure the performance of the algorithm against the performances of predictors from some fixed *comparison class* P. (The comparison class is analogous to the touchstone class of the agnostic PAC model of learning (Kearns *et al.*, 1994).) The algorithm is required to perform well if at least one predictor from the comparison class performs well. At the extremes, the outcomes could be completely random, in which case they can be predicted neither by the algorithm nor any predictor from the comparison class P, or the outcomes might always be completely predicted by one fixed predictor from P, in which case the algorithm should incur only a small loss before learning to follow that predictor.

In general, the predictors $p \in P$ are arbitrary mappings from $\mathbf{R}^N$ to $\mathbf{R}$. In this paper, we concentrate on *linear predictors.* To any vector $\mathbf{u} \in \mathbf{R}^N$ we associate a linear predictor $p_{\mathbf{u}}$, which is defined by $p_{\mathbf{u}}(\mathbf{x}) = \mathbf{u} \cdot \mathbf{x}$ for $\mathbf{x} \in \mathbf{R}^N$. Then any set $\mathscr{U} \subseteq \mathbf{R}^N$ of vectors defines a comparison class P of linear predictors by $P = \{p_{\mathbf{u}} \mid \mathbf{u} \in \mathscr{U}\}$.

Given an ℓ-trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, the total loss of the algorithm is given by $\mathrm{Loss}_L(A, S) = \sum_{t=1}^{\ell} L(y_t, \hat{y}_t)$, where $\hat{y}_t$ is the tth prediction of the algorithm. Analogously, the total loss of a predictor p is given by $\mathrm{Loss}_L(p, S) = \sum_{t=1}^{\ell} L(y_t, p(\mathbf{x}_t))$. For linear predictors, we also use the notation $\mathrm{Loss}_L(\mathbf{u}, S)$ for $\mathrm{Loss}_L(p_{\mathbf{u}}, S)$. A very first goal could be to obtain bounds of the form

$$\mathrm{Loss}_L(A, S) = O(\inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S)), \tag{1.1}$$

when we allow the length ℓ of sequences, and hence the total losses, increase without bound. At this first stage, we simplify the presentation by keeping the number N of dimensions constant and assuming that there is some fixed subset $\mathscr{X} \subseteq \mathbf{R}^N$ to which the instances $\mathbf{x}_t$ always belong. It turns out that we can often obtain something better than (1.1). We can get the coefficient of the leading term on the right-hand side of (1.1) down to 1 and thereby obtain

$$\mathrm{Loss}_L(A, S) = (1 + o(1)) \inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S), \tag{1.2}$$

where the quantity $o(1)$ approaches 0 as $\inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S)$ approaches infinity. Thus, the bound (1.2) means that the *additive additional loss* $\mathrm{Loss}_L(A, S) - \inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S)$ of the algorithm grows at a sublinear rate as a function of $\inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S)$. The asymptotic notation in (1.1) and (1.2) hides the dependence of the total loss of the algorithm on the number N of dimensions, as well as the

ranges $\mathscr{U}$ of the predictor vectors in the comparison class and the domain $\mathscr{X}$ of the instances. As these dependences are usually quite important, we will later also consider the constants not shown in the asymptotic bounds.

Note that the bounds (1.1) and (1.2) are worst-case bounds with respect to S. We may need to assume that the instances $\mathbf{x}_t$ belong to a restricted domain $\mathscr{X}$, but we do not assume that they are drawn from a probability measure, nor do we make any assumptions about how the outcomes y_t are related to the instances $\mathbf{x}_t$. If there is no good linear predictor $\mathbf{u}$ for the trial sequence S, then the right-hand sides of the bounds (1.1) and (1.2) are very large, so the bounds may not be very interesting. However, the bounds always hold. This is in contrast to more common approaches where statistical assumptions about the distribution of the instances and the dependence of the outcomes on the instances are used in order to derive probabilistic loss bounds for the prediction algorithm (Widrow and Stearns, 1985; Haykin, 1991).

The research reported in this paper was inspired by Littlestone (1989b, 1988), who proved worst-case bounds for the case when the comparison class consists of Boolean monomials, or more generally linear threshold functions. In this case it was assumed that the components of the instances, as well as the predictions and the outcomes, were Boolean, and the total loss was measured by the number of mistakes, i.e., the number of incorrect predictions made by the algorithm. More recently, there has been some work on using an arbitrary finite comparison class $P = \{p_1, ..., p_N\}$. Predictors from a finite class are often called *experts* (Cesa-Bianchi *et al.*, 1994). Note that a finite comparison class can be considered as a comparison class with a very restricted set of linear predictors. For $i = 1, ..., N$, let $\mathbf{u}^i \in \mathbf{R}^N$ be the unit vector with $u^i_i = 1$ and $u^i_j = 0$ for $j \neq i$. If we replace an instance $\mathbf{x}_t$ by the vector $\mathbf{x}'_t = (p_1(\mathbf{x}_t), ..., p_N(\mathbf{x}_t))$, the original predictor p_i, applied to the original instance $\mathbf{x}_t$, can be represented as the linear predictor $p_{\mathbf{u}^i}$, applied to the new instance $\mathbf{x}'_t$. Hence, instead of the original comparison class P we consider the comparison class of linear predictors $p_{\mathbf{u}}$ with $\mathbf{u} \in \mathscr{U}$, where $\mathscr{U}$ consists of the N unit vectors $\mathbf{u}^1, ..., \mathbf{u}^N$. The number of dimensions is now the number of experts, i.e., the size N of the original comparison class. Vovk (1990) proved that for a large class of loss functions, a simple algorithm achieves bounds of the form $\mathrm{Loss}_L(A, S) \leqslant \inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S) + c \log N$, where the constant c depends only on the loss function. Such bounds are even tighter than those of the form (1.2). However, for the absolute loss such bounds were not obtained. Vovk (1990) and Littlestone and Warmuth (1994) had bounds of the form (1.1) for the absolute loss. Later Cesa-Bianchi *et al.* (1994) showed how these bounds could be improved to the form (1.2) by a careful choice of certain parameters in Vovk's algorithm. Some of these results assumed that the outcomes y_t must be in $\{0, 1\}$ and were generalized for continuous-valued outcomes $y_t \in [0, 1]$ by Haussler *et al.* (1994).

In this paper, we consider proving bounds of the form (1.2) for a comparison class of general linear predictors, rather than only predictors that choose one of the N components of the instance. We also describe simple experiments that verify that the worst-case bounds reflect the actual behavior of the algorithms. We have succeeded in the proofs only for the square loss $(y_t - \hat{y}_t)^2$, although the basic ideas of this paper can be phrased for general loss functions. The immediate predecessors of

this work are the papers by Cesa-Bianchi *et al.* (1996) and Littlestone *et al.* (1995). Cesa-Bianchi *et al.* consider the *gradient descent algorithm*, or the GD algorithm, for linear predictions. This algorithm is also known as the Widrow–Hoff algorithm and the Least Mean Squares algorithm. It is also one of the main algorithms used in this paper. The algorithm maintains a weight vector and updates it after each trial. The tth weight vector $\mathbf{w}_t$ can be considered as the hypothesis the algorithm has before trial t about the best linear predictor for the trial sequence. At trial t, the algorithm gives the prediction $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$. After receiving the outcome y_t, it updates the weight vector according to the *update rule*

$$\mathbf{w}_{t+1} = \mathbf{w}_t - 2\eta(\hat{y}_t - y_t)\,\mathbf{x}_t,$$

where η is a positive *learning rate*. To motivate our name GD for this algorithm, note that the derivative of the loss $(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2$ of the algorithm with respect to the weight $w_{t,i}$ is given by $2(\mathbf{w}_t \cdot \mathbf{x}_t - y_t)x_{t,i}$. Hence, the update subtracts from the weight vector the gradient $\nabla_{\mathbf{w}_t}(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2$ multiplied by the scalar η. The GD algorithm can be considered as a simple application of the gradient descent heuristic to our on-line prediction problem. Choosing the learning rate η is non-trivial and can significantly affect the performance of the algorithm.

We also introduce a new on-line prediction algorithm, which we call the *exponentiated gradient algorithm*, or the EG algorithm. The EG algorithm is closely related to the algorithm given by Littlestone *et al.* (1995). The EG algorithm also has a weight vector $\mathbf{w}_t$ and predicts with $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$. The update rule is

$$w_{t+1,i} = \frac{r_{t,i} w_{t,i}}{\sum_{j=1}^{N} r_{t,j} w_{t,j}},$$

where $r_{t,j} = e^{-2\eta(\hat{y}_t - y_t)x_{t,j}}$ for some positive learning rate η. Thus, the ith component of the gradient now appears in the exponent of a factor that multiplies $w_{t,i}$.

The weights of the EG algorithm are positive and sum to 1. This restriction on the weight vector clearly also restricts the predictive ability of the algorithm. Therefore, we also introduce the *exponentiated gradient algorithm with positive and negative weights*, or the EG$^{\pm}$ algorithm. The EG$^{\pm}$ algorithm is obtained by applying a simple transformation to the EG algorithm. The components of its weight vector can be positive or negative. Their sum is not fixed, but the algorithm assumes a fixed upper bound for it.

The algorithms GD and EG can be motivated using a common framework. In making an update, the algorithm must balance its need to be conservative, i.e., retain the information it has acquired in the preceding trials, and to be corrective, i.e., to make certain that if the same instance were observed again, the algorithm could make a more accurate prediction, at least if the outcome is also the same. Thus, with an old weight vector $\mathbf{s}$, the algorithm chooses a new weight vector $\mathbf{w}$ that approximately minimizes

$$d(\mathbf{w}, \mathbf{s}) + \eta L(y_t, \mathbf{w} \cdot \mathbf{x}_t),$$

where $d(\mathbf{w}, \mathbf{s})$ is some measure of distance from the old to the new weight vector, L is the loss function, and the magnitude of the positive constant η represents the importance of correctiveness compared to the importance of conservativeness. The measure d is typically not a metric. For the square loss function, using the squared Euclidean distance $d(\mathbf{w}, \mathbf{s}) = \frac{1}{2}\|\mathbf{w} - \mathbf{s}\|_2^2$ results in the GD algorithm. The EG algorithm results from using for d the relative entropy, also known as Kullback–Leibler divergence,

$$d_{\mathrm{re}}(\mathbf{w}, \mathbf{s}) = \sum_{i=1}^{N} w_i \ln \frac{w_i}{s_i}.$$

This assumes that all the components s_i and w_i are positive, and the constraints $\sum_i s_i = \sum_i w_i = 1$ are maintained. The use of the relative entropy as a distance measure is motivated by the *Maximum Entropy Principle* of Jaynes and the more general *Minimum Relative Entropy Principle* of Kullback. These fundamental principles have many applications in information theory, physics and economics. (See Kapur and Kesavan (1992) and Jumarie (1990) for an overview.)

For our work it is central that the distance measure is used in two different ways: first, it motivates the update rule, and second, it is applied as a tool in the analysis of the algorithm thus obtained. By estimating the change of distance from the weight vector $\mathbf{w}_t$ of the algorithm to a comparison vector $\mathbf{u}$ at each update, it is possible to prove the kind of worst-case loss bounds we consider here. This use of a distance measure for obtaining worst-case loss bounds was pioneered by Littlestone's analysis of Winnow (Littlestone, 1989b), which also employs a variant of the relative entropy. Amari's (1994, 1995) approach in using the relative entropy for deriving neural network learning algorithms is similar to the first use we have here for the distance measure. The distance term in the minimized function is also somewhat analogous to regularization terms used in neural network algorithms to avoid overfitting (Haykin, 1994).

We now discuss the actual worst-case bounds we can obtain for the GD and EG$^{\pm}$ algorithms. For the GD algorithm, the bounds we cite were already given by Cesa-Bianchi *et al.* (1996). These include bounds of both the forms (1.1) and (1.2). For the EG$^{\pm}$ algorithm, we give new bounds that are strictly better than those obtained by Littlestone *et al.* (1995) for their algorithm. In particular, we also have bounds of the form (1.2), whereas Littlestone *et al.* (1995) had only bounds of the form (1.1). The importance of considering both the algorithms GD and EG$^{\pm}$ comes from the fact that for these algorithms, the constants hidden by the notation in (1.1) and (1.2) are quite different. To state the exact bounds, recall that for positive p, the L_p norm for vectors $\mathbf{x} \in \mathbf{R}^N$ is defined by $\|\mathbf{x}\|_p = (|x_1|^p + \cdots + |x_N|^p)^{1/p}$. This is generalized for $p = \infty$ by setting $\|\mathbf{x}\|_\infty = \max_i |x_i|$. All the bounds that follow hold only for the square loss, and we omit mentioning the loss function in them.

Assume now that a trial sequence S satisfies $\|\mathbf{x}_t\|_2 \leqslant X$, where X is a known constant, but let S be otherwise arbitrary. For the GD algorithm, setting the learning rate η suitably results in the bound

$$\mathrm{Loss}(\mathrm{GD}, S) \leqslant 2(\mathrm{Loss}(\mathbf{u}, S) + \|\mathbf{u}\|_2^2 X^2)$$

that holds for all vectors $\mathbf{u} \in \mathbf{R}^N$ (Cesa-Bianchi *et al.*, 1996). To make the coefficient in front of $\text{Loss}(\mathbf{u}, S)$ equal to 1 and thus obtain a bound of the form (1.2), the algorithm needs before the first trial reasonably good estimates of some characteristics of the whole trial sequence. These estimates help the algorithm to set the learning rate η. In addition to the bound X, the algorithm need bounds K and U, such that some vector $\mathbf{u}$ with L_2 norm at most U has loss at most K. If the algorithm is given before the first trial any values for the parameters K, U, and X, then the bound

$$\text{Loss}(\text{GD}, S) \leqslant \text{Loss}(\mathbf{u}, S) + 2\sqrt{K}\,UX + \|\mathbf{u}\|_2^2\,X^2 \tag{1.3}$$

holds for all weight vectors $\mathbf{u}$ and trial sequences S such that $\text{Loss}(\mathbf{u}, S) \leqslant K$ and $\|\mathbf{u}\|_2 \leqslant U$ hold and $\|\mathbf{x}_t\| \leqslant X$ holds for all t. If the parameters K, U, and X are given too low values, the bound (1.3) can become vacuous because the conditions for $\mathbf{u}$ and S are not satisfied for any $\mathbf{u}$. On the other hand if the parameters are overly conservative, then the bound also becomes very loose. If it is not possible to obtain satisfactory values for all the parameters before the first trial it is in some cases possible to apply an iterative scheme for obtaining increasingly accurate estimates for some of the mas the trial sequence proceeds. This leads to abound that is similar to (1.3) but has slightly larger constant coefficients (Cesa-Bianchi *et al.*, 1996).

For the $\text{EG}^{\pm}$ algorithm, it is necessary to give as a parameter an upper bound U for the L_1 norm of the vectors $\mathbf{u}$ of the comparison class. Assuming now that the instances of the trial sequence S have a bounded L_∞ norm $\|\mathbf{x}_t\|_\infty \leqslant X$ for some known constant X, we have the bound

$$\text{Loss}(\text{EG}^{\pm}, S) \leqslant 3(\text{Loss}(\mathbf{u}, S) + U^2X^2 \ln 2N)$$

that holds for all $\mathbf{u} \in \mathbf{R}^N$ such that $\|\mathbf{u}\|_1 \leqslant U$. As with the GD algorithm, additional knowledge of the trial sequence helps the algorithm to choose the learning rate η more accurately. If, in addition to U and X, the algorithm is given a parameter K, then it achieves the bound

$$\text{Loss}(\text{EG}^{\pm}, S) \leqslant \text{Loss}(\mathbf{u}, S) + 2\sqrt{K \ln 2N}\,UX + 2U^2X^2 \ln 2N \tag{1.4}$$

for all comparison vectors $\mathbf{u}$ and trial sequences S such that $\|\mathbf{u}\|_1 \leqslant U$ and $\text{Loss}(\mathbf{u}, S) \leqslant K$ hold and $\|\mathbf{x}_t\|_\infty \leqslant X$ holds for all t.

Note that L_p and L_q are *dual norms* if $1/p + 1/q = 1$ (Royden, 1963). Hence, the L_1 norm used for the comparison vectors and the L_∞ norm used for the instances in the bounds for the $\text{EG}^{\pm}$ algorithm are dual. The norm L_2, used for both the comparison vectors and the instances in the bounds for the GD algorithm, is its own dual. We now show that the different pairs of dual norms in the upper bounds for the GD and the $\text{EG}^{\pm}$ algorithms result in certain situations in radically different behavior for large N. For simplicity, we consider the case in which there is a perfect linear relation between the instances and outcomes, and therefore some comparison vector $\mathbf{u}$ satisfies $\text{Loss}(\mathbf{u}, S) = 0$. We can then take $K = 0$ in the bounds (1.3) and (1.4). Assume that all the other parameters are also set optimally, and write

$X_p = \max_t \|\mathbf{x}_t\|_p$ for $p = 2$ and $p = \infty$. Then the bound (1.3) simplifies to $\text{Loss}(\text{GD}, S) \leqslant \|\mathbf{u}\|_2^2 X_2^2$ and the bound (1.4) to $\text{Loss}(\text{EG}^{\pm}, S) \leqslant 2 \|\mathbf{u}\|_1^2 X_\infty^2 \ln 2N$.

For clarity, we consider two extreme cases. First, assume that $\mathbf{u}$ has exactly k components with value 1 and the rest $N-k$ components have value 0. Thus, only k input variables are relevant for the prediction task. Assume that the instances $\mathbf{x}_t$ are from the set $\{-1, 1\}^N$ of vertices of an N-dimensional cube. Then $\|\mathbf{u}\|_2 = \sqrt{k}$, $\|\mathbf{u}\|_1 = k$, $X_2 = \sqrt{N}$, and $X_\infty = 1$. The bounds become $\text{Loss}(\text{GD}, S) \leqslant kN$ and $\text{Loss}(\text{EG}^{\pm}, S) \leqslant 2k^2 \ln 2N$, so for $N \gg k$ the $\text{EG}^{\pm}$ algorithm has clearly the better bound. On the other hand, let $\mathbf{u} = (1, ..., 1)$, and let the instances be rows of the $N \times N$ unit matrix. Then $\|\mathbf{u}\|_2 = \sqrt{N}$, $\|\mathbf{u}\|_1 = N$, and $X_2 = X_\infty = 1$. The bounds become $\text{Loss}(\text{GD}, S) \leqslant N$ and $\text{Loss}(\text{EG}^{\pm}, S) \leqslant N^2 \ln 2N$, so the GD algorithm has clearly the better bound. Thus, the bounds for GD and $\text{EG}^{\pm}$ are incomparable, and for large N the difference can be arbitrarily large in either direction.

The simplified scenario given above can be generalized. If only few of the input variables are relevant for predicting the outcomes, but all the input variables take values of roughly equal magnitudes, then the $\text{EG}^{\pm}$ algorithm has the better bound. The GD algorithm has the better bound if all the input variables are almost equally relevant for predicting, and the L_2 norms of the instances are not much larger than the L_∞ norms. This happens if most of the total weight in the instance vectors is concentrated on the largest components. The conclusions remain similar also when no comparison vector $\mathbf{u}$ achieves $\text{Loss}(\mathbf{u}, S) = 0$, which is the case if there is noise in the instances or outcomes. However, the differences between the total losses of the algorithms become less pronounced in these less pure situation.

While the preceding comparison is based purely on worst-case bounds, the conclusions about the relative merits of the algorithms are confirmed by experiments on simple artificial data. This is true both with and without noise in the outcomes. In the experiments we have also seen that the learning rates suggested by our worst-case upper bound analysis are quite close to the optimal ones.

In particular, we have observed that the number of examples the GD algorithm needs before it obtains an accurate hypothesis is roughly comparable to the number N of input variables, even if almost all of the input variables are irrelevant for the prediction task. For the $\text{EG}^{\pm}$ algorithm, the dependence on the number of irrelevant input variables is only logarithmic, so doubling the number of irrelevant variables results in only a constant increase in the total loss. It seems that the $\text{EG}^{\pm}$ algorithm has a strong bias for hypotheses with few relevant variables. Thus, if only few variables are needed for prediction, then the loss bound of $\text{EG}^{\pm}$ grows sublinearly in the number N of the variables. The GD algorithm is biased towards hypotheses with small L_2 norm, and even if only few variables are relevant, it uses all the dimensions in a futile search for a good predictor with a small norm.

We feel that the situation that favors the $\text{EG}^{\pm}$ algorithm is much more natural and likely to arise in practice. Since linear predictors are very restricted, a natural extension would be to expand the instance $\mathbf{x}_t$ by including as new input variables the values $f_i(\mathbf{x}_t)$ for some suitably chosen basis functions f_i. Then a linear prediction algorithm could actually use a linear combination of the basis functions as its predictor. As an example, we might include all the $O(N^q)$ products of up to q original input variables (Boser *et al.*, 1992). Assuming that the input variables are

in the range $[-1, 1]$, this does not increase the L_∞ norms of the instances. Assume further that the outcomes are actually given by some degree q polynomial of the input variables, with k terms that each have a constant coefficient of at most 1. Then the loss bound for the EG$^\pm$ algorithm after the expansion of the instances would be $O(k^2 q \log N)$. However, the GD algorithm would suffer from the fact that the expansion increases the L_2 norms of the instances, and could have a loss $O(kN^q)$. Unfortunately, expanding the instances increases the amount of computations needed in the predictions and updates.

Worst-case upper bounds have become a powerful tool in analyzing simple learning problems. The learning algorithms GD and EG$^\pm$ can be directly applied to feed-forward neural networks. For the GD algorithm, this leads to the back-propagation algorithm. From the EG$^\pm$ algorithm we obtain a neural network algorithm that uses the same gradient information as the back-propagation algorithm, but applies it in a radically different manner. We expect that some of the differences in the behavior of the GD and EG$^\pm$ algorithms for a linear neuron carry over to feed-forward neural networks, but it seems unlikely that one could prove worst-case bounds in this more complicated setting. For single sigmoided neurons, worst-case bounds have been obtained recently (Helmbold *et al.*, 1995a).

We define the basic notation in Section 2. Our main algorithms are introduced in Section 3, and their derivations using the various distance measures are given in Section 4. In Section 5, we prove our worst-case upper bounds for the losses of the algorithms. Both Section 4 and Section 5 begin with a high-level description of our approach, after which the more technical application of the ideas for the various algorithms follows. Section 6 gives some related lower bound results. In Section 7, we show how the algorithm and their upper bound proofs can be modified for a generalized scenario, in which the algorithm is required to make several predictions at once. Section 8 contains a brief discussion on converting our worst-case total loss bounds for expected instantaneous loss bounds. Our experimental comparisons of the algorithms are described in Section 9.

To quickly get an idea of our main results, the reader can skim through the definitions in Section 2 and the descriptions of the algorithms in Section 3, and then go to Section 9 for the comparison of our theoretical and empirical results for the different algorithms. Section 4 is important for gaining intuition about the algorithms. The most important theoretical results are given in Section 5.

2. PRELIMINARIES

On-line prediction algorithms function as follows. In trial t, for $t = 1, 2, ...$, the algorithm first receives an *instance* $\mathbf{x}_t \in \mathbf{R}^N$. After producing a *prediction* $\hat{y}_t \in \mathbf{R}$, the algorithm receives an *outcome* $y_t \in \mathbf{R}$ as feedback. The performance of the algorithm at trial t is measured in terms of a loss function L that assigns a nonnegative real loss $L(y, \hat{y})$ to each possible outcome–prediction pair $(y, \hat{y})$ and has the property $L(y, y) = 0$ for $y \in \mathbf{R}$. For a more compact notation, we define $L_y(\hat{y}) = L(y, \hat{y})$. In particular, we write $L'_y(\hat{y})$ for $(\partial L(y, z)/\partial z)_{z=\hat{y}}$ when $\hat{y}$ is some given fixed value. Our default loss function is the *square loss*, i.e., $L(y, \hat{y}) = (y - \hat{y})^2$. Another

commonly used loss function, for predictions and outcomes in the interval $[0, 1]$, is the *entropic loss function*: $L(y, \hat{y}) = y \ln(y/\hat{y}) + (1-y) \ln((1-y)/(1-\hat{y}))$. Here we follow the standard convention $0 \ln 0 = 0$.

Technically, an *on-line prediction algorithm* is a mapping A that maps a sequence $(\mathbf{x}_i, y_i)$, $i = 1, ..., t-1$, of instance–outcome pairs and a new instance $\mathbf{x}_t$ into a prediction $\hat{y}_t = A((\mathbf{x}_1, y_1), ..., (\mathbf{x}_{t-1}, y_{t-1}), \mathbf{x}_t)$. In this paper we only consider on-line prediction algorithms that represent all the information they retain from the trials $1, ..., t-1$ by a weight vector $\mathbf{w}_t$ and predict with $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$. Then the weight vector can be considered as the algorithm's linear hypothesis. The initial weight vector $\mathbf{w}_1$ is a parameter of the algorithm. At the end of each trial the algorithm updates its previous weight vector $\mathbf{w}_t$ into $\mathbf{w}_{t+1}$, taking into account the instance $\mathbf{x}_t$ and the outcome y_t as well as $\mathbf{w}_t$. We will discuss various update rules in this paper.

The total loss of A on a sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ is

$$\mathrm{Loss}_L(A, S) = \sum_{t=1}^{\ell} L(y_t, \hat{y}_t).$$

We also define a total loss for a weight vector $\mathbf{u} \in \mathbf{R}^N$ by

$$\mathrm{Loss}_L(\mathbf{u}, S) = \sum_{t=1}^{\ell} L(y_t, \mathbf{u} \cdot \mathbf{x}_t).$$

We omit the subscript L when we use the square loss $L(y - \hat{y}) = (y - \hat{y})^2$.

Our goal is to have algorithms for which the loss $\mathrm{Loss}_L(A, S)$ is low for all possible trial sequences. Obviously, without some knowledge of the trial sequence S, we cannot give any useful guarantees about $\mathrm{Loss}_L(A, S)$. To set a reasonable goal for the algorithms, we consider the loss $\inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S)$ of the best linear hypothesis in some class $\mathscr{U} \subseteq \mathbf{R}^N$ of weight vectors. The quantity

$$\mathrm{Loss}_L(A, S) - \inf_{\mathbf{u} \in \mathscr{U}} \mathrm{Loss}_L(\mathbf{u}, S) \tag{2.1}$$

is then the additional loss of the algorithm compared to the weight vectors of the class $\mathscr{U}$. We seek algorithms with provable upper bounds on the additional loss that hold for arbitrary sequences S. We call the set $\mathscr{U}$ the comparison class and the vectors $\mathbf{u} \in \mathscr{U}$ comparison vectors. We sometimes call a particular comparison vector $\mathbf{u} \in \mathscr{U}$ that has a small loss a target vector. To prove bounds for the additional loss (2.1), we usually need to make some assumptions about the norms of the comparison vectors $\mathbf{u} \in \mathscr{U}$, as well as about the instances $\mathbf{x}_t$ that appear in the trial sequence S. Observe that the infimum measures how well a linear model can do when whole sequence is given in a advance. The on-line learner only sees one example at a time and the additional loss over the infimum measures the price the algorithm has to pay for not seeing the whole sequence in advance.

For any positive real p, the L_p norm for vectors $\mathbf{x} \in \mathbf{R}^N$ is defined by $\|\mathbf{x}\|_p = (|x_1|^p + \cdots + |x_N|^p)^{1/p}$. For the case $p = 1$ we have $\|\mathbf{x}\|_1 = \sum_{i=1}^N |x_i|$,

and for case $p=2$ we have the Euclidean length $\|\mathbf{x}\|_2=\sqrt{\sum_{i=1}^N x_i^2}$ of the vector $\mathbf{x}$. Besides these two norms, we also use the L_∞ norm $\|\mathbf{x}\|_\infty=\max_{i=1}^N |x_i|$, which is obtained as a limit in the definition of L_p when p approaches ∞.

We have various measures of distance between two weight vectors $\mathbf{u}$ and $\mathbf{w}$. In applying the distance measures to be presented shortly, we usually take as $\mathbf{u}$ some comparison vector and as $\mathbf{w}$ the hypothesis of the algorithm at some trial. In general, a distance measure d is any function mapping $\mathbf{R}^N\times\mathbf{R}^N$ to the nonnegative reals in such a way that $d(\mathbf{u},\mathbf{u})=0$ holds for all $\mathbf{u}\in\mathbf{R}^N$. The most basic of our distance measures is the *squared Euclidean distance* d_{sq} defined by $d_{\mathrm{sq}}(\mathbf{u},\mathbf{w})=\frac{1}{2}\|\mathbf{u}-\mathbf{w}\|_2^2$. Some distance measures we use are only defined if $\mathbf{u}$ and $\mathbf{w}$ are in a particular subset of $\mathbf{R}^N$. The *relative entropy* is a distance measure that is only defined when both vector are *probability vectors*, which means that their components are nonnegative and sum to 1. For two probability vectors $\mathbf{u}$ and $\mathbf{w}$, the relative entropy $d_{\mathrm{re}}(\mathbf{u},\mathbf{w})$ is defined by

$$d_{\mathrm{re}}(\mathbf{u},\mathbf{w})=\sum_{i=1}^N u_i\ln\frac{u_i}{w_i}.$$

Note that we allow the components to be zero, and for that case the usual convention $0\ln 0=0$ is used. If we have $w_i=0$ and $u_i>0$ for some i, then $d_{\mathrm{re}}(\mathbf{u},\mathbf{w})=\infty$. It can be shown that $d_{\mathrm{re}}(\mathbf{u},\mathbf{w})$ is always nonnegative, and 0 only if $\mathbf{u}=\mathbf{w}$.

If $\mathbf{w}=(1/N,\ldots,1/N)$ is the uniform probability vector, then for all probability vectors $\mathbf{u}\in[0,1]^N$ we have $d_{\mathrm{re}}(\mathbf{u},\mathbf{w})=\ln N-H(\mathbf{u})$, where the quantity $H(\mathbf{u})=-\sum_{i=1}^N u_i\ln u_i$ is called the *entropy* of the weight vector $\mathbf{u}$. Every probability vector $\mathbf{u}$ satisfies $0\leqslant H(\mathbf{u})\leqslant\ln N$ and, hence, $d_{\mathrm{re}}(\mathbf{u},\mathbf{w})\leqslant\ln N$ for the uniform vector $\mathbf{w}$.

We now generalize the relative entropy by removing the requirement that the components of $\mathbf{u}$ and $\mathbf{w}$ sum to 1 but still keeping the requirement that the components of both vectors are nonnegative. We define the unnormalized relative entropy $d_{\mathrm{reu}}(\mathbf{u},\mathbf{w})$ for all $\mathbf{u}$ and $\mathbf{w}$ in $[0,\infty)^N$ by

$$d_{\mathrm{reu}}(\mathbf{u},\mathbf{w})=\sum_{i=1}^N\left(w_i-u_i+u_i\ln\frac{u_i}{w_i}\right).$$

Note that $d_{\mathrm{reu}}(\mathbf{u},\mathbf{w})=d_{\mathrm{re}}(\mathbf{u},\mathbf{w})$ holds when both $\mathbf{u}$ and $\mathbf{w}$ are probability vectors. It is easy to see that $d_{\mathrm{reu}}(\mathbf{u},\mathbf{w})\geqslant 0$ holds for all vectors $\mathbf{u}$ and $\mathbf{w}$ in $[0,\infty)^N$ and equality holds only if $\mathbf{u}=\mathbf{w}$.

We also consider the distance measure $d_{\chi^2}(\mathbf{u},\mathbf{w})$ defined by

$$d_{\chi^2}(\mathbf{u},\mathbf{w})=\frac{1}{2}\sum_{i=1}^N\frac{(u_i-w_i)^2}{w_i}=\frac{1}{2}\left(\sum_{i=1}^N\frac{u_i^2}{w_i}-1\right),$$

where the second equality is based on assuming $\sum_{i=1}^N u_i=\sum_{i=1}^N w_i=1$. Because the function f given by $f(u)=w-u+u\ln(u/w)$ has around $u=w$ the second-order Taylor expansion $f(u)=(u-w)^2/(2w)+O((u-w)^3)$, the measure $d_{\chi^2}(\mathbf{u},\mathbf{w})$ can be considered an approximation for the measure $d_{\mathrm{reu}}(\mathbf{u},\mathbf{w})$. Since $\ln x\leqslant x-1$, it is also easy to see that $d_{\mathrm{re}}(\mathbf{u},\mathbf{w})\leqslant 2d_{\chi^2}(\mathbf{u},\mathbf{w})$ holds for probability vectors $\mathbf{u}$ and $\mathbf{w}$.

Note that none of the distance measures discussed above satisfies the triangle inequality, and with the exception of the squared Euclidean distance d_{sq} the distance measures are not symmetric. For the squared Euclidean distance d_{sq}, we clearly have $d_{\mathrm{sq}}(\mathbf{u}, \mathbf{u}) \geqslant 0$ for all $\mathbf{u}$, and $d_{\mathrm{sq}}(\mathbf{u}, \mathbf{w}) > 0$ for all $\mathbf{w} \neq \mathbf{u}$. These properties also hold for the distance measures d_{reu} and d_{χ^2} if the vectors $\mathbf{w}$ and $\mathbf{u}$ are restricted to have only nonnegative components, and for d_{re} if $\mathbf{u}$ and $\mathbf{w}$ are restricted to be probability vectors. See Helmbold *et al.* (1996b) for some plots that visualize the distance measures for probability vectors in the three-dimensional case.

3. THE MAIN ALGORITHMS

In this section, we introduce the main on-line prediction algorithms we consider in this paper. In Section 4, we give a motivation that shows how each of the algorithms naturally arises from an approximate solution to a certain minimization problem. A number of additional algorithms are also introduced in Section 4.

All the algorithms share the same basic structure. The algorithm maintains a weight vector, which can be considered as the algorithm's guess of a good linear predictor. We use $\mathbf{w}_t$ to denote the weight vector of an algorithm before trial t. The weight vector $\mathbf{w}_t$ contains the only information the algorithm retains about the trials $1, \ldots, t-1$. The algorithm starts by setting the initial weight vector $\mathbf{w}_1$ to be some *start vector* $\mathbf{s}$. After seeing the tth outcome y_t, the algorithm updates its weight vector to $\mathbf{w}_{t+1}$ according to its *update rule*. The value of the new weight vector $\mathbf{w}_{t+1}$ depends on the old weight vector $\mathbf{w}_t$, the instance $\mathbf{x}_t$, the prediction $\hat{y}_t$, the outcome y_t, and a *learning rate* η. The exact dependence is called the *update rule* of the algorithm. The difference between our various algorithms is that they use different update rules. The learning rate η may be different at different trials, but here we usually keep it fixed. The prediction $\hat{y}_t$ after seeing the instance $\mathbf{x}_t$ at trial t is given by $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$ for all the algorithms. As a small exception to this, for some of the algorithms the predictions are restricted into a fixed interval, and if the value $\mathbf{w}_t \cdot \mathbf{x}_t$ fall outside this interval, the prediction will be the closest value inside the interval.

Figure 1 gives the algorithm which we call the gradient descent algorithm and denote by GD_L. Recall that $L'_{y_t}(\hat{y}_t) = (\partial L(y_t, z)/\partial z)_{z=\hat{y}_t}$. Notice that the ith component of the gradient $\nabla_{\mathbf{w}_t} L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t)$ is given by $\partial L(y_t, \mathbf{w} \cdot \mathbf{x})/\partial w_i = (\partial L(y_t, z)/\partial z)_{z=\mathbf{w} \cdot \mathbf{x}} x_i = L'_{y_t}(\mathbf{w} \cdot \mathbf{x}) x_i$. Thus, the gradient descent algorithm updates the weight vector by subtracting from it the gradient $\nabla_{\mathbf{w}_t} L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t)$ multiplied by the scalar η. The GD_L algorithm can therefore be seen as a straightforward application of the usual gradient descent minimization method to the on-line prediction problem. We let $\mathrm{GD}(\mathbf{s}, \eta)$ denote the algorithm $\mathrm{GD}_L(\mathbf{s}, \eta)$ for the case when the loss function L is the square loss function given by $L(y, \hat{y}) = (y - \hat{y})^2$. The algorithm $\mathrm{GD}(\mathbf{s}, \eta)$ has many names, including the Widrow–Hoff algorithm and the Least Mean Square (LMS) algorithm (Cesa-Bianchi *et al.*, 1996; Widrow and Stearns, 1985). The update for $\mathrm{GD}(\mathbf{s}, \eta)$ is simply

$$\mathbf{w}_{t+1} = \mathbf{w}_t - 2\eta(\hat{y}_t - y_t)\,\mathbf{x}_t. \tag{3.1}$$

Algorithm $\mathrm{GD}_L(\mathbf{s}, \eta)$

Parameters:

L: a loss function from $\mathbf{R} \times \mathbf{R}$ to $[0, \infty)$,

$\mathbf{s}$: a start vector in $\mathbf{R}^N$, and

η: a learning rate in $[0, \infty)$.

Initialization: Before the first trial, set $\mathbf{w}_1 = \mathbf{s}$.

Prediction: Upon receiving the tth instance $\mathbf{x}_t$, give the prediction $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$.

Update: Upon receiving the tth outcome y_t, update the weights according to the rule

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta L'_{y_t}(\hat{y}_t)\mathbf{x}_t.$$

FIG. 1. The gradient descent algorithm $\mathrm{GD}_L(\mathbf{s}, \eta)$.

The start vector $\mathbf{s}$ of the algorithm can be arbitrary. Typically one would choose $\mathbf{s} = \mathbf{0}$. As the trial sequence proceeds, the individual weights $w_{t,i}$ can reach arbitrarily high and low values. A typical learning rate could be $\eta = 1/(4X^2)$ where X is an estimated upper bound for the largest L_2 norm $\max_t \|\mathbf{x}_t\|_2$ of the instances. Later, in Theorem 5.3, we give some theoretical results about the proper choice of η and the resulting performance of GD. As a general heuristic, the learning rate should be low if it is expected, for example because of noise, that there is no linear predictor $\mathbf{u}$ for which $\mathrm{Loss}_L(\mathbf{u}, S)$ is low. The learning rate should be high if it is expected that for all good linear predictors $\mathbf{u}$, the distance $\|\mathbf{u} - \mathbf{s}\|_2$ from the start vector is large.

We also consider one particular method of letting the learning rate of GD vary between trials. Thus, we use the name $\mathrm{GDV}(\mathbf{s}, \eta)$ for the algorithm that is as $\mathrm{GD}(\mathbf{s}, \eta)$ except that the update (3.1) is replaced by

$$\mathbf{w}_{t+1} = \mathbf{w}_t - 2\frac{\eta}{\|\mathbf{x}_t\|_2^2}(\hat{y}_t - y_t)\mathbf{x}_t, \tag{3.2}$$

assuming $\|\mathbf{x}_t\|_2 > 0$. If $\|\mathbf{x}_t\|_2 = 0$, the algorithm makes no update. The variable learning rate algorithm GDV is of particular interest when it is assumed that $y_t = \mathbf{u} \cdot \mathbf{x}_t$ holds for some fixed unknown vector $\mathbf{u}$.

We now turn to the two main new algorithms of this paper. The first, and simpler, of these is given in Fig. 2. We call it the exponentiated gradient algorithm, or EG_L. In the update of EG_L, each weight is multiplied by a factor $r_{t,i}$ that according to (3.4) is obtained by *exponentiating* the ith component $\partial L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t)/\partial w_{t,i} = L'_{y_t}(\mathbf{w}_t \cdot \mathbf{x}_t)x_i$ of the gradient $\nabla_{\mathbf{w}_t} L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t)$. After this multiplication, the weights are normalized, as shown in (3.3), so that they sum to 1. The weights clearly never change sign. Hence, the weight vector $\mathbf{w}_t$ of EG_L is always a probability vector, i.e., it satisfies $\sum_i w_{t,i} = 1$ and $w_{t,i} \geqslant 0$ for all i. Therefore, the prediction $\mathbf{w}_t \cdot \mathbf{x}_t$ is a weighted average of the input variables $x_{t,i}$, and $\mathbf{w}_t$ gives the relative weights of the components in this weighted average. This is in contrast to the GD_L algorithm,

Algorithm $\mathrm{EG}_L(\mathbf{s}, \eta)$

Parameters:

L: a loss function from $\mathbf{R} \times \mathbf{R}$ to $[0, \infty)$,

$\mathbf{s}$: a start vector, with $\sum_{i=1}^{N} s_i = 1$ and $s_i \geqslant 0$ for all i, and

η: a learning rate in $[0, \infty)$.

Initialization: Before the first trial, set $\mathbf{w}_1 = \mathbf{s}$.

Prediction: Upon receiving the tth instance $\mathbf{x}_t$, give the prediction $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$.

Update: Upon receiving the tth outcome y_t, update the weights according to the rule

$$w_{t+1,i} = \frac{w_{t,i} r_{t,i}}{\sum_{j=1}^{N} w_{t,j} r_{t,j}}. \tag{3.3}$$

where

$$r_{t,i} = \exp(-\eta L'_{y_t}(\hat{y}_t)\, x_{t,i}). \tag{3.4}$$

FIG. 2. The exponentiated gradient algorithm $\mathrm{EG}_L(\mathbf{s}, \eta)$.

where also the total weight $\|\mathbf{w}_t\|_1$ can change. The fact that the weight vector is always a probability vector clearly restricts the abilities of EG_L to learn more general linear relationships. We shall soon see how these restrictions can be avoided by a simple reduction.

As the GD_L algorithm, the EG_L algorithm has a loss function, start vector, and a learning rate as its parameters. Again, $L'_{y_t}(\hat{y}_t) = (\partial L(y_t, z)/\partial z)_{z=\hat{y}_t}$. If the loss function L is the square loss function, we denote $\mathrm{EG}_L(\mathbf{s}, \eta)$ simply by $\mathrm{EG}(\mathbf{s}, \eta)$. For the square loss, (3.4) becomes

$$r_{t,i} = e^{-2\eta(\hat{y}_t - y_t)\, x_{t,i}}.$$

We assume that the start vector $\mathbf{s}$ also satisfies $\sum_i s_i = 1$ and $s_i \geqslant 0$ for all i. The usual choice for $\mathbf{s}$ is the uniform probability vector $(1/N, ..., 1/N)$. A typical learning rate could be $\eta = 2/(3R^2)$, where R is an upper bound for the maximum difference $\max_t(\max_i x_{t,i} - \min_i x_{t,i})$ between the components $x_{t,i}$ of an instance $\mathbf{x}_t$. Analogously to the GD algorithm, the EG algorithm should have a low learning rate if no probability vector $\mathbf{u}$ is assumed to be a good predictor, and a high learning rate if it is assumed that for the good predictors $\mathbf{u}$ the distance $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})$ from the start vector is large. If $\mathbf{s}$ is the uniform vector $(1/N, ..., 1/N)$, then the distance is largest for nonuniform vectors $\mathbf{u}$, having its maximum value $\ln N$ when for some i the vector $\mathbf{u}$ has $u_i = 1$ and $u_j = 0$ for $j \neq i$. More detailed results about the choice of the learning rate and the resulting total loss of EG are given in Theorem 5.10.

Before proceeding to our second main algorithm, we give a simplified alternative for the update rule of the EG_L algorithm. The alternative has the benefit of avoiding the use of the exponential function and thus possibly saving some computation

time. We use the name *approximated* EG_L for the algorithm obtained from the EG_L algorithm by replacing the update (3.3) by

$$w_{t+1,i} = w_{t,i}(1 - \eta L'_{y_t}(\hat{y}_t)(x_{t,i} - \hat{y}_t)). \tag{3.5}$$

To see how the approximated update (3.5) arises, note that the first order Taylor approximation of e^{-av} for v close to v_0 is given by $e^{-av} \approx e^{-av_0}(1 - a(v - v_0))$. By replacing the exponentials on the right hand side of (3.3) by this approximation, with $a = \eta L'_{y_t}(\hat{y}_t)$, $v = x_{t,i}$, and $v_0 = \hat{y}_t$, we obtain

$$w_{t+1,i} = \frac{w_{t,i} e^{-a\hat{y}_t}(1 - a(x_{t,i} - \hat{y}_t))}{\sum_{i=1}^{N} w_{t,j} e^{-a\hat{y}_t}(1 - a(x_{t,j} - \hat{y}_t))}. \tag{3.6}$$

We get (3.5) from (3.6) by noticing that $\sum_{i=1}^{N} w_{t,j} = 1$ and $\sum_{i=1}^{N} w_{t,j}(x_{t,j} - \hat{y}_t) = 0$, which makes the denominator on the right hand side of (3.6) equal to $e^{-a\hat{y}_t}$. Admittedly, it may seem somewhat arbitrary to use $\hat{y}_t$ as the center of the Taylor approximation for $e^{-ax_{t,i}}$. However, we shall see in Subsection 4.4 that the update rule (3.5) also has another motivation that is not based on approximating (3.3). It has also been noticed that in applying the exponentiated gradient update to a certain unsupervised learning problem (Helmbold *et al.*, 1996b, 1996c) the approximation given here leads to a generalization of the Expectation Maximization algorithm (Dempster *et al.*, 1977).

Note that the update rule (3.5) maintains the invariant $\sum_{i=1}^{N} w_{t,j} = 1$. However, it may make some of the weights $w_{t+1,i}$ zero or negative. A weight that once gets set to 0 can never recover because of the multiplicative nature of the update, and this might be a problem. One way to ensure that the weights remain positive after the update (3.5) is to enforce that the learning rate satisfies

$$\eta < (L'_{y_t}(\hat{y}_t)(x_{t,i} - \hat{y}_t))^{-1} \tag{3.7}$$

for all indices i for which the quantity on the right − hand side of (3.7) is positive. In some very preliminary experiments we have performed it seems that the performance of the approximated EG is hardly distinguishable from that of the unapproximated EG, and that there is no real problem with weights going to zero. However, difficulties may arise in more complicated situations.

The second new algorithm, which we call the exponentiated gradient algorithm with positive and negative weights, or $\mathrm{EG}_L^{\pm}$, is given in Fig. 3. The $\mathrm{EG}_L^{\pm}$ algorithm can best be understood as a way to generalize the EG_L algorithm for more general weight vectors by using a reduction. Given a trial sequence S, let S' be a modified trial sequence obtained from S by replacing each instance $\mathbf{x}_t$ by $\mathbf{x}'_t = (Ux_1, ..., Ux_N, -Ux_1, ..., -Ux_N)$. Hence, the number of dimensions is doubled. For a start vector pair $(\mathbf{s}^+, \mathbf{s}^-)$ for $\mathrm{EG}_L^{\pm}$, let $\mathbf{s} = (s_1^+, ..., s_N^+, s_1^-, ..., s_N^-)$. Consider using $\mathrm{EG}_L^{\pm}(U, (\mathbf{s}^+, \mathbf{s}^-), \eta)$ on a trial sequence S and using $\mathrm{EG}_L(\mathbf{s}, \eta)$ on the modified trial sequence S'. If we let $\mathbf{w}'_t$ be the tth weight vector of $\mathrm{EG}_L(\mathbf{s}, \eta)$ on the trial sequence S', it is easy to see that $U\mathbf{w}'_t = (w_{t,1}^+, ..., w_{t,N}^+, w_{t,1}^-, ..., w_{t,N}^-)$ holds for all t and,

Algorithm $\mathrm{EG}_L^{\pm}(U, (\mathbf{s}^+, \mathbf{s}^-), \eta)$

Parameters:

L: a loss function from $\mathbf{R} \times \mathbf{R}$ to $[0, \infty)$,

U: the total weight of the weight vectors,

$\mathbf{s}^+$ and $\mathbf{s}^-$: a pair of start vectors in $[0, 1]^N$, with $\sum_{i=1}^N (s_i^+ + s_i^-) = 1$, and

η: a learning rate in $[0, \infty)$.

Initialization: Before the first trial, set $\mathbf{w}_1^+ = U\mathbf{s}^+$ and $\mathbf{w}_1^- = U\mathbf{s}^-$.

Prediction: Upon receiving the tth instance $\mathbf{x}_t$, give the prediction

$$\hat{y}_t = (\mathbf{w}_t^+ - \mathbf{w}_t^-) \cdot \mathbf{x}_t.$$

Update: Upon receiving the tth outcome y_t, update the weights according to the rules

$$w_{t+1,i}^+ = U \cdot \frac{w_{t,i}^+ r_{t,i}^+}{\sum_{j=1}^N (w_{t,j}^+ r_{t,j}^+ + w_{t,j}^- r_{t,j}^-)} \tag{3.8}$$

$$w_{t+1,i}^- = U \cdot \frac{w_{t,i}^- r_{t,i}^-}{\sum_{j=1}^N (w_{t,j}^+ r_{t,j}^+ + w_{t,j}^- r_{t,j}^-)}, \tag{3.9}$$

where

$$r_{t,i}^+ = \exp(-\eta L'_{y_t}(\hat{y}_t)\, U x_{t,i}) \tag{3.10}$$

$$r_{t,i}^- = \exp(\eta L'_{y_t}(\hat{y}_t)\, U x_{t,i}) = \frac{1}{r_{t,i}^+} \tag{3.11}$$

FIG. 3. Exponential gradient algorithm with positive and negative weights $\mathrm{EG}_L^{\pm}(U, (\mathbf{s}^+, \mathbf{s}^-), \eta)$.

therefore, $\mathbf{w}'_t \cdot \mathbf{x}'_t = (\mathbf{w}_t^+ - \mathbf{w}_t^-) \cdot \mathbf{x}_t$. Hence, the predictions of $\mathrm{EG}_L^{\pm}$ on S and EG_L on S' are identical, so $\mathrm{EG}_L^{\pm}$ is a result of applying a simple transformation to EG_L. This transformation leads to an algorithm that in effect uses a weight vector $\mathbf{w}_t^+ - \mathbf{w}_t^-$, which can contain negative components. Further, by using the scaling factor U, we can make the weight vector $\mathbf{w}_t^+ - \mathbf{w}_t^-$ range over all vectors $\mathbf{w} \in \mathbf{R}$ for which $\|\mathbf{w}\|_1 \leqslant U$. Although $\|\mathbf{w}_t^+\|_1 + \|\mathbf{w}_t^-\|_1$ is always exactly U, vectors $\mathbf{w}_t^+ - \mathbf{w}_t^-$ with $\|\mathbf{w}_t^+ - \mathbf{w}_t^-\|_1 < U$ result simply from having both $w_{t,i}^+ > 0$ and $w_{t,i}^- > 0$ for some i. For other examples of reductions of this type, see Littlestone *et al.* (1995).

The parameters of $\mathrm{EG}_L^{\pm}$ are a loss function L, a scaling factor U, a pair $(\mathbf{s}^+, \mathbf{s}^-)$ of start vectors in $[0, 1]^N$ with $\sum_{i=1}^N (s_i^+ + s_i^-) = 1$, and a learning rate η. We simply write $\mathrm{EG}^{\pm}$ for $\mathrm{EG}_L^{\pm}$ where L is the square loss function. As the start vectors for $\mathrm{EG}^{\pm}$, one would typically use $\mathbf{s}^+ = \mathbf{s}^- = (1/(2N), ..., 1/(2N))$. This gives $\mathbf{w}_1^+ - \mathbf{w}_1^- = \mathbf{0}$. A typical learning rate function could be $\eta = 1/(3U^2X^2)$ where X is an estimated upper bound for the maximum L_∞ norm $\max_t \|\mathbf{x}_t\|_\infty$ of the instances. More detailed theoretical results are given in Theorem 5.11.

Again, we introduce one particular variable learning rate version of $\mathrm{EG}^{\pm}$. We use the name $\mathrm{EGV}^{\pm}$ for the algorithm that is as $\mathrm{EG}^{\pm}$ except that (3.10) and (3.11) are replaced by

$$r_{t,i}^{+} = \exp\left(-\frac{2\eta}{\|\mathbf{x}_t\|_\infty^2}(\hat{y}_t - y_t)\, Ux_{t,i}\right) \tag{3.12}$$

$$r_{t,i}^{-} = \exp\left(\frac{2\eta}{\|\mathbf{x}_t\|_\infty^2}(\hat{y}_t - y_t)\, Ux_{t,i}\right) = \frac{1}{r_{t,i}^{+}}, \tag{3.13}$$

assuming $\|\mathbf{x}_t\|_\infty > 0$. If $\|\mathbf{x}_t\|_\infty = 0$, the algorithm makes no update. Again, EGV^{+} turns out to be interesting in the noise-free case $y_t = \mathbf{u} \cdot \mathbf{x}_t$.

As with the EG$_L$ algorithm, we can replace the exponential functions using the approximation $e^{-av} \approx e^{-av_0}(1 - a(v - v_0))$. Here we choose $a = \eta L'_{y_t}(\hat{y}_t)\, U$, $v_0 = \hat{y}_t / U$, and $v = x_{t,i}$ or $v = -x_{t,i}$, which yields the update rule

$$w_{t+1,i}^{+} = w_{t,i}^{+}(1 - \eta L'_{y_t}(\hat{y}_t)(Ux_{t,i} - \hat{y}_t)) \tag{3.14}$$

$$w_{t+1,i}^{-} = w_{t,i}^{-}(1 - \eta L'_{y_t}(\hat{y}_t)(-Ux_{t,i} - \hat{y}_t)). \tag{3.15}$$

We call the resulting algorithm the *approximated* EG$_L^{\pm}$ *algorithm*. As with the approximated EG$_L$ algorithm, to guarantee that the weights remain positive the learning rate η must satisfy

$$\eta < (L'_{y_t}(\hat{y}_t)(Ux_{t,i} - \hat{y}_t))^{-1} \tag{3.16}$$

for all i for which the right-hand side of (3.16) is positive, and

$$\eta < (L'_{y_t}(\hat{y}_t)(-Ux_{t,i} - \hat{y}_t))^{-1} \tag{3.17}$$

for all i for which the right-hand side of (3.17) is positive.

4. DERIVATION OF THE UPDATES

4.1. Basic Method

In this section, we give a common motivation for the algorithms GD and EG introduced in Section 3, as well as some additional algorithms. Consider an algorithm that before a given trial has $\mathbf{s}$ as its weight vector. At the trial, the algorithm receives an instance $\mathbf{x}$, gives a prediction $\hat{y} = \mathbf{s} \cdot \mathbf{x}$, and receives an outcome y. The algorithm then updates its weight vector to $\mathbf{w}$. In choosing the new weight vector $\mathbf{w}$, there are two main considerations. First, the algorithm should learn something from the trial. Thus, if the same instance and outcome were to be observed again, the loss $L(y, \mathbf{w} \cdot \mathbf{x})$ of the algorithm with the new weight vector should be smaller than the loss $L(y, \mathbf{s} \cdot \mathbf{x})$ with the old weight vector. We call the tendency to improve the prediction on the example *correctiveness*. Second, the algorithm should remember at least part of what it learned in the preceding trials. Since all the information that the algorithm has retained from the preceding trials is contained in the weight vector $\mathbf{s}$, the new weight vector should be close to the old weight vector $\mathbf{s}$, as measured by some distance measure $d(\mathbf{w}, \mathbf{s})$. We call the tendency to remain close to the old weight vector *conservativeness*.

The correctiveness and conservativeness requirements are usually at odds with each other, so the algorithm needs to make a compromise. One way for the algorithm to obtain such a compromise is to minimize a function

$$U(\mathbf{w}) = d(\mathbf{w}, \mathbf{s}) + \eta L(y, \mathbf{w} \cdot \mathbf{x}), \tag{4.1}$$

where the coefficient $\eta > 0$ is the importance given to correctiveness relative to conservativeness. If η is close to 0, minimizing $U(\mathbf{w})$ is close to merely minimizing $d(\mathbf{w}, \mathbf{s})$, and hence the algorithm tends to make only small updates. In the limit where η approaches infinity, the problem of minimizing $U(\mathbf{w})$ approaches the problem of minimizing $d(\mathbf{s}, \mathbf{w})$ subject to the constraint $L(y, \mathbf{w} \cdot \mathbf{x}) = 0$. If we expect that the instances or outcomes given to the algorithm are subject to noise or otherwise unreliable, we choose a small value of η, which prohibits the algorithm from making too radical changes based on a single trial.

To minimize $U(\mathbf{w})$, we would need to set its N partial derivatives $\partial U(\mathbf{w})/\partial w_i$ to zero. Since $\partial L(y, \mathbf{w} \cdot \mathbf{x})/\partial w_i = L'_y(\mathbf{w} \cdot \mathbf{x}_t) x_i$, this means finding, for $i = 1, ..., N$, the value w_i that satisfies

$$\frac{\partial d(\mathbf{w}, \mathbf{s})}{\partial w_i} + \eta L'_y(\mathbf{w} \cdot \mathbf{x}) x_i = 0. \tag{4.2}$$

Solving (4.2) for w_i is, in general, very difficult. However, if we replace $L'_y(\mathbf{w} \cdot \mathbf{x})$ by $L'_y(\mathbf{s} \cdot \mathbf{x})$ in (4.2), we get the equation

$$\frac{\partial d(\mathbf{w}, \mathbf{s})}{\partial w_i} + \eta L'_y(\mathbf{s} \cdot \mathbf{x}) x_i = 0, \tag{4.3}$$

which turns out to be easy to solve for all the distance measures d we consider. If the update is small, i.e., if the new weight vector $\mathbf{w}$ is close to the old weight vector $\mathbf{s}$, then replacing $L'_y(\mathbf{w} \cdot \mathbf{x})$ by $L'_y(\mathbf{s} \cdot \mathbf{x})$, which leads from (4.2) to (4.3), gives a reasonable approximation. Thus, we apply in our algorithms update rules that result from solving (4.3) for w_i with various distance measures d.

Helmbold *et al.* (1996b, 1996c) give an alternative motivation for (4.3). Recall that our goal is to minimize $U(\mathbf{w})$, and that solving this minimization problem exactly is difficult because both $d(\mathbf{w}, \mathbf{s})$ and $L(y, \mathbf{w} \cdot \mathbf{x})$ depend on $\mathbf{w}$. To simplify the dependence of $L(y, \mathbf{w} \cdot \mathbf{x})$ on $\mathbf{w}$ we approximate $L(y, \mathbf{w} \cdot \mathbf{x})$ with its first-order Taylor polynomial with respect to $\mathbf{w}$ around $\mathbf{w} = \mathbf{s}$. In other words, instead of minimizing $U(\mathbf{w})$ we minimize its approximation $\hat{U}(\mathbf{w})$ defined by

$$\hat{U}(\mathbf{w}) = d(\mathbf{w}, \mathbf{s}) + \eta(L(y, \mathbf{s} \cdot \mathbf{x}) + L'_y(\mathbf{s} \cdot \mathbf{x})\, \mathbf{x} \cdot (\mathbf{w} - \mathbf{s})).$$

Now the equation $\partial \hat{U}(\mathbf{w})/\partial w_i = 0$ simplifies to (4.3).

Of course, instead of approximating U with $\hat{U}$ and then solving the minimization problem for $\hat{U}$ exactly, we could apply some numerical method directly to find an approximate minimum for U. It is not clear whether this would results in a better

prediction performance, but it certainly would make the computations more complicated.

For another view to the meaning of minimizing U, assume that there is a unique weight vector $\mathbf{w}'$ such that that $U(\mathbf{w})$ is minimized for $\mathbf{w} = \mathbf{w}'$, and write $p = L(y, \mathbf{w}' \cdot \mathbf{x})$. Thus, p is a real number that depends on η, y, $\mathbf{s}$, and $\mathbf{x}$. When η approaches ∞, the relative importance of $L(y, \mathbf{w} \cdot \mathbf{x})$ in $U(\mathbf{w})$ increases and hence p approaches 0. It is the easy to see that $\mathbf{w}'$ is also the unique solution to the constrained problem of minimizing $d(\mathbf{w}, \mathbf{s})$ subject to $L(y, \mathbf{w} \cdot \mathbf{x}) \leqslant p$. Hence, the optimal weight vector $\mathbf{w}'$ can be seen as obtained from moving from $\mathbf{s}$ into a region of low loss, defined by the condition $L(y, \mathbf{w} \cdot \mathbf{x}) \leqslant p$, along the shortest possible route. For large values of η, the value p is close to 0 and the new weight vector is required to be almost correct for the received instance and outcome. For small values of η, the value p is slightly less than $L(y, \mathbf{s} \cdot \mathbf{x})$, so the weight vector is made to make only a small corrective movement. This approach to updating a weight vector is similar to the methods introduced by Amari (1994, 1995) for more general neural network learning problems.

In the next subsections we derive the updates of this paper using the method described above with the distance measures d_{sq}, d_{re}, d_{reu}, and d_{χ^2}. Sometimes, in particular with the distance measure d_{re}, we wish to guarantee the additional property $\sum_{i=1}^{N} w_i = 1$. We do this by the usual method of introducing a Lagrangian multiplier γ. Hence, instead of minimizing $\hat{U}$ we minimize $\tilde{U}$ defined by

$$\tilde{U}(\mathbf{w}, \gamma) = d(\mathbf{w}, \mathbf{s}) + \eta(L(y, \mathbf{s} \cdot \mathbf{x}) + L'_y(\mathbf{s} \cdot \mathbf{x})\, \mathbf{x} \cdot (\mathbf{w} - \mathbf{s})) + \gamma \left(\left(\sum_{i=1}^{N} w_i \right) - 1 \right).$$

Setting the $N+1$ partial derivatives of $\tilde{U}$ to zero gives us the equations

$$\frac{\partial d(\mathbf{w}, \mathbf{s})}{\partial w_i} + \eta L'_y(\mathbf{s} \cdot \mathbf{x})\, x_i + \gamma = 0 \tag{4.4}$$

for $i = 1, ..., N$ and the additional equation

$$\sum_{i=1}^{N} w_i = 1. \tag{4.5}$$

Thus, when the additional constraint $\sum_{i=1}^{N} w_i = 1$ is needed, we solve for $i = 1, ..., N$ Eqs. (4.4) instead of (4.3) and then apply (4.5) to obtain the value for γ.

4.2. Gradient Descent Algorithms

The gradient descent algorithm GD_L is obtained by using the squared Euclidean distance d_{sq} as the distance measure. For this case, Eqs. (4.3) result in the update

$$w_i = s_i - \eta L'_y(\mathbf{s} \cdot \mathbf{x})\, x_i, \tag{4.6}$$

which is the update rule for the GD_L algorithm.

Note that if the loss function is the square loss, the gradient descent update becomes

$$\mathbf{w} = \mathbf{s} - 2\eta(\mathbf{s} \cdot \mathbf{x} - y)\,\mathbf{x}.$$

For the square loss and the squared Euclidean distance we can also minimize $U(\mathbf{w})$ directly without making any approximations. In this case, (4.2) becomes

$$\mathbf{w} = \mathbf{s} - 2\eta(\mathbf{w} \cdot \mathbf{x} - y)\,\mathbf{x}. \tag{4.7}$$

From (4.7), one can solve for $\mathbf{w} \cdot \mathbf{x}$ by first taking the dot product of both sides with $\mathbf{x}$. If $\mathbf{w} \cdot \mathbf{x}$ is substituted back into (4.7), one gets

$$\mathbf{w} = \mathbf{s} - \frac{2\eta}{1 + 2\eta\,\|\mathbf{x}\|_2^2}(\mathbf{s} \cdot \mathbf{x} - y)\,\mathbf{x}. \tag{4.8}$$

Thus, minimizing $U(\mathbf{w})$ gives the same update as minimizing $\hat{U}(\mathbf{w})$, except that the learning rate parameter is changed in a manner that is independent of $\mathbf{w}$.

If we use the squared Euclidean distance together with the constraint that the weights w_i must sum to 1, we obtain an algorithm GP_L that is known as the gradient projection algorithm (Luenberger, 1984). For this case, Eq. (4.4) implies

$$w_i = s_i - \eta L'_y(\mathbf{s} \cdot \mathbf{x})\,x_i - \gamma. \tag{4.9}$$

By substituting (4.9) into (4.5) and using the assumption $\sum_{i=1}^{N} s_i = 1$, we obtain

$$\gamma = -\eta L'_y(\mathbf{s} \cdot \mathbf{x})\,\mathrm{avg}(\mathbf{x}), \qquad \text{where} \quad \mathrm{avg}(\mathbf{x}) = \sum_{i=1}^{N} x_i/N.$$

Substituting this back into (4.9) gives us the update rule

$$w_i = s_i - \eta L'_y(\mathbf{s} \cdot \mathbf{x})(x_i - \mathrm{avg}(\mathbf{x})). \tag{4.10}$$

for the GP_L algorithm. We introduce GP_L in this paper for the purpose of comparison to our new algorithm EG_L which also maintains the constraint that the weight sum to 1. Actually the new algorithm EG_L also keeps the weights positive. One can keep the weights of GP_L positive as well by setting a suitable upper bound for the learning rate as we did for the approximated EG_L and $\mathrm{EG}_L^{\pm}$ algorithms in Section 3.

4.3. Exponential Gradient Algorithms

We now use the relative entropy distance measures d_{reu} and d_{re}. Both measures assume that the weight vectors have non-negative components, and $d_{\mathrm{re}}(\mathbf{w}, \mathbf{s})$

requires that the components of the weight vectors sum to one. Substituting $d = d_{\text{reu}}$ in (4.3) gives us

$$\ln \frac{w_i}{s_i} + \eta L'_y(\mathbf{s} \cdot \mathbf{x}) x_i = 0.$$

By solving for w_i, we obtain the update rule

$$w_i = s_i \exp(-\eta L'_y(\mathbf{s} \cdot \mathbf{x}) x_i). \tag{4.11}$$

We call the algorithm with this update rule the *exponentiated gradient algorithm with unnormalized weights* and denote it by EGU_L. The update rule of the EGU_L algorithm is like the update rule of EG_L, except for the normalization in the update rule for EG_L. Assuming that the components of $\mathbf{s}$ are nonnegative, the components of the updated weight vector $\mathbf{w}$ are nonnegative as well. Thus, the nonnegativity constraints are always preserved by this update.

Consider now the distance measure d_{re}, which requires the constraint $\sum_{i=1}^N w_i = \sum_{i=1}^N s_i = 1$. For this case, Eq. (4.4) becomes

$$\ln \frac{w_i}{s_i} + 1 + \eta L'_y(\mathbf{s} \cdot \mathbf{x})\, x_i + \gamma = 0,$$

from which we obtain

$$w_i = s_i \exp(-\eta L'_y(\mathbf{s} \cdot \mathbf{x})\, x_i - 1 - \gamma).$$

Hence, $w_i = s_i r_i \exp(-\gamma - 1)$ where

$$r_i = \exp(-\eta L'_y(\mathbf{s} \cdot \mathbf{x}) x_i).$$

Applying (4.5), we obtain $\exp(-\gamma - 1) = (\sum_{j=1}^N s_j r_j)^{-1}$. Hence, the update rule is

$$w_i = \frac{s_i r_i}{\sum_{j=1}^N s_j r_j}.$$

Note that the update rule keeps the weights w_i positive if the weights s_i are positive.

4.4. Approximated Exponential Gradient

In Section 3, we introduced a simple approximation (3.5) for the update rule of the EG algorithm. We next show that we can motivate the update rule (3.5) of the approximated EG algorithm starting directly from a distance measure, as we already did for the GD and EG algorithms. Hence, the approximated EG algorithm seems to have some interest in its own right. We use the distance measure d_{χ^2} to motivate the approximated EG algorithm. As noted in Section 2, $d_{\chi^2}(\mathbf{w}, \mathbf{s})$ approximates $d_{\text{reu}}(\mathbf{w}, \mathbf{s})$ if $\mathbf{w}$ and $\mathbf{s}$ are close to each other. We first look at the restricted case with $\sum_{i=1}^N w_i = \sum_{i=1}^N s_i = 1$. In this case, we obviously can also approximate $d_{\text{re}}(\mathbf{w}, \mathbf{s})$ by $d_{\chi^2}(\mathbf{w}, \mathbf{s})$.

For the distance function d_{χ^2}, Eq. (4.3) becomes

$$\frac{w_i}{s_i} - 1 + \eta L'_y(\mathbf{s}\cdot\mathbf{x})\, x_i + \gamma = 0.$$

Solving for w_i, and for γ that gives $\sum_{i=1}^N w_i = 1$, we get from this

$$w_i = s_i(1 - \eta L'_y(\mathbf{s}\cdot\mathbf{x})(x_i - \mathbf{s}\cdot\mathbf{x})),$$

which is the update rule (3.5) of the approximated EG algorithm. The non-negativity of the weights w_i is guaranteed if we bound the learning rate by requiring

$$\eta < (L'_y(\mathbf{s}\cdot\mathbf{x})(x_i - \mathbf{s}\cdot\mathbf{x}))^{-1}$$

to hold for all i such that $L'_y(\mathbf{s}\cdot\mathbf{x})(x_i - \mathbf{s}\cdot\mathbf{x})$ is positive. Since the update is multiplicative, a component s_i that is zero will stay at zero. Therefore, we should not allow w_i to be set to 0.

We can use the measure d_{χ^2} also when the sums $\sum_{i=1}^N s_i$ and $\sum_{i=1}^N w_i$ are not necessarily 1. Omitting the normalization constraint from the previous derivation gives us the update rule

$$w_i = s_i(1 - \eta L'_y(\mathbf{s}\cdot\mathbf{x})\, x_i),$$

which is an approximation to the update rule of the EGU algorithm. In a simple unsupervised learning problem for learning mixture coefficients it has been noticed that the distance measure d_{χ^2} can also be used to motivate a generalization of the Expectation Maximization (EM) optimization method (Helmbold *et al.*, 1996b, 1996c).

In summary, we have seen that there are two different ways of arriving at the same approximated EG algorithm. First, one can approximate the exponential function in the update rule of EG. Second, one can use the d_{χ^2} distance measure, which is an approximation for the relative entropy d_{re}, to derive an algorithm in the same manner the relative entropy was used to derive the EG algorithm.

4.5. The Approximation Step in the Derivations

In the following sections, we prove that learning algorithms based on the preceding semi-rigorous motivations actually perform well for the square loss. We prove worst-case square loss bounds for all algorithms introduced in this section, except for the approximated versions of the exponentiated gradient updates. However, experimental results suggest that the approximated version of the exponentiated gradient updates behave very closely to the actual exponentiated gradient updates. (See Subsection 9.5.)

We have been unable to prove worst-case loss bounds expressed as a function of the loss of the best linear weight vector for other loss functions than the square loss. In fact, we have reason to believe that the step of evaluating the derivative

$\partial L(y, z)/\partial z = L'_y(z)$ at $z = \mathbf{s} \cdot \mathbf{x}$ instead of $z = \mathbf{w} \cdot \mathbf{x}$ may lead to bad results particularly if the loss function is unbounded. For example, let L be the relative entropy loss. We then have

$$L'_y(z) = -\frac{y}{z} + \frac{1-y}{1-z}.$$

For $0 < y < 1$, the value $L'_y(z)$ changes dramatically as z approaches 0 or 1. Hence, if the prediction $\mathbf{s} \cdot \mathbf{x}$ was very close to 0 or 1, then the value $L'_y(\mathbf{s} \cdot \mathbf{x})$ may not be close to the value $L'_y(\mathbf{w} \cdot \mathbf{x})$, even if $\mathbf{w}$ and $\mathbf{s}$ were relatively close to each other. Therefore, the approximation made in the derivation of the algorithms may be very inaccurate.

To see the possible consequences of using the algorithms based on the questionable approximation, consider the algorithm EG_L for the relative entropy loss in a simple two-dimensional case with $\mathbf{x} = (0, 1)$ and $\mathbf{s} = (1-p, p)$ for some p. Then $\mathbf{s} \cdot \mathbf{x} = p$. As p approaches 0 and y remains fixed to, say 1/2, the value $\exp(-\eta L'_y(p) x_i)$ approaches ∞ for $i = 2$, but remains 1 for $i = 1$. Hence, if we write $\mathbf{w} = (1-q, q)$ for the weight vector $\mathbf{w}$ after the update, we see that q approaches 1 as p approaches 0. Similarly, if p approaches 1, then q approaches 0. Thus, if at some stage of the trial sequence the weight vector get too close to $(0, 1)$ or $(1, 0)$, the consequent updates cause the weight vector to oscillate wildly, even if the outcomes remain constant. This eventually leads to arbitrarily large losses for the algorithm, although a fixed linear predictor might have a very small loss.

We believe that for linear prediction with the relative entropy loss and other unbounded loss functions, better prediction results can be obtained by solving the minimization problem for $U(\mathbf{w})$ numerically. However, we have no results on this. Obviously, the numerical solving would increase the computational cost of the algorithm.

5. WORST CASE UPPER BOUNDS FOR THE TOTAL LOSS

5.1. Basic Method

We next introduce the basic method used in our proofs for worst case upper bounds for the losses of on-line prediction algorithms. The method is an abstraction of the proof method employed by Littlestone (1989b, 1990) and others (Littlestone *et al.*, 1995; Cesa-Bianchi *et al.*, 1996). In the subsequent subsections, we show how this basic idea can be applied to the specific algorithms introduced in Section 3. We have succeeded in this application only for the square loss function, but we hope it could also be applicable to other loss functions. In a simpler situation, where the learner is not trying to learn a linear function but merely to pick out the best single component of the instances for predicting the outcomes, it has been possible to use a similar approach to prove bounds for a very general class of loss functions (Vovk, 1990). As noted in Section 4, the algorithms can be motivated by applying a distance measure to the weight vectors maintained by the algorithms. These distance

measures will also be useful in proving worst-case loss bounds. They have a role similar to that of potential functions in amortized algorithm analysis (Cormen *et al.*, 1990).

Let $\mathbf{w}_1, ..., \mathbf{w}_\ell$ be the sequence of weight vectors produced by an algorithm A on an N-dimensional trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, and let $\hat{y}_t$ be the tth prediction of A. Then $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$ holds for the algorithms we consider, except in some special cases where we constrain the range of $\hat{y}_t$. For convenience, we assume that the algorithm makes an update even after the last trial, although the resulting weight vector $\mathbf{w}_{\ell+1}$ is never used for predicting. Let d be a distance measure and L a loss function. Given a weight vector $\mathbf{u} \in \mathbf{R}^N$, we say that $d(\mathbf{u}, \mathbf{w}_t) - d(\mathbf{u}, \mathbf{w}_{t+1})$ is the amount of *progress* made by the algorithm at trial t towards $\mathbf{u}$. Negative progress towards $\mathbf{u}$ means actually moving away from $\mathbf{u}$. Naturally, $\mathbf{u}$ must be in the domain of the distance measure d. For instance, $\mathbf{u}$ must satisfy $\mathbf{u} \in [0, 1]^N$ and $\sum_i u_i = 1$ if $d(\mathbf{u}, \mathbf{w}) = d_{\text{re}}(\mathbf{u}, \mathbf{w})$.

In a single trial, we would expect the algorithm to make positive progress towards those vectors $\mathbf{u}$ that made an accurate prediction, and negative progress towards those vectors $\mathbf{u}$ that predicted inaccurately. This is reflected in the motivation of the algorithms, where our goal is to move the weight vector towards those weight vectors that made an exactly correct prediction. Over a whole sequence of trials, we would expect the net effect to be that the algorithm makes positive progress towards those vectors $\mathbf{u}$ for which the total loss $\text{Loss}_L(\mathbf{u}, S)$ is relatively small, and negative progress towards those vectors $\mathbf{u}$ for which the total loss $\text{Loss}_L(\mathbf{u}, S)$ is relatively large.

Consider first the special case that for some particular vector $\mathbf{u}$ we have $y_t = \mathbf{u} \cdot \mathbf{x}_t$ for all t, and hence $\text{Loss}_L(\mathbf{u}, S) = 0$. We then say $\mathbf{u}$ is the target vector, and the algorithm should try to find it. We could require that the progress towards the target $\mathbf{u}$ at trial t should be proportional to the loss of the algorithm at trial t. This is a specific way of saying that the algorithm should learn from its mistakes. Generally, we wish to allow the situation that $\text{Loss}_L(\mathbf{u}, S) > 0$ holds for all $\mathbf{u}$. Then there is no obvious target vector for the algorithm to try to approach. However, we can require that at each trial t, for all weight vectors $\mathbf{u}$, the progress of the algorithm towards $\mathbf{u}$ is at least $aL(y_t, \hat{y}_t) - bL(y_t, \mathbf{u} \cdot \mathbf{x}_t)$ for some positive coefficients a and b. Thus, the algorithm should make large progress towards those weight vectors that predicted much more accurately than the algorithm did.

Hence, we try to establish bounds of the form

$$aL(y_t, \hat{y}_t) - bL(y_t, \mathbf{u} \cdot \mathbf{x}_t) \leqslant d(\mathbf{u}, \mathbf{w}_t) - d(\mathbf{u}, \mathbf{w}_{t+1}), \tag{5.1}$$

which we require to hold for all weight vectors $\mathbf{u}$ that we consider as possible targets. Proving bounds of this form is the main technical problem in this paper. To get the tightest bound, we would wish a to be as large and b to be as small as possible in (5.1). Expectedly, there is a trade − off, and for any given positive value b there is some largest value of a for which we can prove (5.1). It turns out to be convenient to introduce a new parameter, c, and two functions, f and g, such that for all values $c > 0$ if we take $b = g(c)$, then $a = f(c)$ is the largest value of a for which we can prove (5.1). To obtain (5.1) for $a = f(c)$ and $b = g(c)$, the learning rate η

must be set in a particular manner that depends on the value c. The bound for the total loss of the algorithm follows by adding the bounds (5.1) with $a = f(c)$ and $b = g(c)$ for $t = 1, ..., \ell$, which yields

$$\begin{aligned} f(c)\,\mathrm{Loss}_L(A, S) - g(c)\,\mathrm{Loss}_L(\mathbf{u}, S) &\leqslant \sum_{t=1}^{\ell} (d(\mathbf{u}, \mathbf{w}_t) - d(\mathbf{u}, \mathbf{w}_{t+1})) \\ &= d(\mathbf{u}, \mathbf{w}_1) - d(\mathbf{u}, \mathbf{w}_{\ell+1}) \\ &\leqslant d(\mathbf{u}, \mathbf{s}), \end{aligned}$$

since $\mathbf{w}_1 = \mathbf{s}$ and $d(\mathbf{u}, \mathbf{w}_{\ell+1}) \geqslant 0$. Hence, we have

$$\mathrm{Loss}_L(A, S) \leqslant \frac{g(c)}{f(c)}\,\mathrm{Loss}_L(\mathbf{u}, S) + \frac{d(\mathbf{u}, \mathbf{s})}{f(c)}. \tag{5.2}$$

Note that (5.2) holds for all possible weight vectors $\mathbf{u}$, although we naturally get the best bound if $\mathbf{u}$ has a small loss and is close to the start vector. The final step in the proof is to choose the value c that minimizes the right-hand side of (5.2) and, hence, gives the tightest bound. For the functions f, we obtain in our proofs, it is always the case that as c approaches 0, the ratio $g(c)/f(c)$ approaches 1 and $1/f(c)$ approaches infinity; as c goes to infinity, the ratio $g(c)/f(c)$ also goes to infinity while $1/f(c)$ approaches some positive constant. Therefore, the larger the loss $\mathrm{Loss}_L(\mathbf{u}, S)$ is compared to the distance $d(\mathbf{u}, \mathbf{s})$, the smaller value of c we wish to use. For the particular functions f and g we have in this paper, choosing c in the optimal way gives bounds of the form

$$\mathrm{Loss}_L(A, S) \leqslant \inf_{\mathbf{u}} (\mathrm{Loss}_L(\mathbf{u}, S) + c_1 \sqrt{\mathrm{Loss}_L(\mathbf{u}, S)\, d(\mathbf{u}, \mathbf{s})} + c_2 d(\mathbf{u}, \mathbf{s})). \tag{5.3}$$

The coefficients c_1 and c_2 can depend on the norms of the instances $\mathbf{x}_t$.

Since the learning rate η for which (5.2) is achieved depends on c, and the bound (5.3) is obtained only by choosing c based on some estimates on $\mathrm{Loss}_L(\mathbf{u}, S)$ and $d(\mathbf{u}, \mathbf{s})$ for a suitable target $\mathbf{u}$, we do not directly obtain the bound (5.3) in the absence of such estimates. However, if such estimates are not known before the trial sequence begins, it is in some situations possible to use an iterative method, commonly known as the doubling technique (Cesa-Bianchi *et al.*, 1994; Cesa-Bianchi *et al.*, 1996), for obtaining increasingly accurate estimates as the trial sequence proceeds and modifying the learning rate accordingly. This leads to bounds of the form (5.3), but with slightly worse constant coefficients. Another possibility is to settle for weaker bounds of the form

$$\mathrm{Loss}_L(A, S) \leqslant c_3 \inf_{\mathbf{u}} (\mathrm{Loss}_L(\mathbf{u}, S) + d(\mathbf{u}, \mathbf{s})) \tag{5.4}$$

that has a coefficient $c_3 > 1$ for the leading term $\mathrm{Loss}_L(\mathbf{u}, S)$. The weaker bounds can be achieved without any additional knowledge.

5.2. *Worst-Case Loss Bounds for* GD

In this subsection we give a streamlined version of the worst-case analysis of the GD algorithm. The analysis was originally presented by Cesa-Bianchi *et al.* (1996). We start by bounding the loss of the algorithm at a single trial in terms of the loss of a comparison vector $\mathbf{u}$ at that trial and the progress of the algorithm towards $\mathbf{u}$.

LEMMA 5.1. *Let $\mathbf{w}_t$ be the weight vector of* GD$(\mathbf{s}, \eta)$ *before trial t in a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, and let $\mathbf{u} \in \mathbf{R}^N$ be arbitrary. Let t be arbitrary, and let X be such that $\|\mathbf{x}_t\|_2 \leqslant X$. For all values a and b such that $0 < a \leqslant b/(1 + 2X^2 b)$, and a learning rate η that satisfies $\eta = b/(1 + 2X^2 b)$, we have*

$$a(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - b(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \leqslant \tfrac{1}{2}(\|\mathbf{u} - \mathbf{w}_t\|_2^2 - \|\mathbf{u} - \mathbf{w}_{t+1}\|_2^2). \tag{5.5}$$

For any values a and b such that $0 < b/(1 + 2\|\mathbf{x}_t\|_2^2 b) < a$, for any learning rate η there are trial sequences S and vectors $\mathbf{u} \in \mathbf{R}^N$ such that (5.5) *does not hold.*

Proof. Write $p_t = y_t - \mathbf{w}_t \cdot \mathbf{x}_t$ and $q_t = y_t - \mathbf{u} \cdot \mathbf{x}_t$. For $\mathbf{w}_{t+1} = \mathbf{w}_t + 2\eta p_t \mathbf{x}_t$ we then have

$$\begin{aligned}\tfrac{1}{2}(\|\mathbf{u} - \mathbf{w}_t\|_2^2 - \|\mathbf{u} - \mathbf{w}_{t+1}\|_2^2) &= -2\eta p_t(\mathbf{w}_t \cdot \mathbf{x}_t - \mathbf{u} \cdot \mathbf{x}_t) - 2\eta^2 \|\mathbf{x}_t\|_2^2 p_t^2 \\ &\geqslant -2\eta p_t(q_t - p_t) - 2\eta^2 X^2 p_t^2.\end{aligned}$$

Here equality holds if $X = \|\mathbf{x}_t\|_2$. Hence, to prove (5.5), it is sufficient to show $F(p_t, q_t, \eta) \leqslant 0$, where

$$F(p_t, q_t, \eta) = 2\eta p_t(q_t - p_t) + 2\eta^2 X^2 p_t^2 + a p_t^2 - b q_t^2.$$

Further, if $X = \|\mathbf{x}_t\|_2$, then $F(p_t, q_t, \eta) \leqslant 0$ is also a necessary condition for (5.5).

We now omit the subscript t from the formulas. As $F(p, q, \eta)$ is a second degree polynomial in q and b is positive, we easily see that for a fixed p and η, the value $F(p, q, \eta)$ is maximized when $q = \eta p/b$. Hence, it is sufficient to show for all p that $G(p, \eta) \leqslant 0$, where

$$G(p, \eta) = F(p, \eta p/b, \eta) = p^2((2X^2 + 1/b)\eta^2 - 2\eta + a).$$

Again, we easily see that for fixed p the value $G(p, \eta)$ is minimized if we choose $\eta = b/(1 + 2X^2 b)$. For this optimal choice we get

$$G(p, b/(1 + 2X^2 b)) = \frac{p^2}{1 + 2X^2 b}(2X^2 ab + a - b).$$

Thus, if $0 < a \leqslant b/(1 + 2X^2 b) = \eta$, we have $G(p, \eta) \leqslant 0$, and (5.5) holds. If $b/(1 + 2X^2 b) < a$, then for all values of η we have $F(p, q, \eta) > 0$ for some p and q. Since we can easily construct a trial sequence S and a vector $\mathbf{u}$ for which

$y_t - \mathbf{w}_t \cdot \mathbf{x}_t = p$ and $y_t - \mathbf{u} \cdot \mathbf{x}_t = q$ hold, this shows that (5.5) does not hold if $X = \|\mathbf{x}_t\|_2$ and $b/(1 + 2X^2 b) < a$. ∎

Note that the expression $b/(1 + 2X^2 b)$ used to give the learning rate in Lemma 5.1 is similar in form to the expression $\eta/(1 + 2\eta \|\mathbf{x}\|_2^2)$ that appears in the place of the learning rate in (4.8). Therefore, by the remarks preceding (4.8), we see that the new weight vector $\mathbf{w}_{t+1}$ chosen by the GD algorithm with the learning rate $b/(1 + 2b \|\mathbf{x}_t\|_2^2)$ suggested by Lemma 5.1 is the vector $\mathbf{w}$ for which the value

$$\tfrac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|_2^2 + b(y_t - \mathbf{w} \cdot \mathbf{x}_t)^2$$

is minimized.

The next simple lemma shows how repeated application of Lemma 5.1 to all the trials of a sequence gives a total loss bound. We introduce a new parameter c for the purpose of choosing the values of a and b in the applications of Lemma 5.1.

LEMMA 5.2. *Let $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ be an arbitrary trial sequence. Let $\eta = c/(X^2(1 + 2c))$ where $c > 0$ is an arbitrary parameter and $X > 0$ an upper bound such that $\|\mathbf{x}_t\|_2 \leqslant X$ holds for all t. For all start vectors $\mathbf{s} \in \mathbf{R}^N$, and comparison vectors $\mathbf{u} \in \mathbf{R}^N$ we then have*

$$\mathrm{Loss}(\mathrm{GD}(\mathbf{s}, \eta), S) \leqslant (1 + 2c)\,\mathrm{Loss}(\mathbf{u}, S) + \left(1 + \frac{1}{2c}\right)\|\mathbf{u} - \mathbf{s}\|_2^2 X^2. \tag{5.6}$$

Proof. Let $b = c/X^2$ and $a = b/(1 + 2X^2 b) = c/(X^2(1 + 2c))$. Let $\mathbf{w}_t$ be the tth weight vector of $\mathrm{GD}(\mathbf{s}, \eta)$ on the trial sequence S. Then (5.5) holds by Lemma 5.1, and therefore

$$\frac{c}{1 + 2c}(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - c(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \leqslant \frac{X^2}{2}(\|\mathbf{u} - \mathbf{w}_t\|_2^2 - \|\mathbf{u} - \mathbf{w}_{t+1}\|_2^2). \tag{5.7}$$

By adding the bounds (5.7) for $t = 1, ..., \ell$ we get

$$\begin{aligned}\frac{c}{1 + 2c}\,\mathrm{Loss}(\mathrm{GD}(\mathbf{s}, \eta), S) - c\,\mathrm{Loss}(\mathbf{u}, S) &\leqslant \frac{X^2}{2}(\|\mathbf{u} - \mathbf{w}_1\|_2^2 - \|\mathbf{u} - \mathbf{w}_{\ell+1}\|_2) \\ &\leqslant \frac{\|\mathbf{u} - \mathbf{w}_1\|_2^2 X^2}{2},\end{aligned}$$

which is equivalent with (5.6). ∎

We now show how the final loss bounds are obtained by choosing the parameter c in Lemma 5.2 appropriately. If we have no knowledge of the relative magnitudes of the loss $\mathrm{Loss}(\mathbf{u}, S)$ of the comparison vector $\mathbf{u}$ and the product $\|\mathbf{u} - \mathbf{s}\|_2^2 X^2$, we can choose c in such a way that the coefficients of these quantities become the same. If we have some estimates of these quantities, we obtain a tighter bound by choosing c in such a way that the larger quantity gets a smaller coefficient.

THEOREM 5.3. *For a trial sequence* $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, *let* $X > 0$ *be an upper bound such that* $\|\mathbf{x}_t\|_2 \leqslant X$ *holds for all* t. *With the learning rate* $\eta = 1/(4X^2)$ *and an arbitrary start vector* $\mathbf{s} \in \mathbf{R}^N$, *we have for any vector* $\mathbf{u}$ *the bound*

$$\text{Loss}(\text{GD}(\mathbf{s}, \eta), S) \leqslant 2(\text{Loss}(\mathbf{u}, S) + \|\mathbf{u} - \mathbf{s}\|_2^2 X^2). \tag{5.8}$$

Further, let K *and* U *be arbitrary constants, and let the learning rate* η *satisfy*

$$\eta = \frac{U}{2X\sqrt{K} + 2UX^2}. \tag{5.9}$$

Then for all $\mathbf{u} \in \mathbf{R}^N$ *such that* $\text{Loss}(\mathbf{u}, S) \leqslant K$ *and* $\|\mathbf{u} - \mathbf{s}\|_2 \leqslant U$ *hold, we have*

$$\text{Loss}(\text{GD}(\mathbf{s}, \eta), S) \leqslant \text{Loss}(\mathbf{u}, S) + 2\sqrt{K}\, UX + \|\mathbf{u} - \mathbf{s}\|_2^2 X^2. \tag{5.10}$$

Note that the second bound becomes vacuous if there is no $\mathbf{u} \in \mathbf{R}^N$ such that $\text{Loss}(\mathbf{u}, S) \leqslant K$ and $\|\mathbf{u} - \mathbf{s}\|_2 \leqslant U$. The typical start vector for GD is $\mathbf{s} = \mathbf{0}$. For that start vector, U is an upper bound on the L_2 norm of $\mathbf{u}$.

Proof. We first apply Lemma 5.2 with $c = 1/2$. For this value we have $1 + 2c = 1 + 1/(2c) = 2$, so the bound (5.6) gives (5.8). The bound (5.6) with $c = 1/2$ holds for the learning rate $\eta = (1/2)/(X^2(1+1)) = 1/(4X^2)$ as required.

To obtain (5.10), we first notice that for K and U that satisfy the assumptions of the theorem, the bound (5.6) implies

$$\text{Loss}(\text{GD}(\mathbf{s}, \eta), S) \leqslant \text{Loss}(\mathbf{u}, S) + \|\mathbf{u} - \mathbf{s}\|_2^2 X^2 + F(c), \tag{5.11}$$

where $F(c) = 2cK + U^2X^2/(2c)$. Assume first that $K > 0$. As $F'(c) = 2K - U^2X^2/(2c^2)$, we then have $F'(c) = 0$ for $c = UX/(2\sqrt{K})$. Since $F''(c) > 0$ for all c, the value $F(c)$ is minimized for $c = UX/(2\sqrt{K})$. Substituting this value of c into (5.11) yields (5.10).

In the special case $K = 0$, we also have $\text{Loss}(\mathbf{u}, S) = 0$, and hence the right-hand side of (5.11) has the limit $\|\mathbf{u} - \mathbf{s}\|_2^2 X^2$ as c approaches infinity. Let us denote by η_c the learning rate $\eta_c = c/(X^2(1+2c))$. Lemma 5.2 now implies $\lim_{c \to \infty} \text{Loss}(\text{GD}(\mathbf{s}, \eta_c), S) \leqslant \|\mathbf{u} - \mathbf{s}\|_2^2 X^2$. Let now $\eta_\infty = \lim_{c \to \infty} \eta_c = 1/(2X^2)$. Since the loss $\text{Loss}(\text{GD}(\mathbf{s}, \eta), S)$ is a continuous function of the learning rate η, we obtain

$$\text{Loss}(\text{GD}(\mathbf{s}, \eta_\infty), S) = \lim_{c \to \infty} \text{Loss}(\text{GD}(\mathbf{s}, \eta_c), S) \leqslant \|\mathbf{u} - \mathbf{s}\|_2^2 X^2,$$

which is the results we claim for $K = 0$. ∎

We can perform a simple dimension check to see that the learning rates given in Theorem 5.3 are, to an extent, meaningful. Assume, for instance, that the input variables $x_{t,i}$ represent times measured in seconds, and the outcomes y_t represent lengths measured in meters. Then also the unit of the predictions $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$ should be 1 meter, so the unit of the weights $w_{t,i}$ should be 1 meter per second. More generally, let the dimension of the input variables be $[x]$ and the dimension of the

outcomes $[y]$. Then the dimension of the weights in $[x]^{-1}[y]$. By considering the update rule (3.1) we see that the dimension of the learning rate η should be $[x]^{-2}$. Since the dimensions of X, K, and U are $[x]$, $[y]^2$, and $[x]^{-1}[y]$, respectively, we see that this is indeed the case. It is also true that all the terms on the right-hand side of the bounds (5.8) and (5.10) have the same dimension as the loss of the algorithm, namely $[y]^2$.

Note that this analysis assumes that all the input variables have the same dimension. If this is not the case, and we change the unit used to measure certain input values while keeping other units unchanged, then the behavior of the algorithm is changed.

Recall that by GDV we mean the algorithm that works as GD except that the learning rate η has been replaced by $\eta/\|\mathbf{x}_t\|_2$, in other words, the update rule (3.1) has been replaced by (3.2). We now see that the GDV algorithm is particularly well suited for prediction if the losses at each trial are suitably scaled. Thus, consider measuring the loss at trial t by $(y_t-\hat{y}_t)^2/\|\mathbf{x}_t\|_2^2$, assuming $\|\mathbf{x}_t\|_2>0$. We ignore trials with $\|\mathbf{x}_t\|_2=0$; this is justified, since at such a trial the GDV algorithm by our definition makes no update, and all linear predictors make the same prediction $\hat{y}_t=0$. Let Loss$'$ denote the loss of an algorithm or a comparison vector on a trial sequence measured by this scaled square loss. Given a trial sequence $S=((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, we consider the modified trial sequence $S'=((\mathbf{x}'_1, y'_1), ..., (\mathbf{x}'_\ell, y'_\ell))$, where $\mathbf{x}'_t=\mathbf{x}_t/\|\mathbf{x}_t\|_2$ and $y'_t=y_t/\|\mathbf{x}_t\|_2$. Since

$$\frac{(\hat{y}_t-y_t)^2}{\|\mathbf{x}_t\|_2^2}=\frac{(\mathbf{x}_t\cdot\mathbf{w}_t-y_t)^2}{\|\mathbf{x}_t\|_2^2}=\left(\frac{\mathbf{x}_t}{\|\mathbf{x}_t\|_2}\cdot\mathbf{w}_t-\frac{y_t}{\|\mathbf{x}_t\|_2}\right)^2,$$

we have $\text{Loss}'(\mathbf{u}, S)=\text{Loss}(\mathbf{u}, S')$ for all $\mathbf{u}$. Note that (3.2) is equivalent with

$$\mathbf{w}_{t+1}=\mathbf{w}_t-2\eta(\mathbf{w}_t\cdot\mathbf{x}'_t-y'_t)\,\mathbf{x}'_t,$$

which implies that the weight vectors of GDV$(\mathbf{s}, \eta)$ on the trial sequence S are the same as the weight vectors of GD$(\mathbf{s}, \eta)$ on the trial sequence S'. Hence, $\text{Loss}'(\text{GDV}(\mathbf{s}, \eta), S)=\text{Loss}(\text{GD}(\mathbf{s}, \eta), S')$. Therefore, Theorem 5.3 applied to the trial sequence S', in which $\|\mathbf{x}'_t\|_2=1$ for all t, gives the following corollary.

COROLLARY. *Let $S=((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ be an arbitrary trial sequence. For arbitrary start vector $\mathbf{s}\in\mathbf{R}^N$ and comparison vector $\mathbf{u}$, we have*

$$\text{Loss}'(\text{GDV}(\mathbf{s}, 1/4), S)\leqslant 2(\text{Loss}'(\mathbf{u}, S)+\|\mathbf{u}-\mathbf{s}\|_2^2).$$

Further, let K and U be arbitrary constants, and let the learning rate be

$$\eta=\frac{U}{2\sqrt{K}+2U}.$$

Then for all $\mathbf{u}\in\mathbf{R}^N$ such that $\text{Loss}'(\mathbf{u}, S)\leqslant K$ and $\|\mathbf{u}-\mathbf{s}\|_2\leqslant U$ hold, we have

$$\text{Loss}'(\text{GDV}(\mathbf{s}, \eta), S)\leqslant\text{Loss}'(\mathbf{u}, S)+2\sqrt{K}\,U+\|\mathbf{u}-\mathbf{s}\|_2^2.$$

The GDV algorithm can also be applied in the situation in which we assume the trial sequence to be noise-free, e.g., $y_t = \mathbf{u} \cdot \mathbf{x}_t$ for all t.

THEOREM 5.5. *Let* $\mathbf{u} \in \mathbf{R}^N$ *be arbitrary, and consider a trial sequence* $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ *in which* $y_t = \mathbf{u} \cdot \mathbf{x}_t$ *holds for all* t. *We then have*

$$\begin{aligned} &\text{Loss}(\text{GDV}(\mathbf{s}, 1/2), S) \leqslant \|\mathbf{u} - \mathbf{s}\|_2^2 \max_t \|\mathbf{x}_t\|_2^2 \text{ and} \\ &\text{Loss}'(\text{GVD}(\mathbf{s}, 1/2), S) \leqslant \|\mathbf{u} - \mathbf{s}\|_2^2, \end{aligned} \tag{5.12}$$

for all start vectors $\mathbf{s}$.

Proof. First note that if at trial t we have $\|\mathbf{x}_t\|_2 = 0$, then $y_t = \mathbf{u} \cdot \mathbf{x}_t = \mathbf{w}_t \cdot \mathbf{x}_t = 0$ regardless of $\mathbf{u}$ and $\mathbf{w}_t$, and the algorithm makes no update. Hence, without loss of generality we can assume that $\|\mathbf{x}_t\|_2 > 0$ holds for all t.

Let $c \geqslant 0$ be an arbitrary parameter, and let $\mathbf{w}_t^c$ be the tth weight vector of GDV($\mathbf{s}$, $c/(1+2c)$) on trial sequence S. By applying Lemma 5.1 with $X = \|\mathbf{x}_t\|_2$ and $b = c/\|\mathbf{x}_t\|_2^2$ and assuming $y_t = \mathbf{u} \cdot \mathbf{x}_t$, we get

$$\frac{1}{\|\mathbf{x}_t\|_2^2} \frac{c}{1+2c} (y_t - \mathbf{w}_t^c \cdot \mathbf{x}_t)^2 \leqslant \frac{1}{2} (\|\mathbf{u} - \mathbf{w}_t^c\|_2^2 - \|\mathbf{u} - \mathbf{w}_{t+1}^c\|_2^2).$$

Let $\mathbf{w}_t$ be the tth weight vector of GDV($\mathbf{s}$, 1/2) on trial sequence S. By considering the limit where c approaches ∞, we see that

$$(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 \leqslant \|\mathbf{x}_t\|_2^2 (\|\mathbf{u} - \mathbf{w}_t\|_2^2 - \|\mathbf{u} - \mathbf{w}_{t+1}\|_2^2).$$

In particular, we have $\|\mathbf{u} - \mathbf{w}_t\|_2^2 - - \|\mathbf{u} - \mathbf{w}_{t+1}\|_2^2 > 0$, and we get

$$(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 \leqslant (\max_t \|\mathbf{x}_t\|_2^2)(\|\mathbf{u} - \mathbf{w}_t\|_2^2 - \|\mathbf{u} - \mathbf{w}_{t+1}\|_2^2).$$

By adding these inequalities for $t = 1, ..., \ell$ and observing that $\|\mathbf{u} - \mathbf{w}_{\ell+1}\|_2 \geqslant 0$, we get the first inequality of (5.12). The second inequality is proven similarly. ∎

5.3. *Worst-Case Loss Bounds for* GP

In this subsection, we show how the worst-case upper bounds for the square loss of the GD algorithm imply similar bounds for the GP algorithm, which uses the update rule (4.10). Let $\text{avg}(\mathbf{x}) = \sum_{i=1}^N x_i/N$, and let $\mathbf{avg}(\mathbf{x})$ denote the N-dimensional vector in which each component has the same value $\text{avg}(\mathbf{x})$. Then for the square loss, the update rule (4.10) becomes

$$\mathbf{w}_{t+1} = \mathbf{w}_t - 2\eta(\hat{y}_t - y_t)(\mathbf{x}_t - \mathbf{avg}(\mathbf{x}_t)),$$

where $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$. The new weight vector $\mathbf{w}_{t+1}$ satisfies $\sum_{i=1}^N w_{t+1,i} = \sum_{i=1}^N w_{t,i}$. Hence, if the GP algorithm uses a start vector $\mathbf{s}$ with $\sum_{i=1}^N s_i = W$, then the algorithm maintains the invariant $\sum_{i=1}^N w_{t,i} = W$ for all trials t.

Consider now applying the algorithm GP($\mathbf{s}, \eta$), with $\sum_{i=1}^{N} s_i = W$, to a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$. Define a modified trial sequence S' by $S' = ((\mathbf{x}'_1, y'_1), ..., (\mathbf{x}'_\ell, y'_\ell))$, where $\mathbf{x}'_t = \mathbf{x}_t - \mathbf{avg}(\mathbf{x}_t)$ and $y'_t = y_t - W \operatorname{avg}(\mathbf{x}_t)$. Then it is easy to see that the weight vectors of the algorithm GD($\mathbf{s}, \eta$) on the trial sequence S' are the same as the weight vectors of the algorithm GP($\mathbf{s}, \eta$) algorithm on the trial sequence S. Further, if $\hat{y}_t$ is the tth prediction of GP($\mathbf{s}, \eta$) on the trial sequence S, then the tth prediction of GD($\mathbf{s}, \eta$) on the trial sequence S' is given by $\hat{y}_t - W \operatorname{avg}(\mathbf{x}_t)$. Hence, $\operatorname{Loss}(\mathrm{GP}(\mathbf{s}, \eta), S) = \operatorname{Loss}(\mathrm{GD}(\mathbf{s}, \eta), S')$. By applying Theorem 5.3, we obtain the following bounds.

COROLLARY 5.6. *For a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, let V be an upper bound such that $\|\mathbf{x}_t - \mathbf{avg}(\mathbf{x}_t)\|_2 \leqslant V$ holds for all t. With the learning rate $\eta = 1/(4V^2)$ and an arbitrary start vector $\mathbf{s} \in \mathbf{R}^N$, we have for all vectors $\mathbf{u}$ such that $\sum_{i=1}^{N} u_i = \sum_{i=1}^{N} s_i$ the bound*

$$\operatorname{Loss}(\mathrm{GP}(\mathbf{s}, \eta), S) \leqslant 2(\operatorname{Loss}(\mathbf{u}, S) + \|\mathbf{u} - \mathbf{s}\|_2^2 V^2). \tag{5.13}$$

Further, let K and U be arbitrary constants, and let

$$\eta = \frac{U}{2V\sqrt{K} + 2UV^2}.$$

Then for all $\mathbf{u} \in \mathbf{R}^N$ such that $\sum_{i=1}^{N} u_i = \sum_{i=1}^{N} s_i$, $\operatorname{Loss}(\mathbf{u}, S) \leqslant K$, and $\|\mathbf{u} - \mathbf{s}\|_2 \leqslant U$ hold, we have

$$\operatorname{Loss}(\mathrm{GP}(\mathbf{s}, \eta), S) \leqslant \operatorname{Loss}(\mathbf{u}, S) + 2\sqrt{K}\, UV + \|\mathbf{u} - \mathbf{s}\|_2^2 V^2. \tag{5.14}$$

Thus, if the values $x_{t,i}$ are concentrated close to their average value $\mathbf{avg}(\mathbf{x}_t)$, but the average values $\mathbf{avg}(\mathbf{x}_t)$ are large, and we know the value $\sum_{i=1}^{N} u_i$ for the comparison vectors $\mathbf{u}$ we wish to use, then the GP algorithm can make use of this additional knowledge and incur a lower loss than the GD algorithm would.

As with the GD algorithm, we define for GP a variant GPV with variable learning rates. The update rule of GPV is

$$\mathbf{w}_{t+1} = \mathbf{w}_t - 2 \frac{\eta}{\|\mathbf{x}_t - \mathbf{avg}(\mathbf{x}_t)\|_2^2} (\hat{y}_t - y_t)(\mathbf{x}_t - \mathbf{avg}(\mathbf{x}_t)),$$

and we have the following upper bound.

THEOREM 5.7. *Let $\mathbf{u} \in \mathbf{R}^N$ and $\mathbf{s} \in \mathbf{R}^N$ be such that $\sum_{i=1}^{N} u_i = \sum_{i=1}^{N} s_i$, and consider a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ in which $y_t = \mathbf{u} \cdot \mathbf{x}_t$ holds for all t. We then have*

$$\operatorname{Loss}(\mathrm{GPV}(\mathbf{s}, 1/2), S) \leqslant \|\mathbf{u} - \mathbf{s}\|_2^2 \max_t \|\mathbf{x}_t - \mathbf{avg}(\mathbf{x}_t)\|_2^2. \tag{5.15}$$

5.4. *Worst-Case Loss Bounds for* EG

In this subsection, we give worst-case upper bounds for the loss of the EG algorithm, which was derived in Section 4 using the relative entropy as a distance measure. Similar bounds were earlier proven by Littlestone *et al.* (1995) for their algorithm, which is related to ours but does not have an analogous derivation. Our bounds are lower than those of Littlestone *et al.* In particular, we have bounds of the form (5.3), which seem unobtainable for the algorithm of Littlestone *et al.*

Again, we start by proving an upper bound for the loss of the algorithm at a single trial in terms of the loss of a comparison vector $\mathbf{u}$ and the progress of the algorithm at that trial.

LEMMA 5.8. *Let* $\mathbf{w}_t$ *be the weight vector of* EG($\mathbf{s}, \eta$) *before trial* t *in a trial sequence* $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, *and let* $\mathbf{u} \in [0, 1]^N$ *be a vector with* $\sum_i u_i = 1$. *Consider an arbitrary trial* t. *Let* $R > 0$ *be an upper bound such that* $\max_i x_{t,i} - \min_i x_{t,i} \leqslant R$. *For any constants* a *and* b *such that* $0 < a \leqslant 2b/(2 + R^2 b)$, *and a learning rate* $\eta = 2b/(2 + R^2 b)$, *we have*

$$a(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - b(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \leqslant d_{\mathrm{re}}(\mathbf{u}, \mathbf{w}_t) - d_{\mathrm{re}}(\mathbf{u}, \mathbf{w}_{t+1}). \tag{5.16}$$

Proof. Let $\beta_t = e^{2\eta(y_t - \hat{y}_t)}$. Then $w_{t+1,i} = w_{t,i}\beta_t^{x_{t,i}} / \sum_j w_{t,j}\beta_t^{x_{t,j}}$, and we have

$$d_{\mathrm{re}}(\mathbf{u}, \mathbf{w}_t) - d_{\mathrm{re}}(\mathbf{u}, \mathbf{w}_{t+1}) = \sum_{i=1}^{N} u_i \ln \frac{w_{t+1,i}}{w_{t,i}} = \sum_{i=1}^{N} u_i x_{t,i} \ln \beta_t - \ln \sum_{i=1}^{N} w_{t,i}\beta_t^{x_{t,i}}.$$

Hence, (5.16) is equivalent with $F(\mathbf{w}_t, \mathbf{x}_t, \mathbf{w}_t \cdot \mathbf{x}_t, y_t, \mathbf{u} \cdot \mathbf{x}_t, \beta_t) \leqslant 0$ where (omitting now the subscript t)

$$F(\mathbf{w}, \mathbf{x}, \hat{y}, y, r, \beta) = \ln \sum_{i=1}^{N} w_i \beta^{x_i} - r \ln \beta + a(y - \hat{y})^2 - b(y - r)^2 \tag{5.17}$$

and $\beta_t = e^{2\eta(y_t - \hat{y}_t)}$.

Let now B be such that $B \leqslant x_{t,i} \leqslant B + R$ holds for $1 \leqslant i \leqslant N$. We then have $0 \leqslant (x_{t,i} - B)/R \leqslant 1$ for $1 \leqslant i \leqslant N$. The bound $\alpha^x \leqslant 1 - x(1 - \alpha)$ holds for $\alpha \geqslant 0$ and $0 \leqslant x \leqslant 1$, and is tight for $x = 0$ and $x = 1$. By applying this with $\alpha = \beta^R$, we obtain

$$\beta^{x_i} = \beta^B (\beta^R)^{(x_i - B)/R} \leqslant \beta^B \left(1 - \frac{x_i - B}{R}(1 - \beta^R)\right).$$

Using the above gives us

$$\ln \sum_{i=1}^{N} w_i \beta^{x_i} \leqslant B \ln \beta + \ln \left(1 - \frac{\mathbf{w} \cdot \mathbf{x} - B}{R}(1 - \beta^R)\right)$$

when $\sum_{i=1}^{N} w_i = 1$. Hence, we get $F(\mathbf{w}, \mathbf{x}, \mathbf{w} \cdot \mathbf{x}, y, r, \beta) \leqslant G(\mathbf{w} \cdot \mathbf{x}, y, r, \beta)$, where

$$G(\hat{y}, y, r, \beta) = B \ln \beta + \ln\left(1 - \frac{\hat{y} - B}{R}(1 - \beta^R)\right) - r \ln \beta + a(y - \hat{y})^2 - b(y - r)^2.$$

Note that the inequality is tight if, for instance, $N = 2$ and $\mathbf{x} = (B, B + R)$.

To obtain (5.16), it is now sufficient to show that $G(\hat{y}, y, r, \beta) \leqslant 0$ holds for all values of $\hat{y}$, y, and r, when $\beta = e^{2\eta(y - \hat{y})}$ with $\eta = 2b/(2 + R^2 b)$. Since $\partial^2 G(\hat{y}, y, r, \beta)/\partial r^2 = -2b < 0$, the value $G(\hat{y}, y, r, \beta)$ is maximized when r is such that $\partial G(\hat{y}, y, r, \beta)/\partial r = 0$. Solving this gives $r = y - \ln \beta/(2b)$. In particular, for $\beta = e^{2\eta(y - \hat{y})}$, we see that proving $G(\hat{y}, y, r, e^{2\eta(y - \hat{y})}) \leqslant 0$ for $r = y + \eta(\hat{y} - y)/b$ implies $G(\hat{y}, y, r, e^{2\eta(y - \hat{y})}) \leqslant 0$ for all values r. For $r = y + \eta(\hat{y} - y)/b$ we have $G(\hat{y}, y, r, e^{2\eta(y - \hat{y})}) = H(\hat{y}, y)$ where

$$H(\hat{y}, y) = 2\eta B(y - \hat{y}) + \ln\left(1 - \frac{\hat{y} - B}{R}(1 - e^{2\eta R(y - \hat{y})})\right)$$

$$- 2\eta y(y - \hat{y}) + \left(a + \frac{\eta^2}{b}\right)(y - \hat{y})^2.$$

It remains to show that $H(\hat{y}, y) \leqslant 0$. We apply the bound $\ln(1 - q(1 - e^p)) \leqslant pq + p^2/8$, which holds for $0 \leqslant q \leqslant 1$ and $p \in \mathbf{R}$ (Helmbold *et al.*, 1996b, Lemma 1). We get $H(\hat{y}, y) \leqslant S(\hat{y}, y)$ where

$$S(\hat{y}, y) = 2\eta B(y - \hat{y}) + 2\eta R(y - \hat{y})\frac{\hat{y} - B}{R} + \frac{1}{8}(2\eta R(y - \hat{y}))^2$$

$$- 2\eta y(y - \hat{y}) + \left(a + \frac{\eta^2}{b}\right)(y - \hat{y})^2$$

$$= \frac{(y - \hat{y})^2}{2b}((2 + R^2 b)\eta^2 - 4b\eta + 2ab).$$

Therefore, it remains to show $Q(\eta) \leqslant 0$ where $Q(\eta) = (2 + R^2 b)\eta^2 - 4b\eta + 2ab$. We easily see that $Q(\eta)$ is minimized for $\eta = 2b/(2 + R^2 b)$, and that for this value of η we have $Q(\eta) \leqslant 0$ if and only if $a \leqslant 2b/(2 + R^2 b)$. ∎

As with the GD algorithm, we can combine the bounds for individual trials to give a bound for the total loss of the algorithm. We introduce a parameter c, which is later chosen in a suitable way to balance the two terms in the loss bound.

LEMMA 5.9. *Let $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ be an arbitrary trial sequence and $R > 0$ an upper bound such that $\max_i x_{t,i} - \min_i x_{t,i} \leqslant R$ holds for all t. Let c be an arbitrary positive constant, and let $\eta = 2c/(R^2(2 + c))$. Then for any start vector $\mathbf{s} \in \mathbf{R}^N$ and comparison vector $\mathbf{u} \in \mathbf{R}^N$, we have the bound*

$$\mathrm{Loss}(\mathrm{EG}(\mathbf{s}, \eta), S) \leqslant \left(1 + \frac{c}{2}\right)\mathrm{Loss}(\mathbf{u}, S) + \left(\frac{1}{2} + \frac{1}{c}\right)R^2 d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}). \qquad (5.18)$$

Proof. Let $b = c/R^2$ and $a = 2b/(2 + R^2 b) = 2c/(R^2(2 + c))$. Let $\mathbf{w}_t$ be the tth weight vector of $\mathrm{EG}(\mathbf{s}, \eta)$ on the trial sequence S with $\eta = a$. Then (5.16) holds by Lemma 5.8, and therefore

$$\frac{2c}{2+c}(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - c(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \leqslant R^2(d_{\mathrm{re}}(\mathbf{u}, \mathbf{w}_t) - d_{\mathrm{re}}(\mathbf{u}, \mathbf{w}_{t+1})). \tag{5.19}$$

By adding the bounds (5.19) for $t = 1, ..., \ell$, we get

$$\frac{2c}{2+c}\,\mathrm{Loss}(\mathrm{EG}(\mathbf{s}, \eta), s) - c\,\mathrm{Loss}(\mathbf{u}, S) \leqslant R^2(d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) - d_{\mathrm{re}}(\mathbf{u}, \mathbf{w}_{l+1})) \leqslant R^2 d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}),$$

which is equivalent with (5.18). ∎

We now obtain actual loss bounds for the EG algorithm by choosing a suitable value c in Lemma 5.9. The simplest way is to balance the terms proportional to the loss of the comparison vector $\mathbf{u}$ and to the distance $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})$. If we have estimates K and D for these quantities, we can do a more careful analysis of the trade-off and obtain a tighter bound.

THEOREM 5.10. *For a trial sequence* $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, *let* $R > 0$ *be a bound such that* $\max_i x_{t,i} - \min_i x_{t,i} \leqslant R$ *holds for all* t. *Let* $\eta = 2/(3^2)$. *For any start vector* $\mathbf{s} \in [0, 1]^N$ *and comparison vector* $\mathbf{u} \in [0, 1]^N$ *with* $\sum_{i=1}^n s_i = \sum_{i=1}^N u_i = 1$, *we have the bound*

$$\mathrm{Loss}(\mathrm{EG}(_{\mathrm{s}}, \eta), S) \leqslant \tfrac{3}{2}(\mathrm{Loss}(_{\mathrm{u}}, S) + R^2 d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})). \tag{5.20}$$

Further, let K *and* D *be arbitrary constants, and let*

$$\eta = \frac{2\sqrt{S}}{\sqrt{2K} + E^2\sqrt{D}}. \tag{5.21}$$

If then additionally $\mathrm{Loss}(\mathbf{u}, S) \leqslant K$ *and* $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) \leqslant D$ *hold, we have*

$$\mathrm{Loss}(\mathrm{EG}(\mathbf{s}, \eta), S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + \sqrt{2KD} + \frac{R^2 d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})}{2}. \tag{5.22}$$

Typically, we apply EG with the start vector $\mathbf{s} = (1/N, ..., 1/N)$. In this case, we have $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) = \ln N - H(\mathbf{u})$, where $H(\mathbf{u}) = -\sum_{i=1}^N u_i \ln u_i$ is the entropy of $\mathbf{u}$. Since the entropy is always positive, we then have $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) \leqslant \ln N$.

Proof. We apply Lemma 5.9. With the choice $c = 1$, the bound (5.18) simplifies to (5.20), and the bound is achieved by applying the learning rate $\eta = 2/(3R^2)$.

Let now K and D be such that $\mathrm{Loss}(\mathbf{u}, S) \leqslant K$ and $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) \leqslant D$. Then (5.18) implies

$$\mathrm{Loss}(\mathrm{EG}(\mathbf{s}, \eta), S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + \frac{R^2 d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})}{2} + F(c), \tag{5.23}$$

where $F(c) = Kc/2 + R^2 D/c$. Assume first that $K > 0$. As $F'(c) = K/2 - R^2 D/c^2$, we then have $F'(c) = 0$ for $c = R\sqrt{2D/K}$. Since $F''(c) > 0$ for all c, the value $F(c)$ is minimized for $c = R\sqrt{2D/K}$. Substituting this value of c into (5.23) yields (5.22). The special case $K = 0$ follows by considering the limit where c approaches ∞. ∎

As for the GD algorithm, a simple dimension analysis provides a crude check for the learning rates given in Theorem 5.10. First note that due to the update, the weights $w_{t,i}$ of the EG algorithm are always dimensionless. This is a natural consequence of requiring their sum to be 1. Hence, the predictions $\hat{y}_t$ have the same dimension as the input variables $x_{t,i}$, and therefore the outcomes y_t must also have this dimension. Let $[x]$ denote this common dimension. Then the dimension of the learning rate η must be $[x]^{-2}$ in order to make the exponent in the update factor $r_{t,i} = e^{2\eta(y_t - \hat{y}_t) x_{t,i}}$ dimensionless. This is true for the learning rates given in Theorem 5.10, since the quantity D is dimensionless and the quantities R and $\sqrt{K}$ have the dimension $[x]$.

Note that EG requires that the start vector and the hypotheses are probability vectors. By doubling the number of components we can allow negative weights as well. The resulting algorithm EG$^{\pm}$ is our main competitor for the standard gradient descent algorithm GD. The EG$^{\pm}$ algorithm still requires a parameter U such that we use only comparison vectors $\mathbf{u}$ with $\|\mathbf{u}\|_1 \leqslant U$. Let $\mathbf{u} \in \mathbf{R}^N$ be an arbitrary weight vector. We define two weight vectors with only positive weights, $\mathbf{u}^+$ and $\mathbf{u}^-$, by setting $u_i^+ = u_i$ if $u_i > 0$ and $u_i^+ = 0$ otherwise, and $u_i^- = -u_i$ if $u_i < 0$ and $u_i^- = 0$ otherwise. Then $\mathbf{u} = \mathbf{u}^+ - \mathbf{u}^-$. Given an instance vector $\mathbf{x} \in \mathbf{R}^N$, if we define $\mathbf{u}' = (u_1^+, ..., u_N^+, u_1^-, ..., u_N^-) \in [0, \infty)^N$ and $\mathbf{x}' = (x_1, ..., x_N, -x_1, ..., -x_N) \in \mathbf{R}^N$, we have $\mathbf{u}' \cdot \mathbf{x}' = \mathbf{u} \cdot \mathbf{x}$. Thus, the $2N$-dimensional vector $\mathbf{u}'$ with only positive weights represents the same linear function as $\mathbf{u}$, assumed that the instances $\mathbf{x}$ are duplicated before taking the dot product with the weight vector.

For the vector $\mathbf{u}'$ defined above, we have $\|\mathbf{u}'\|_1 = \|\mathbf{u}\|_1$. If we wish to define a weight vector $\mathbf{u}''$ with $\|\mathbf{u}''\|_1 = U > \|\mathbf{u}\|_1$, we can simply set $u_i'' = u_i' + (U - \|\mathbf{u}\|_1)/(2N)$ for $i = 1, ..., 2N$. That is, we distribute the excess weight $U - \|\mathbf{u}\|_1$ uniformly to the components of $\mathbf{u}''$. We can also distribute the excess weight non-uniformly. We need to only maintain the relations $u_i'' - u_{i+N}'' = u_i$ for $i = 1, ..., N$. Thus, given a weight vector $\mathbf{u} \in \mathbf{R}^N$ with $\|\mathbf{u}\|_1 \leqslant U$, we say that a vector $\mathbf{u}'' \in [0, U]^{2N}$ is a *norm U representation* of $\mathbf{u}$ if $\|\mathbf{u}''\|_1 = U$ and $u_i'' - u_{i+N}'' = u_i$ for $i = 1, ..., N$. We now see how this reduction can be used to obtain an upper bound for the loss of the EG$^{\pm}$ algorithm, with positive and negative weights, from the known upper bounds for the EG algorithm.

THEOREM 5.11. *Let $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ be a trial sequence and $X > 0$ a bound such that $\|\mathbf{x}_t\|_\infty \leqslant X$ holds for all t. Let $\mathbf{u} \in \mathbf{R}^N$ be an arbitrary weight vector with $\|\mathbf{u}\|_1 \leqslant U$, and let $\mathbf{u}' \in [0, U]^{2N}$ be an arbitrary norm U representation for $\mathbf{u}$. Let $\mathbf{s} = (\mathbf{s}^+, \mathbf{s}^-) \in [0, 1]^N \times [0, 1]^N$ be a pair of start vectors with $\sum_{i=1}^N (s_i^+ + s_i^-) = 1$, and let $\mathbf{s}' = (s_1^+, ..., s_N^+, s_1^-, ..., s_N^-)$. For the learning rate $\eta = 1/(3U^2X^2)$ we have*

$$\mathrm{Loss}(\mathrm{EG}^{\pm}(U, \mathbf{s}, \eta)) \leqslant 3(\mathrm{Loss}(\mathbf{u}, S) + U^2X^2 d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}')). \tag{5.24}$$

Further, let K and D be positive constants, and let

$$\eta = \frac{\sqrt{D}}{UX\sqrt{2K} + 2U^2X^2\sqrt{D}}.$$

For all weight vectors $\mathbf{u} \in \mathbf{R}^N$ and all norm U representations $\mathbf{u}'$ of $\mathbf{u}$, if $\mathrm{Loss}(\mathbf{u}, S) \leqslant K$ and $d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}') \leqslant D$ hold, we have

$$\mathrm{Loss}(\mathrm{EG}^{\pm}(U, \mathbf{s}, \eta), S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + 2UX\sqrt{2KD} + 2U^2X^2 d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}'). \qquad (5.25)$$

Finding a norm U representation $\mathbf{u}'$ of $\mathbf{u}$ that minimizes the relative entropy $d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}')$ seems to be nontrivial. However, for the uniform start vector this relative entropy is always at most $\ln 2N$, and using $D = \ln 2N$ leads to reasonable bounds.

Proof. We define a new trial sequence $S' = ((\mathbf{x}'_1, y_1), ..., (\mathbf{x}'_\ell, y_\ell))$ by setting $\mathbf{x}'_t = (Ux_{t,1}, ..., Ux_{t,N}, -Ux_{t,1}, ..., -Ux_{t,N})$. The algorithm $\mathrm{EG}^{\pm}$ has been defined in such a way that the predictions produced by $\mathrm{EG}^{\pm}(U, \mathbf{s}, \eta)$ on the trial sequence S are the same as those produced by $\mathrm{EG}(\mathbf{s}', \eta)$ on the trial sequence S'. In particular, $\mathrm{Loss}(\mathrm{EG}^{\pm}(U, \mathbf{s}, \eta), S) = \mathrm{Loss}(\mathrm{EG}(\mathbf{s}', \eta), S')$. We further note that $\mathrm{Loss}(\mathbf{u}'/U, S') = \mathrm{Loss}(\mathbf{u}, S)$ and $\max_i x'_{t,i} - \min_i x'_{t,i} = 2U \|\mathbf{x}_t\|_\infty$ for all t. Therefore, the bound (5.25), and the learning rates that achieve this bound, follow directly from the corresponding part of Theorem 5.10.

To obtain (5.24), we apply Lemma 5.9 to the trial sequence S' and comparison vector $\mathbf{u}'/U$ with $R = 2UX$. Then the bound (5.18) yields

$$\mathrm{Loss}(\mathrm{EG}^{\pm}(U, \mathbf{s}, \eta), S) \leqslant \left(1 + \frac{c}{2}\right) \mathrm{Loss}(\mathbf{u}, S) + \left(2 + \frac{4}{c}\right) U^2X^2 d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}'),$$

and the bound (5.24) follows by choosing $c = 4$. The resulting learning rates satisfies $\eta = 1/(3U^2X^2)$. ∎

To check the dimension of the learning rates, let again the dimension of the input variables be $[x]$ and of the outcomes $[y]$. Then the dimension of the weights $w_{t,i}$, and of the parameter U, is $[x]^{-1}[y]$. The update includes exponentiating the value $2\eta(y_t - \hat{y}_t)Ux_{t,i}$. Hence, this value should be dimensionless, which means that the dimension of the learning rate η should be $[y]^{-2}$. Since the quantity D in Theorem 5.11 is dimensionless, and the quantity K has dimension $[y]$, this is the case for the learning rates given in Theorem 5.11.

Recall that we defined the algorithm $\mathrm{EGV}^{\pm}$ as a modification of $\mathrm{EG}^{\pm}$ in which η is replaced by $\eta/\|\mathbf{x}_t\|_\infty$ in the update after trial t; in other words, the formulas (3.10) and (3.11) are replaced by (3.12) and (3.13). Like with GD and GDV, there are some situations in which we can obtain loss bounds for $\mathrm{EGV}^{\pm}$ from our bounds for $\mathrm{EG}^{\pm}$. First, we can obtain an upper bound for the scaled loss Loss′ defined for a weight vector $\mathbf{u}$ by

$$\mathrm{Loss}'(\mathbf{u}, S) = \sum_{t=1}^{\ell} \left(\frac{y_t - \mathbf{u} \cdot \mathbf{x}_t}{\|\mathbf{x}_t\|_\infty}\right)^2,$$

and generalized to define $\mathrm{Loss}'(\mathrm{EG}^{\pm}(U, \mathbf{s}, \eta))$ is the obvious manner. Here we again omit from the loss calculation the trials with $\|\mathbf{x}_t\|_\infty = 0$. A reduction analogous to the one applied to obtain Corollary 5.4 gives the following result.

COROLLARY 5.12. *Let $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ be an arbitrary trial sequence. Let $\mathbf{u} \in \mathbf{R}^N$ be an arbitrary vector, with U a bound such that $\|\mathbf{u}\|_1 \leqslant U$, and let $\mathbf{u}' \in [0, U]^{2N}$ be an arbitrary norm U representation for $\mathbf{u}$. Let $\mathbf{s} = (\mathbf{s}^+, \mathbf{s}^-) \in [0, 1]^N \times [0, 1]^N$ be a start vector pair with $\sum_{i=1}^N (s_i^+ + s_i^-) = 1$, and let $\mathbf{s}' = (s_1^+, ..., s_N^+, s_1^-, ..., s_N^-)$. We then have*

$$\mathrm{Loss}'(\mathrm{EGV}^{\pm}(U, \mathbf{s}, 1/(3U^2)), S) \leqslant 3(\mathrm{Loss}'(\mathbf{u}, S) + U^2 d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}')).$$

Further, let K and U be arbitrary constants, and let

$$\eta = \frac{\sqrt{D}}{U\sqrt{2K} + 2U^2\sqrt{D}}.$$

Then for all $\mathbf{u} \in \mathbf{R}^N$ such that $\mathrm{Loss}'(\mathbf{u}, S) \leqslant K$ and $d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}') \leqslant U$ hold, we have

$$\mathrm{Loss}'(\mathrm{EGV}^{\pm}(\mathbf{s}, \eta), S) \leqslant \mathrm{Loss}'(\mathbf{u}, S) + 2U\sqrt{2KD} + 2U^2 d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}').$$

Second, we can apply $\mathrm{EGV}^{\pm}$ in the noise-free case.

THEOREM 5.13. *Let $\mathbf{u} \in \mathbf{R}^N$ be an arbitrary vector, with U a bound such that $\|\mathbf{u}\|_1 \leqslant U$, and let $\mathbf{u}' \in [0, U]^{2N}$ be an arbitrary norm U representation for $\mathbf{u}$. Consider a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ in which $y_t = \mathbf{u} \cdot \mathbf{x}_t$ holds for all t. Let $\mathbf{s} = (\mathbf{s}^+, \mathbf{s}^-) \in [0, 1]^N \times [0, 1]^N$ be a start vector pair with $\sum_{i=1}^N (s_i^+ + s_i^-) = 1$, and let $\mathbf{s}' = (s_1^+, ..., s_N^+, s_1^-, ..., s_N^-)$. We then have*

$$\mathrm{Loss}(\mathrm{EGV}^{\pm}(U, \mathbf{s}, 1/(2U^2)), S) \leqslant 2U^2 (\max_t \|\mathbf{x}_t\|_\infty)^2 d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}') \text{ and}$$

$$\mathrm{Loss}'(\mathrm{EGV}^{\pm}(U, \mathbf{s}, 1/(2U^2)), S) \leqslant 2U^2 d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s}').$$

Proof. The proof is analogous with the proof on Theorem 5.5; we omit the details. ∎

5.5. *Worst-Case Loss Bounds for* EGU

We now consider the EGU algorithm introduced in Subsection 4.3. This algorithm uses a multiplicative update similar to that of the EG algorithm. The difference to the EG algorithm is that in the EGU algorithm, the total weight $\sum_{i=1}^N w_{t,i}$ is not kept constant. Accordingly, the EGU algorithm is useful when we wish to allow comparison vectors $\mathbf{u}$ for which the norm $\|\mathbf{u}\|_1$ is not known.

For the EGU algorithm, we have been able to prove worst-case loss bounds of the form (5.3) only in the case that all the outcomes and the input variables are positive, and the comparison vectors have only positive components. Preliminary experiments suggest that the algorithm works well also when the input variables can be negative, but much work remains to be done on this. For our proof it is also necessary to restrict the range of the predictions and outcomes. Thus, we give an

additional parameter Y to the algorithm, with the understanding that the outcomes are in the range $[0, Y]$. We write EGU($\mathbf{s}, Y, \eta$) for the EGU algorithm that has a start vector $\mathbf{s}$ and learning rate function η, and predicts with $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$ if $\mathbf{w}_t \cdot \mathbf{x}_t \leqslant Y$ holds and with $\hat{y}_t = Y$ otherwise.

As usual, we start with a technical lemma.

LEMMA 5.14. *Let $\mathbf{w}_t$ be the tth weight vector and $\hat{y}_t$ the tth prediction of* EGU($\mathbf{s}, Y, \eta$) *in a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{w}_\ell, y_\ell))$, and let $\mathbf{u} \in [0, \infty)^N$ be arbitrary. Consider an arbitrary trial t. Let $X > 0$ be such that $0 \leqslant x_{t,i} \leqslant X$ holds for all i, and assume that $0 \leqslant y_t \leqslant Y$ holds. For constants a and b such that $0 < a \leqslant b/(1 + 2XYb)$, and the learning rate $\eta = b/(1 + 2XYb)$, we have*

$$a(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - b(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \leqslant d_{\text{reu}}(\mathbf{u}, \mathbf{w}_t) - d_{\text{reu}}(\mathbf{u}, \mathbf{w}_{t+1}). \tag{5.26}$$

For any constants a and b such that $0 < b/(1 + 2XYb) < a$ and for any learning rate function η, there are a weight vector $\mathbf{w}_t$, comparison vector $\mathbf{u} \in [0, \infty)^N$, and an outcome y_t such that (5.26) *does not hold for $N = 2$ and $\mathbf{x}_t = (0, X)$.*

The proof of Lemma 5.14 is similar to the proof of Lemma 5.8, but somewhat more complicated; for details, see the Appendix.

As with the algorithms GD and EG, we now combine the single trial bounds given by Lemma 5.14.

LEMMA 5.15. *Consider a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ with $\mathbf{x}_t \in [0, X]^N$ and $y_t \in [0, Y]$ for all t for some constants $X > 0$ and $Y > 0$. Let c be an arbitrary positive constant, and let $\eta = c/(XY(1 + 2c))$. Then for all start vectors $\mathbf{s} \in [0, \infty)^N$ and comparison vectors $\mathbf{u} \in [0, \infty)^N$ we have the bound*

$$\text{Loss}(\text{EGU}(\mathbf{s}, Y, \eta), S) \leqslant (1 + 2c)\,\text{Loss}(\mathbf{u}, S) + \left(2 + \frac{1}{c}\right) XY d_{\text{reu}}(\mathbf{u}, \mathbf{s}). \tag{5.27}$$

Proof. For $t = 1, ..., \ell$, let $b = c/XY$ and $a = b/(1 + 2XYb) = c/(XY(1 + 2c))$. Let $\mathbf{w}_t$ be the tth weight vector of EGU($\mathbf{s}, Y, \eta$) on the trial sequence S with η such that $\eta(\mathbf{x}_t) = a$. Then (5.26) holds by Lemma 14, and therefore

$$\frac{c}{1 + 2c}(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - c(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \leqslant XY(d_{\text{reu}}(\mathbf{u}, \mathbf{w}_t) - d_{\text{reu}}(\mathbf{u}, \mathbf{w}_{t+1})). \tag{5.28}$$

By adding the bounds (5.28) for $t = 1, ..., \ell$, we get

$$\begin{aligned} &\frac{c}{1 + 2c}\,\text{Loss}(\text{EGU}(\mathbf{s}, Y, \eta), S) - c\,\text{Loss}(\mathbf{u}, S) \\ &\quad \leqslant XY(d_{\text{reu}}(\mathbf{u}, \mathbf{s}) - d_{\text{reu}}(\mathbf{u}, \mathbf{w}_{\ell+1})) \\ &\quad \leqslant XY d_{\text{reu}}(\mathbf{u}, \mathbf{s}), \end{aligned}$$

which is equivalent with (5.27). ∎

Finally, we show suitable values for c for obtaining good loss bounds from Lemma 5.15.

THEOREM 5.16. *Consider a trial sequence* $S=((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$ *with* $\mathbf{x}_t \in [0, X]^N$ *and* $y_t \in [0, Y]$ *for all* t *for some constants* X *and* Y. *With the learning rate* $\eta = 1/(3XY)$ *and an arbitrary start vector* $\mathbf{s} \in [0, \infty)^N$, *we have for any vector* $\mathbf{u} \in [0, \infty)^N$ *the bound*

$$\mathrm{Loss}(\mathrm{EGU}(\mathbf{s}, Y, \eta), S) \leqslant 3(\mathrm{Loss}(\mathbf{u}, S) + XYd_{\mathrm{reu}}(\mathbf{u}, \mathbf{s})). \tag{5.29}$$

Further, let K *and* D *be arbitrary constants, and let*

$$\eta = \frac{\sqrt{D}}{\sqrt{2KXY} + 2XY\sqrt{D}}. \tag{5.30}$$

If then additionally $\mathrm{Loss}(\mathbf{u}, S) \leqslant K$ *and* $d_{\mathrm{reu}}(\mathbf{u}, \mathbf{s}) \leqslant D$ *hold, we have*

$$\mathrm{Loss}(\mathrm{EGU}(\mathbf{s}, Y, \eta), S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + 2\sqrt{2KXYD} + 2XYd_{\mathrm{reu}}(\mathbf{u}, \mathbf{s}). \tag{5.31}$$

Proof. We apply Lemma 5.15. With the choice $c=1$, the bound (5.27) simplifies to (5.29), and we get $\eta = 1/(3XY)$. Let now K and D be such that $\mathrm{Loss}(\mathbf{u}, S) \leqslant K$ and $d_{\mathrm{reu}}(\mathbf{u}, \mathbf{s}) \leqslant D$. Assume first $K > 0$. Then (5.27) implies

$$\mathrm{Loss}(\mathrm{EGU}(\mathbf{s}, Y, \eta), S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + 2XYd_{\mathrm{reu}}(\mathbf{u}, \mathbf{s}) + F(c), \tag{5.32}$$

where $F(c) = 2Kc + XYD/c$. Then $F'(c) = 2K - XYD/c^2$, and $F'(c) = 0$ for $c = \sqrt{XYD/(2K)}$. Since $F''(c) > 0$ for all c, the value $F(c)$ is minimized for $c = \sqrt{XYD/(2K)}$. Substituting this value of c into (5.32) yields (5.31), and the learning rate η for this c satisfies (5.30). In the special case $K=0$ we consider limits as c approaches infinity, as we did in the proof of Theorem 5.3. ∎

To check the dimensions of the learning rates, let the dimension of the input variables be $[x]$ and of the outcomes $[y]$. Then the quantity $2\eta(y_t - \hat{y}_t)x_{t,i}$ that appears exponentiated in the update rule is dimensionless if the dimension of η is $[x]^{-1}[y]^{-1}$. In Theorem 5.16, the dimension of the quantity D is $[x]^{-1}[y]$ and the dimension of K is $[y]^2$, so the learning rates satisfy this condition.

6. LOWER BOUNDS

We first consider the case where the instances $\mathbf{x}_t$ and the target $\mathbf{u}$ satisfy norm constraints $\|\mathbf{u}\|_p \leqslant U$ and $\|\mathbf{x}_t\|_q \leqslant X$ for some p and q in $\mathbf{R}_+ \cup \{\infty\}$, but the outcomes y_t can be arbitrary. Recall that the norms L_p and L_q are *dual* if $1/p + 1/q = 1$. Hence, the L_2 norm is its own dual, and the L_1 norm is the dual of L_∞ norm. If the norms L_p and L_q are dual, then the Cauchy–Schwartz Inequality can be generalized to show that $\|\mathbf{u}\|_p \leqslant U$ and $\|\mathbf{x}\|_q \leqslant X$ together imply $|\mathbf{u} \cdot \mathbf{x}| \leqslant UX$ (Royden, 1963).

THEOREM 6.1. *Let $p, q \in \mathbf{R}_+ \cup \{\infty\}$. Let A be an arbitrary on-line prediction algorithm, and let K, U, and X be arbitrary positive reals. Then for all $N \in \mathbf{N}_+$ there are an instance $\mathbf{x} \in \mathbf{R}^N$ with $\|\mathbf{x}\|_q = X$, an outcome $y \in \mathbf{R}$, and a comparison vector $\mathbf{u} \in \mathbf{R}^N$ with $\|\mathbf{u}\|_p = U$, such that for the 1-trial sequence $S = ((\mathbf{x}, y))$ we have* $\mathrm{Loss}(\mathbf{u}, S) = K$ *and*

$$\mathrm{Loss}(A, S) \geqslant K + 2c_N UX\sqrt{K} + (c_N UX)^2$$

where $c_N = N^{1-1/p-1/q}$. In particular, if $1/p + 1/q = 1$ then $c_N = 1$, and if $1/p + 1/q < 1$ then $\lim_{N\to\infty} c_N = \infty$.

Proof. We define two potential target vectors $\mathbf{u}_+ = (UN^{-1/p}, ..., UN^{-1/p})$ and $\mathbf{u}_- = -\mathbf{u}_+$, and an instance vector $\mathbf{x} = (XN^{-1/q}, ..., XN^{-1/q})$. Then $\|\mathbf{u}_+\|_p = \|\mathbf{u}_-\|_p = U$, $\|\mathbf{x}\|_q = X$, and $\mathbf{u} \cdot \mathbf{x} = UXN^{1-1/p-1/q}$. Let $\hat{y}$ be the prediction of the algorithm A, when it sees the instance $\mathbf{x}$ at the first trial. We further choose $y = UXN^{1-1/p-1/q} + \sqrt{K}$ if $\hat{y} \leqslant 0$ and $y = -UXN^{1-1/p-1/q} - \sqrt{K}$ otherwise. Then either $\mathrm{Loss}(\mathbf{u}_+, S) = K$ or $\mathrm{Loss}(\mathbf{u}_-, S) = K$. Since $\mathrm{Loss}_L(A, S) \geqslant y^2$, we get the stated bound. ∎

The special case $p = q = 2$ of Theorem 6.1 was noted already by Cesa-Bianchi *et al.* (1996). The lower bound given in Theorem 6.1 for this case coincides with the upper bound given in Theorem 5.3 for the GD algorithm. Hence, the GD algorithm has the best obtainable worst case loss bound.

Note that in Theorem 6.1, K cannot be made arbitrarily large without also making the absolute value of the outcome arbitrarily large. The following lower bound, also from Cesa-Bianchi *et al.* (1996), shows that if the number N of dimensions can be arbitrarily large, then again the loss bound for GD is the best possible, even if range of the outcomes is restricted. For a comparison vector $\mathbf{u}$ and instances $\mathbf{x}_t$, the range of the outcomes is $[-UX, UX]$, where $U = \|\mathbf{u}\|_2$ and $X = \max_t \|\mathbf{x}_t\|_2$. Since $UX = \max\{\mathbf{u} \cdot \mathbf{x} \mid \|\mathbf{u}\|_2 = U,\ \|\mathbf{x}\|_2 = X\}$, this is a natural range for the outcomes.

THEOREM 6.2. *Let U, X, and K be arbitrary positive reals, and let the dimension N be at least $(1 + \sqrt{K}/(UX))^2$. Let A be an arbitrary on-line prediction algorithm. There is a comparison vector $\mathbf{u} \in \mathbf{R}^N$, with $\|\mathbf{u}\|_2 = U$, and a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N))$, with $\mathbf{x}_t \in \{-X, X\}^N$ and $y_t \in [-UX, UX]$ for all t, such that* $\mathrm{Loss}(\mathbf{u}, S) = K$ *and*

$$\mathrm{Loss}(A, S) \geqslant K + 2UX\sqrt{K} + (UX)^2.$$

Consider now the lower bound of Theorem 6.1 for $p = 1$ and $q = \infty$, which is the case related to the EG$^{\pm}$ algorithm. The lower bound has $c_N = 1$ for all N. However, the upper bound for the EG$^{\pm}$ algorithm in Theorem 5.11 includes the factors $\sqrt{2D}$ and $2d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s})$, which can grow logarithmically in N. Thus, for large N there is a significant gap between the upper and lower bounds. We would like to know if it is possible to improve the upper bounds by eliminating the $\ln N$ factors. In the general case, we have had no success in solving this problem. We now present two

partial results that hint that our upper bounds may be reasonably tight. We consider the upper bounds for the simpler EG algorithm, from which the bounds for $EG^{\pm}$ are obtained via a reduction. If we were able to improve the bounds for EG, then an improvement for $EG^{\pm}$ would automatically follow.

The following result of Littlestone *et al.* (1995) shows that in the case $\mathrm{Loss}(\mathbf{u}, S) = 0$, a factor $\ln N$ in the loss of the algorithm cannot be avoided. For simplicity, we consider only the case $\mathbf{x}_t \in [0, 1]^N$. For the case $\mathrm{Loss}(\mathbf{u}, S) = K > 0$, the lower bound in this results does not come close to the upper bound, as it does not contain a term proportional to $\sqrt{K}$. It remains an open question whether the $\sqrt{K}$ term can be avoided if the range of the outcomes is $[-UX, UX]$.

THEOREM 6.3. *Let k and N be positive integers, with $k \leqslant N$. Let K be an arbitrary positive real, and let A be an arbitrary on-line prediction algorithm. There is a target vector $\mathbf{u} \in [0, 1]^N$, with $\sum_{i=1}^N u_i = 1$, and a trial sequence $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, with $\mathbf{x}_t \in [0, 1]^N$ and $y_t \in [0, 1]$ for all t, such that $\mathrm{Loss}(\mathbf{u}, S) = K$ and*

$$\mathrm{Loss}(A, S) \geqslant K + \frac{\ln N - \ln k}{4 \ln 2} - \frac{1}{2}.$$

The comparison vectors $\mathbf{u}$ used in the proof Theorem 6.3 contain k components with value $1/k$, with the rest of the components having the value 0. Hence, for the uniform vector $\mathbf{s} = (1/N, ..., 1/N)$ we have $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) = \ln N - \ln k$.

The next theorem shows that if we consider only upper bounds of the form used in Theorem 5.10, then the constant coefficients given in the theorem are optimal. However, this leaves open the possibility that smaller coefficients could be obtained by inserting, for example, an additive constant term.

THEOREM 6.4. *Let A be an arbitrary prediction algorithm and let $\mathbf{s} = (1/2, 1/2)$. If p and q are constants such that*

$$\mathrm{Loss}(A, S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + p\sqrt{\mathrm{Loss}(\mathbf{u}, S)\, d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})} + q d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) \tag{6.1}$$

holds for all non-trial sequences $S = ((\mathbf{x}_1, y_1))$ with $\mathbf{x}_1 \in [0, 1]^2$ and $y_1 \in [0, 1]$, and all weight vectors $\mathbf{u} \in [0, 1]^2$ with $\sum_{i=1}^N u_i = 1$, then $p \geqslant \sqrt{2}$, and if $p = \sqrt{2}$ then $q \geqslant 1/2$.

Proof. We take $\mathbf{x}_1 = (0, 1)$ as the only instance in the sequence. Then the prediction of A at the first trial must be 1/2, or the weight vector $\mathbf{u} = \mathbf{s} = (1/2, 1/2)$ would violate the assumptions of the theorem for the outcome $y = 1/2$. Consider now $\mathbf{u} = (1/2 - \varepsilon, 1/2 + \varepsilon)$ for $0 < \varepsilon < 1/4$. Let the outcome of the trial be $y = 3/4 + \varepsilon$. Then $\mathrm{Loss}(\mathbf{u}, S) = 1/16$ and $\mathrm{Loss}(A, S) = 1/16 + \varepsilon/2 + \varepsilon^2$. On the other hand, we have

$$d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) = \ln 2 + (\tfrac{1}{2} + \varepsilon)\ln(\tfrac{1}{2} + \varepsilon) + (\tfrac{1}{2} - \varepsilon)\ln(\tfrac{1}{2} - \varepsilon) = 2\varepsilon^2 + O(\varepsilon^4),$$

and hence $\sqrt{d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})} = \sqrt{2}\varepsilon + O(\varepsilon^3)$. Therefore, in the case $\mathrm{Loss}(\mathbf{u}, S) = 1/16$ the right-hand side of (6.1) can be expanded as

$$\frac{1}{16} + \frac{p\sqrt{d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})}}{4} + q d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) = \frac{1}{16} + \frac{p\sqrt{2}}{4}\varepsilon + 2q\varepsilon^2 + O(\varepsilon^3).$$

Therefore, (6.1) cannot hold for for small ε unless p and q satisfy the stated bounds. ∎

7. BATCH PREDICTIONS

We consider generalizing the prediction problem into a setting in which at each trial, the prediction algorithm predicts for each of a batch of several instances and then receives the outcomes for all these instances. A *generalized trial sequence* is a sequence $((M_1, \mathbf{y}_1), ..., (M_\ell, \mathbf{y}_\ell))$, where for each t we have $M_t \in \mathbf{R}^{m_t \times N}$ and $\mathbf{y}_t \in \mathbf{R}^{m_t}$ for some m_t. We define $M_{t,i}$ to be the ith column of the tth instance matrix M_t. A prediction algorithm for generalized trial sequences is defined as with usual trial sequences, except that now the tth prediction $\hat{\mathbf{y}}_t$ is a vector in $\mathbf{R}^{m_t}$. To measure the loss of an algorithm, we now need a loss function from $\mathbf{R}^m \times \mathbf{R}^m$ to $[0, \infty)$. Here we consider only the square loss function defined by $L(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2$. The notions of the loss of an algorithm or a weight vector on a trial sequence are defined in the obvious way.

All of the algorithms introduced in Section 3 can be converted for generalized trial sequences in a straightforward manner. The predictions $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$ are naturally replaced by $\hat{\mathbf{y}}_t = M_t \mathbf{w}_t$. In the updates, we replace the derivatives

$$\frac{\partial L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t)}{\partial w_{t,i}} = L_{y_t}(\mathbf{w}_t \cdot \mathbf{x}_t) x_{t,i} \tag{7.1}$$

by

$$\frac{\partial L(\mathbf{y}_t, M_t \mathbf{w}_t)}{\partial w_{t,i}} = \left(\frac{\partial L(\mathbf{y}_t, \mathbf{z})}{\partial \mathbf{z}} \right)_{\mathbf{z} = M_t \mathbf{w}_t} \cdot M_{t,i}. \tag{7.2}$$

In particular, for the square loss the generalization of the GD algorithm, which we call the GDM algorithm, has the update rule

$$\mathbf{w}_{t+1} = \mathbf{w}_t - 2\eta M_t^{\mathrm{T}} (\hat{\mathbf{y}} - \mathbf{y}_t)$$

and the generalization for the EG algorithm, which we call the EGM algorithm, has the update rule $w_{t+1,i} = w_{t,i} r_{t,i} / (\sum_j w_{t,j} r_{t,j})$ where

$$r_{t,i} = \exp(-2\eta(\hat{\mathbf{y}} - \mathbf{y}_t) \cdot M_{t,i}). \tag{7.3}$$

It has been previously shown (Cesa-Bianchi *et al.*, 1996; Schapire and Warmuth, 1994) that the GDM algorithm has a loss bound similar to that of GD. Recall that the norm $\|A\|_2$ for a matrix A is defined as $\|A\|_2 = \max\{\|A\mathbf{x}\|_2 \mid \|\mathbf{x}\|_2 = 1\}$.

THEOREM 7.1. *Let $S = ((M_1, \mathbf{y}_1), ..., (M_\ell, \mathbf{y}_\ell))$ be a generalized trial sequence such that $\|M_t\|_2 \leqslant X$ for all t. For the batch prediction algorithm* GDM($\mathbf{s}, \eta$) *with the learning rate $\eta = 1/(4X^2)$, we have for all weight vectors* $\mathbf{u}$ *the bound*

$$\mathrm{Loss}(\mathrm{GDM}(\mathbf{s}, \eta), S) \leqslant 2(\mathrm{Loss}(\mathbf{u}, S) + \|\mathbf{s} - \mathbf{u}\|_2^2 X^2).$$

Assume further that we know bounds K and U such that for some weight vector $\mathbf{u}$ *we have* $\mathrm{Loss}(\mathbf{u}, S) \leqslant K$ *and* $\|\mathbf{s}-\mathbf{u}\|_2 \leqslant U$. *Then for the learning rate*

$$\eta = \frac{U}{2UX^2 + 2X\sqrt{K}}$$

we have

$$\mathrm{Loss}(\mathrm{GDM}(\mathbf{s}, \eta), S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + 2UX\sqrt{K} + \|\mathbf{s}-\mathbf{u}\|_2^2 X^2.$$

The proof of Theorem 7.1 is based on noticing that the proof of Theorem 5.3 for the GD algorithm easily generalizes to the situations where the instances are matrices instead of vectors. The upper bound of Theorem 7.1 can be shown to be tight (Schapire and Warmuth, 1994). We now give a similar result for the EGM algorithm. The proof is based on a reduction that allows us to apply directly the upper bound given in Theorem 5.10 for the EG algorithm. We could easily generalize result for the more general algorithm $\mathrm{EG}^{\pm}$ when it is applied to matrices. In the noise-free case $K=0$, similar results were given by Littlestone *et al.* (1995). The reduction could also be applied to the GD algorithm to obtain Theorem 7.1.

THEOREM 7.2. *Let* $S = ((M_1, \mathbf{y}_1), ..., (M_\ell, \mathbf{y}_\ell))$ *be a generalized trial sequence such that for all* t *and* i, *the* L_2 *norm of the* i*th column* $M_{t,i}$ *of the matrix* M_t *is at most* $R/2$, *i.e.*, $\|M_{t,i}\|_2 \leqslant R/2$. *For the batch prediction algorithm* $\mathrm{EGM}(\mathbf{s}, \eta)$, *with the learning rate* $\eta = 2/(3R^2)$, *and for any comparison vector* $\mathbf{u} \in [0,1]^N$ *with* $\sum_{i=1}^N u_i = 1$, *we have the bound*

$$\mathrm{Loss}(\mathrm{EGM}(\mathbf{s}, \eta), S) \leqslant \tfrac{3}{2}(\mathrm{Loss}(\mathbf{u}, S) + R^2 d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})).$$

Assume further that we know bounds K *and* U *such that for some* $\mathbf{u} \in [0,1]^N$ *with* $\sum_{i=1}^N u_i = 1$ *we have* $\mathrm{Loss}(\mathbf{u}, S) \leqslant K$ *and* $d_{\mathrm{re}}(\mathbf{u}, \mathbf{s}) \leqslant D$. *Then for the learning rate*

$$\eta = \frac{2\sqrt{D}}{R\sqrt{2K} + R^2\sqrt{D}} \tag{7.4}$$

we have

$$\mathrm{Loss}(\mathrm{EGM}(\mathbf{s}, \eta), S) \leqslant \mathrm{Loss}(\mathbf{u}, S) + R\sqrt{2KD} + \frac{R^2 d_{\mathrm{re}}(\mathbf{u}, \mathbf{s})}{2}.$$

Proof. We prove the theorem by reducing it to the upper bound given in Theorem 5.10 for the EG algorithm.

Let $\mathbf{w}_1, ..., \mathbf{w}_{\ell+1}$ be the sequence of weight vectors that $\mathrm{EGM}(\mathbf{s}, \eta)$ produces on the trial sequence S. We define a trial sequence $S' = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell))$, with $\mathbf{x}_t \in \mathbf{R}^N$ and $y_t \in \mathbf{R}$, such that on the trial sequence S', the algorithm $\mathrm{EG}(\mathbf{s}, \eta)$ also produces the sequence $\mathbf{w}_1, ..., \mathbf{w}_{\ell+1}$ of weight vectors. Let

$$x_{t,i} = \frac{M_t \mathbf{w}_t - \mathbf{y}_t}{\|M_t \mathbf{w}_t - \mathbf{y}_t\|_2} \cdot M_{t,i}$$

and

$$y_t = \frac{M_t \mathbf{w}_t - \mathbf{y}_t}{\|M_t \mathbf{w}_t - \mathbf{y}_t\|_2} \cdot \mathbf{y}_t.$$

For any weight vector $\mathbf{r}$ we then have

$$\mathbf{r} \cdot \mathbf{x}_t - y_t = \frac{(M_t \mathbf{w}_t - \mathbf{y}_t)}{\|M_t \mathbf{w}_t - \mathbf{y}_t\|_2} \cdot (M_t \mathbf{r} - \mathbf{y}_t). \tag{7.5}$$

Applying (7.5) for $\mathbf{r} = \mathbf{w}_t$ we get $\mathbf{w}_t \cdot \mathbf{x}_t - y_t = \|M_t \mathbf{w}_t - \mathbf{y}_t\|_2$, so $(\mathbf{w}_t \cdot \mathbf{x}_t - y_t) x_{t,i} = (M_t \mathbf{w}_t - \mathbf{y}_t) \cdot M_{t,i}$. Hence, when $\mathbf{x}_t$ and y_t are defined as given here, the gradients given in (7.1) and (7.2) have the same values. Thus, the weight vectors generated by EGM$(\mathbf{s}, \eta)$ on the sequence S are the same as the weight vectors generated by EG$(\mathbf{s}, \eta)$ on the sequence S'.

Further, we get $(\mathbf{w}_t \cdot \mathbf{x}_t - y_t)^2 = \|M_t \mathbf{w}_t - \mathbf{y}_t\|_2^2$ and, hence, Loss(EGM$(\mathbf{s}, \eta)$, S) = Loss(EG$(\mathbf{s}, \eta)$, S'). Applying (7.5) for $\mathbf{r} = \mathbf{u}$ we get

$$(\mathbf{u} \cdot \mathbf{x}_t - y_t)^2 = \frac{((M_t \mathbf{w}_t - \mathbf{y}_t) \cdot (M_t \mathbf{u} - \mathbf{y}_t))^2}{\|M_t \mathbf{w}_t - \mathbf{y}_t\|_2^2} \leqslant \|M_t \mathbf{u} - \mathbf{y}_t\|_2^2,$$

so Loss$(\mathbf{u}, S') \leqslant$ Loss$(\mathbf{u}, S)$. Further, since $\|M_{t,i}\|_2 \leqslant R/2$ implies $-R/2 \leqslant x_{t,i} \leqslant R/2$, by applying the upper bound of Theorem 5.10 with the learning rate $\eta = 2/(3R^2)$ we get

$$\text{Loss}(\text{EGM}(\mathbf{s}, \eta), S) = \text{Loss}(\text{EG}(\mathbf{s}, \eta), S') \leqslant \tfrac{3}{2}(\text{Loss}(\mathbf{u}, s) + R^2 d_{\text{re}}(\mathbf{u}, \mathbf{s}))$$

for any comparison vector $\mathbf{u} \in [0, 1]^N$ with $\sum_{i=1}^N u_i = 1$. Finally, assume that Loss$(\mathbf{u}, S) \leqslant K$ and $d_{\text{re}}(\mathbf{u}, \mathbf{s}) \leqslant D$ hold for some $\mathbf{u}$. We then get Loss$(\mathbf{u}, S') \leqslant K$ and, hence,

$$\begin{aligned}
\text{Loss}(\text{EGM}(\mathbf{s}, \eta), S) &= \text{Loss}(\text{EG}(\mathbf{s}, \eta), S') \\
&\leqslant \text{Loss}(\mathbf{u}, S') + R\sqrt{2KD} + \frac{R^2 d_{\text{re}}(\mathbf{u}, \mathbf{s})}{2} \\
&\leqslant \text{Loss}(\mathbf{u}, S) + R\sqrt{2KD} + \frac{R^2 d_{\text{re}}(\mathbf{u}, \mathbf{s})}{2}
\end{aligned}$$

for the learning rate given in (7.4). ∎

8. OBTAINING EXPECTED INSTANTANEOUS LOSS BOUNDS

So far the focus of this paper has been worst-case bounds for the total loss of on-line algorithms. We now show how from these worst-case total loss bounds we can derive bounds for the expected total loss and for the expected loss on the next instance.

We study only a very simple probabilistic model. As can be seen from the proofs, much weaker probabilistic assumption would lead to similar bounds. An *example* is a pair $e=(\mathbf{x}, y)$ that consists of an instance $\mathbf{x}\in\mathbf{R}^N$ and an outcome $y\in\mathbf{R}$. We assume that there is a fixed but unknown probability distribution D on the example domain $\mathbf{R}^N\times\mathbf{R}$, and that the examples are drawn independently at random from this distribution. The following bounds on the expected loss are still worst-case in the sense that we make no assumptions about the distribution D.

The first part of this section relies on the fact that once we have an inequality of the form of (5.2) which always holds, then by taking expectations of both sides we get for all ℓ the bound

$$\begin{aligned}\mathrm{E}_{S\sim D^\ell}(\mathrm{Loss}_L(A, S)) &\leqslant \frac{g(c)}{f(c)}\,\mathrm{E}_{S\sim D^\ell}(\mathrm{Loss}_L(\mathbf{u}, S)) + \frac{d(\mathbf{u}, \mathbf{s})}{f(c)}\\ &= \ell\,\frac{g(c)}{f(c)}\,\mathrm{E}_{e\sim D}(\mathrm{Loss}_L(\mathbf{u}, e)) + \frac{d(\mathbf{u}, \mathbf{s})}{f(c)}.\end{aligned}$$

The parameter c and thus the learning rate can now be optimized as a function of upper bounds on the expected loss $\mathrm{E}_{e\sim D}(\mathrm{Loss}_L(\mathbf{u}, e))$ of the vector $\mathbf{u}$ on a single example and the distance $d(\mathbf{u}, \mathbf{s})$. For example, in the case of the GD algorithm this leads to the following probabilistic version of Theorem 5.3.

THEOREM 8.1. *Let D be a probability distribution on $\{\mathbf{x} \mid \|\mathbf{x}\|_2\leqslant X\}\times\mathbf{R}$. With the learning rate $\eta=1/(4X^2)$ and an arbitrary start vector $\mathbf{s}\in\mathbf{R}^N$, we have for any vector $\mathbf{u}$ and all $\ell\geqslant 1$ the bound*

$$\mathrm{E}_{S\sim D^\ell}(\mathrm{Loss}(\mathrm{GD}(\mathbf{s}, \eta), S))\leqslant 2(\ell\,\mathrm{E}_{e\sim D}(\mathrm{Loss}(\mathbf{u}, e)) + \|\mathbf{u}-\mathbf{s}\|_2^2\,X^2). \tag{8.1}$$

Further, let K and U be arbitrary constants, and let η be the learning rate

$$\eta=\frac{U}{2X\sqrt{\ell K}+2UX^2}.$$

Then for all $\mathbf{u}\in\mathbf{R}^N$ such that $\mathrm{E}_{e\sim D}(\mathrm{Loss}(\mathbf{u}, e))\leqslant K$ and $\|\mathbf{u}-\mathbf{s}\|_2\leqslant U$ hold, we have

$$\mathrm{E}_{S\sim D^\ell}(\mathrm{Loss}(\mathrm{GD}(\mathbf{s}, \eta), S))\leqslant \ell\,\mathrm{E}_{e\sim D}(\mathrm{Loss}(\mathbf{u}, e)) + 2\sqrt{\ell K}\,UX + \|\mathbf{u}-\mathbf{s}\|_2^2\,X^2. \tag{8.2}$$

In many cases we are not interested in worst-case total loss bounds of an on-line algorithm but rather than that we are looking for a hypothesis which predicts well on a random instance. We define a *hypothesis* h to be any mapping from $\mathbf{R}^N$ to $\mathbf{R}$. The *instantaneous loss* $\mathrm{InstLoss}_L(h, D)$ of a hypothesis h with respect to a distribution D on $\mathbf{R}^{N+1}$ is defined as the expected loss when the hypothesis h is used to predict on a random instance drawn from D, that is,

$$\mathrm{InstLoss}_L(h, D)=\mathrm{E}_{(\mathbf{x}, y)\sim D}(L(h(\mathbf{x}), y)).$$

A common goal of learning is to produce a hypothesis with small instantaneous loss after seeing a reasonable number of examples. In the case when the instantaneous loss is measured with respect to to some distribution D, it is assumed that the training examples are drawn independently at random from the same distribution.

Giving a sequence $S=(e_1, ..., e_\ell)$ of ℓ examples to an on-line predicting algorithm naturally leads to $\ell+1$ hypothesis. For our linear on-line prediction algorithms, the sequence S, interpreted as a trial sequence, would lead to the $\ell+1$ weight vectors $\mathbf{w}_1, ..., \mathbf{w}_{\ell+1}$, and each of these defines a hypothesis that maps the instance $\mathbf{x} \in \mathbf{R}^N$ to $\mathbf{w}_t \cdot \mathbf{x}$. More generally, we define the tth hypothesis h_t of an on-line prediction algorithm A on the example sequence S to be the mapping that maps an instance $\mathbf{x}$ to the prediction $\hat{y}$ the algorithm would give if it were given $\mathbf{x}$ as the tth instance after the instance-outcome pairs $e_1, ..., e_{t-1}$ of previous trials. We denote this hypothesis h_t by $A(e_1, ..., e_{t-1})$. For $t=1$, this is the initial hypothesis of A.

In obtaining a good hypothesis from an on-line prediction algorithm, it might seem a good strategy to give the whole example sequence to the algorithm and then pick the last hypothesis, which is based on all the examples. Since additional information should only help the learner, one might think that the expected instantaneous loss for the hypothesis h_t is lower than that for the hypothesis h_{t-1}, or in general for any earlier hypothesis. This could be formalized in the inequality

$$\begin{aligned}&\mathrm{E}_{(e_1, ..., e_t) \sim D^t}(\mathrm{InstLoss}_L(A(e_1, ..., e_t), D)) \\ &\quad \leqslant \mathrm{E}_{(e_1, ..., e_{t-1}) \sim D^{t-1}}(\mathrm{InstLoss}_L(A(e_1, ..., e_{t-1}), D)).\end{aligned}$$

However, this inequality does not necessarily hold for our algorithms. As a trivial counterexample, assume that there is a unique weight vector $\mathbf{u}$ for which the expected loss $\mathrm{E}_{(\mathbf{x}, y) \sim D}(L(\mathbf{u} \cdot \mathbf{x}, y))$ is minimized. If the start vector of the algorithm is equal to $\mathbf{u}$ and the learning rate is positive, then the expected loss of the second hypothesis $\mathbf{w}_2$ obviously is higher than that of the initial hypothesis $\mathbf{w}_1 = \mathbf{u}$.

We conclude this section by presenting a simple method (Helmbold and Warmuth, 1995) that can be used for proving expected instantaneous loss bounds for all algorithms and distributions. We can rewrite the expected total loss as

$$\begin{aligned}\mathrm{E}_{S \sim D^\ell}(\mathrm{Loss}_L(A, S)) &= \mathrm{E}_{(e_1, ..., e_\ell) \sim D^\ell}\left(\sum_{t=1}^{\ell} \mathrm{Loss}_L(A((e_1, ..., e_{t-1})), e_t) \right) \\ &= \sum_{t=1}^{\ell} \mathrm{E}_{(e_1, ..., e_{t-1}) \sim D^{t-1}}(\mathrm{E}_{e \sim D}(\mathrm{Loss}_L(A((e_1, ..., e_{t-1})), e))) \\ &= \sum_{t=1}^{\ell} \mathrm{E}_{(e_1, ..., e_{t-1}) \sim D^{t-1}}(\mathrm{InstLoss}_L(A(e_1, ..., e_{t-1}), D))).\end{aligned}$$

Given the ℓ hypotheses $h_1, ..., h_\ell$, we define the randomized hypothesis h_{R} as follows. Give an instance $\mathbf{x}$, we first choose an index t from the uniform distribution on $\{1, ..., \ell\}$. We then let the prediction $h_{\mathrm{R}}(\mathbf{x})$ of the hypothesis be the prediction $h_t(\mathbf{x})$

of the tth hypothesis. From the definition of h_R and the preceding equality we obtain

$$\mathrm{E}_{(e_1, \ldots, e_{\ell-1}) \sim D^{\ell-1}}(\mathrm{InstLoss}_L(h_R, D)) = \frac{1}{\ell}\, \mathrm{E}_{S \sim D^{\ell}}(\mathrm{Loss}_L(A, S)). \tag{8.3}$$

In this paper, the hypotheses are represented by N-dimensional weight vectors. Let h_A denote the average hypothesis represented by the average weight vector $\mathbf{w}_A = \sum_{t=1}^{\ell} \mathbf{w}_t/\ell$. Assume now that for all fixed $\mathbf{x}$ and y, the value $L(\mathbf{w} \cdot \mathbf{x}, y)$ is a convex function of $\mathbf{w}$, which is a reasonable assumption and holds for the loss functions L we are interested in. Then for any $\mathbf{x}$ and y, Jensen′s Inequality yields

$$L(h_A(\mathbf{x}), y) = L\left(\left(\frac{1}{\ell}\sum_{t=1}^{\ell} \mathbf{w}_t\right) \cdot \mathbf{x}, y\right) \leqslant \frac{1}{\ell}\sum_{t=1}^{\ell} L(\mathbf{w}_t \cdot \mathbf{x}, y).$$

By taking expectations when $(\mathbf{x}, y)$ is drawn from D we obtain $\mathrm{InstLoss}(h_A, D) \leqslant \mathrm{InstLoss}(h_R, D)$ and, hence,

$$\mathrm{E}_{(e_1, \ldots, e_{\ell-1}) \sim D^{\ell-1}}(\mathrm{InstLoss}_L(h_A, D)) \leqslant \mathrm{E}_{(e_1, \ldots, e_{\ell-1}) \sim D^{\ell-1}}(\mathrm{InstLoss}_L(h_R, D)).$$

This together with equality (8.3) can be seen as a crude method for convertingan algorithm whose expected total loss is bounded to an algorithm with bounded instantaneous loss. More sophisticated conversion methods are given by Cesa-Bianchi *et al.* (1994) and Littlestone (1989).

9. EXPERIMENTAL AND THEORETICAL COMPARISON OF THE ALGORITHMS

9.1. Comparison of the Worst-Case Upper Bounds

In this subsection we compare the worst-case upper bounds given for GD and EG$^{\pm}$ in Theorems 5.3 and 5.11. Considering the upper bounds helps us to understand the circumstances in which the algorithms could be expected to perform well or poorly. We later perform experiments with artificial data to verify that the upper bounds give us correct ideas about the actual behavior of the algorithms. The experimental setting is described in Subsection 9.2, and the experiments are described in the following subsections.

The bounds given in Theorems 5.3 and 5.11 are not directly comparable, since they are given in terms of different quantities. For both algorithms, the bound is of the form $(\sqrt{K} + C)^2$, where $K = \mathrm{Loss}(\mathbf{u}, S)$ for some vector $\mathbf{u}$ and the quantity C depends on the distance from the start vector to the target vector and the norms of the instances. For simplicity, let us replace in the following discussion the relative entropy $d_{\mathrm{re}}(\mathbf{u}'/U, \mathbf{s})$ in the bound for EG$^{\pm}$ by its upper bound $\ln 2N$. For the GD algorithm, we have $C = U_2 X_2$, where $U_2 = \|\mathbf{u}\|_2$ and $X_2 = \max\{\|x_t\|_2 \mid t = 1, \ldots, \ell\}$. For the EG$^{\pm}$ algorithm, we have $C = U_1 X_\infty \sqrt{2 \ln 2N}$, where $U_1 = \|\mathbf{u}\|_1$ and $X_\infty = \max\{\|x_t\|_\infty \mid t = 1, \ldots, \ell\}$.

Figure 4 illustrates the trade-offs between the different norms in the bounds. Recall that always $\|\mathbf{w}\|_\infty \leq \|\mathbf{w}\|_2 \leq \|\mathbf{w}\|_1$, and how tight these inequalities are depends on the vector $\mathbf{w}$. Hence, the EG$^\pm$ algorithm has the advantage over the GD algorithm on the instance side of the figure, since its loss bound includes the factor X_∞ that is less than (or in special cases equal to) the factor X_2 in the loss bound for GD. Similarly, GD has the advantage on the target side, since the factor U_2 in the bound for GD never exceeds the factor U_1 in the bound for EG$^\pm$. The additional factor $2 \ln 2N$ in the bound for EG$^\pm$ further favors GD. As the products $X_2 U_2$ and $X_\infty U_1$ are incomparable, the total effect can favor either GD or EG$^\pm$.

We first construct a situation in which the bound for EG$^\pm$ is, for a large number N of dimensions, is better. To make GD lose its advantage on the target side, we choose $\mathbf{u} = (1, 0, ..., 0)$. Then $U_1 = U_2 = 1$, and the only advantage for the bound for GD now comes from the factor $2 \ln 2N$ in the bound for EG$^\pm$. To maximize the advantage of EG$^\pm$ on the instance side, we take $x_{t,i} \in \{-1, 1\}$ for all t and i. This gives $X_\infty = 1$ and $X_2 = \sqrt{N}$, which maximizes the ratio X_2/X_∞. Hence, in the case $K = 0$, we have the upper bound $2 \ln 2N$ for the loss of EG$^\pm$ and the upper bound N for the loss of GD. A less exaggerated setting that leads to similar results is obtained by choosing the target $\mathbf{u} = (1, ..., 1, 0, ..., 0)$, with k nonzero components, for some small k. Then $U_1 = k$ and $U_2 = \sqrt{k}$. Taking again $\mathbf{x}_t \in \{-1, 1\}^N$, we get in the noise free case the bound $2k^2 \ln 2N$ for EG$^\pm$ and the bound kN for GD. Thus, if the number $N - k$ of irrelevant variables is large, then EG$^\pm$ has the advantage over GD. More generally, if a large part of the total weight $\|\mathbf{u}\|_1$ is concentrated on few components of $\mathbf{u}$, then U_1 is reasonably close to U_2, and GD has only a small advantage on the target side.

In Subsection 9.3, we perform simple experiments in situations such as just described. We see that on artificial random data, the actual losses of the algorithms compare to each other as we could predict based on the analysis of worst case upper bounds. In other words, random irrelevant variables confuse the GD algorithm much more than the EG$^\pm$ algorithm. When the number k of relevant variables is kept constant, the loss of the GD algorithm grows linearly in N, whereas for the EG$^\pm$ algorithm the growth is only logarithmic.

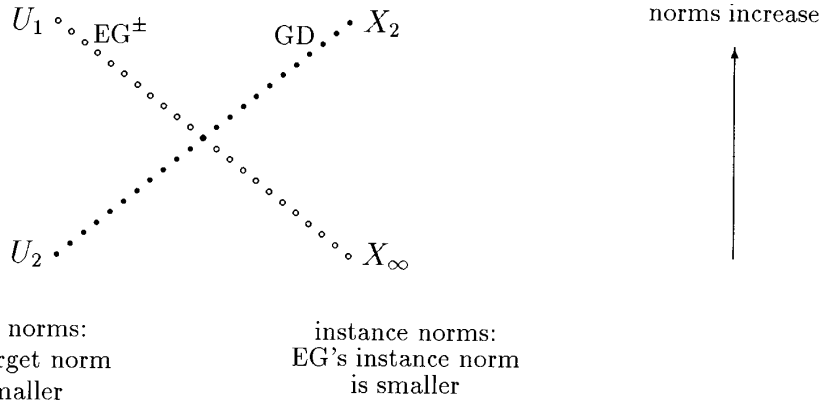

FIG. 4. Schematic representation of the main factors affecting the loss bounds of the GD and EG$^\pm$ algorithms.

It could be argued that in natural data, even irrelevant variables are usually not random. However, we propose applying our algorithm to nonlinear prediction problems by expanding the instances to include the values of a number of basis functions; see Subsection 9.6 for details. Via this expansion, even a small number of truly random variables generates a large number of pseudorandom variables, which also seem to confuse the GD algorithm.

We now show that the GD algorithm can be better, as well. We can make $X_2 = X_\infty$ by taking the instance vectors $\mathbf{x}_t$ to be unit vectors in the direction of the coordinate axes. Then $X_1 = X_\infty = 1$, and EG$^\pm$ has no advantage on the instance side. To make U_2 as much smaller than U_1 as possible, we choose $\mathbf{u} = (1, ..., 1)$, which minimizes the ratio $\|\mathbf{u}\|_2/\|\mathbf{u}\|_1$. Then $U_2 = \sqrt{N}$ and $U_1 = N$, so in the case $K = 0$, the upper bound for GD is N, while the upper bound for EG$^\pm$ is $2N^2 \ln 2N$. In Subsection 9.4 we study experimentally this situation and some of its less extreme variants. Again, we see that the worst-case upper bounds describe the real behavior of the algorithms reasonably well.

9.2. *The Experimental Setting*

The theoretical results in Section 5 are derived for worst-case situations, where an adversary may generate the examples. We wish to see if these theoretical results describe the behavior of the algorithms also when the examples are not chosen adversarially. For this purpose, we consider simple artificial data. First, we generate ℓ instances $\mathbf{x}_t$ by drawing each instance $\mathbf{x}_t$ independently from some probability measure in $\mathbf{R}^N$. Typical probability measures that we use include the uniform measure on the unit cube $[-1, 1]^N$, the uniform measure on the set $\{-1, 1\}^N$ of vertices of the unit cube, and the uniform measure on the unit sphere $\{\mathbf{x} \mid \|\mathbf{x}\|_2 = 1\}$. We choose a target $\mathbf{u} \in \mathbf{R}^N$ to suit the particular experiment we wish to perform. To generate the actual outcomes, we take the values $\mathbf{u} \cdot \mathbf{x}_t$ predicted by the weight vector $\mathbf{u}$ and add random noise to them. We quantify the amount of noise by a *noise rate* γ, which roughly gives the error $|\mathbf{u} \cdot \mathbf{x}_t - y_t|$ as a fraction of $\mathbf{u} \cdot \mathbf{x}_t$. Thus, let $C = \max\{|\mathbf{u} \cdot \mathbf{x}_t| \mid t = 1, ..., \ell\}$ be a scaling factor that gives the range of the predictions $\mathbf{u} \cdot \mathbf{x}_t$. The tth outcome y_t is chosen uniformly from $[\mathbf{u} \cdot \mathbf{x}_t - \gamma C, \mathbf{u} \cdot \mathbf{x}_t + \gamma C]$. We then run the algorithms on the example sequence and plot the cumulative losses $\sum_{t=1}^{m} (y_t - \hat{y}_t)^2$, for $m = 1, ..., \ell$, for the various algorithms.

Recall that the algorithms GD, GP, and EG$^\pm$ have their variants GDV, GPV, and EGV$^\pm$. As suggested by Theorems 5.5, 5.7, and 5.13, we use the variable learning rate algorithms in the noise-free case $\gamma = 0$ if the norms of the instances $\mathbf{x}_t$ are not same for all t. (Of course, if, say, $\|\mathbf{x}_t\|_2 = X$ for all t, there is no difference between GD and GDV, so we just use GD.)

Our experiments are all on artificial data. However, we use these experiments in an unusual way. We do not merely compare the actual performances of some algorithms A and B on particular artificial data. We also compare the actual losses of the algorithms to their worst-case upper bounds. In the cases when the loss bound of B is much larger than the loss bound of A, we typically see that already the actual loss of B is much larger than the loss bound of A. We do not need the experiments to show A performs well, as this is taken care of by proving a

worst-case loss bound for A. The point is to show that already on a simple artificial data, the competing algorithm B exceeds the worst-case bound of A. The worst-case bounds depend only on the distance of the start vector $\mathbf{s}$ to the target vector $\mathbf{u}$ as measured by some distance measure, and the total loss of the target vector $\mathbf{u}$. They are not based on assumptions about the distributions of instances or the noise mechanism. If we consider other data, with a different instance distribution and noise process but the same target and same total loss of the target, then the loss A will always stay below its upper bound. However, the loss of B might become low as well.

We represent our experimental results by showing the cumulative loss curves of some typical experiments. In other words, we plot the cumulative loss $\sum_{i=1}^{t} L(y_i, \hat{y}_i)$ of an algorithm up to trial t as a function of t. It should be noted that the actual numerical values of the cumulative losses of the algorithms are not important for us. The experiments are meant to demonstrate that by changing the target and the instances, we can make the differences in the losses of the various algorithms arbitrarily large in either direction.

Recall that we discuss two forms of upper bounds in this paper. We need very little information for bounds of the form (5.4). However, for the more sophisticated upper bounds of the form (5.3), as well as for the learning rates to be used by the algorithms so as to achieve these bounds, we need a number of parameters such as U, X, K, and D. The parameter U bounds the norm of the target vector, X bounds the norm of the instances, K bounds the loss of the target $\mathbf{u}$, and D bounds the distance $d(\mathbf{u}, \mathbf{s})$ from the start vector $\mathbf{s}$ to the target $\mathbf{u}$. In practice, these quantities are usually not known, and some other methods must be used to obtain a good learning rate. If only one of the parameters is unknown, there are strategies for guessing its value with increasing accuracy (Cesa-Bianchi *et al.*, 1994; Cesa-Bianchi *et al.*, 1996). These strategies sometimes lead to loss bounds of the form (5.3), but with the coefficient c_1 and c_2 somewhat larger than the ones obtained in Theorems 5.3 and 5.11 when good values of the parameters are known. In our experiments, we have used our knowledge of the target to set all parameters optimally and tune the learning rate as a function of the optimal choices. This is because we did not want the difficulties of choosing the learning rates hinder a fair comparison of the algorithms. It turns out that the learning rates given in the theorems are in our experiments reasonably close to the best possible ones.

In applying a learning algorithm, one is usually not so much concerned with the cumulative loss as with the quality of the final hypothesis. In our experimental setting, one would wish the hypotheses $\mathbf{w}_t$ of the algorithm to converge to the target vector $\mathbf{u}$. As shown in Section 8, bounds for the rate of convergence can be obtained from the worst-case total loss bounds. Further, in the experiments we have performed we have noticed that the algorithm with the smaller cumulative loss usually also converges faster. However, the methods we have used, in particular in choosing the learning rates in the various algorithms, have not been designed with convergence in mind. Consequently, it is possible that another approach would result in different algorithms with better convergence. In particular, one might wish to initially use a high learning rate in order to quickly get close to the target, and then decrease the learning rate in order to decrease the oscillations around the target

which are caused by noise. One can also improve convergence by averaging several hypothesis. These considerations are beyond the scope of this paper.

9.3. *Sparse Target, Instances from the Unit Cube*

We consider some situations in which the analysis of Subsection 9.1 suggests that EG$^{\pm}$ would be better. Figure 5 shows the cumulative losses for the GD and EG$^{\pm}$ algorithms in a typical experiment with a sparse target and instances from $\{-1, 1\}^N$. The number N of dimensions is 100, and the target has been chosen as $\mathbf{u} = (-1, 1, -1, 0, 0, ..., 0)$. The instances $\mathbf{x}_t$ have been chosen uniformly from $\{-1, 1\}^N$. Hence, we have $X_2 = \sqrt{100}$, $X_\infty = 1$, $U_2 = \sqrt{3} \approx 1.7321$, and $U_1 = 3$. The start vectors for GD is the all zero vector. For EG$^{\pm}$ we set the parameter U to U_1 and all components of both the start vectors to $1/(2N)$, which effectively starts EG$^{\pm}$ with the all zero vector as well. Then $d_{\mathrm{re}}(\mathbf{u}/U_1, \mathbf{s}) = \ln(200/3) \approx 4.1997$. The noise rate has been set to 0, so the upper bounds obtained from Theorems 5.3 and 5.11 become $10^2 \cdot 3 = 300$ for GD and $2 \cdot 3^2 \ln(100/3) \approx 75.5947$ for EG$^{\pm}$. The figure shows the actual cumulative losses for this experiment and their respective upper bounds. In the special case with no noise, and hence the loss of the target being 0, the learning rates suggested in Theorems 5.3 and 5.11, and used in this experiment, depend only on the instances and not on the target or the outcomes.

From Fig. 5, we see that for both algorithms, the upper bound is reasonably tight. In experiments we have observed that typically the cumulative loss of GD approaches its upper bound when the length of the trial sequence increases. The actual loss of GD is clearly higher than the worst-case loss bound of EG$^{\pm}$. The loss

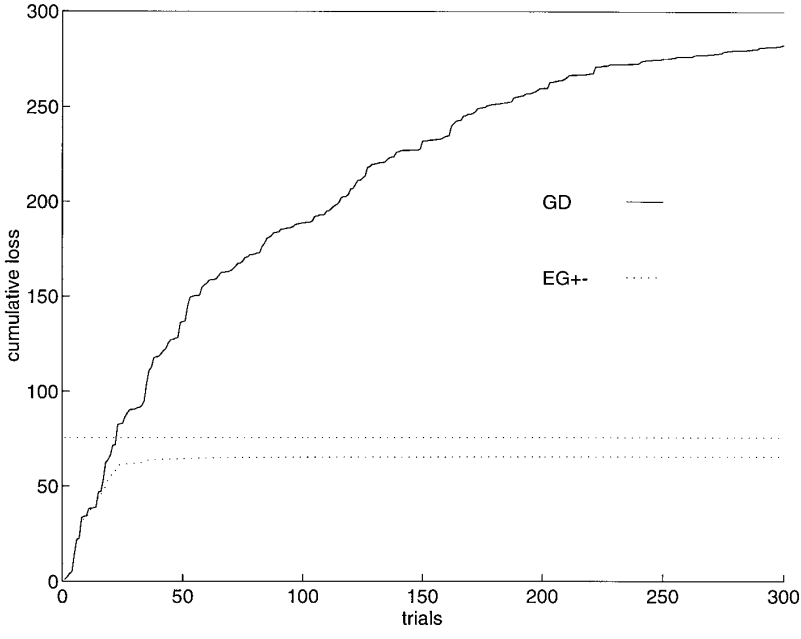

FIG. 5. Cumulative losses of GD (solid line) and EG$^{\pm}$ (dotted line), with their upper bounds, for instances $\mathbf{x}_t \in \{-1, 1\}^{100}$ and target $\mathbf{u} = (-1, 1, -1, 0, ..., 0)$.

curve of EG$^{\pm}$, and later that of GD, turns horizontal, as the hypothesis of the algorithm converges to the target and there is no more loss.

Figure 6 shows the results of an experiment similar to that of Fig. 5, except that now there is a moderate amount of noise. The noise rate γ has been set to 0.2. Now the knowledge of the distance between the start vector and the target, and the total loss of the target vector on the 300 examples, have been used in calculating both the learning rates and the worst-case upper bounds for the loss. We notice that the worst-case upper bounds are less tight, but the performances of the algorithms compared to each other are similar to those observed in the noise free case. Due to the presence of noise, the loss curves do not turn horizontal but approach a positive constant slope.

The following experiment illustrates the behavior of the GD algorithm when the instances are orthogonal. A square matrix with all its components from $\{-1, 1\}$ is a Hadamard matrix if its rows are orthogonal. We take the instance $\mathbf{x}_t$ to be the $(((t-1) \bmod N)+1)$st row of an N by N Hadamard matrix, for $N=256$. We take $\mathbf{u}=(1, 0, ..., 0)$ as the target, and set the noise rate to 0. The cumulative losses of the algorithms are shown in Fig. 7.

In the special case that the instances $\mathbf{x}_1, ..., \mathbf{x}_N$ are orthogonal, and there is no noise, the weight vector $\mathbf{w}_{t+1}$ of the GDV algorithm with the learning rate $\eta=1/2$ is the least squares solution to the (possibly underdetermined) system of equations $\mathbf{w} \cdot \mathbf{x}_j = y_j$, $j=1, ..., t$. That is, it is the solution with the least L_2 norm. Hence, applying linear least squares prediction in an on-line manner in this situation results in the same large loss as shown for GD in Fig. 7. More generally, it can be

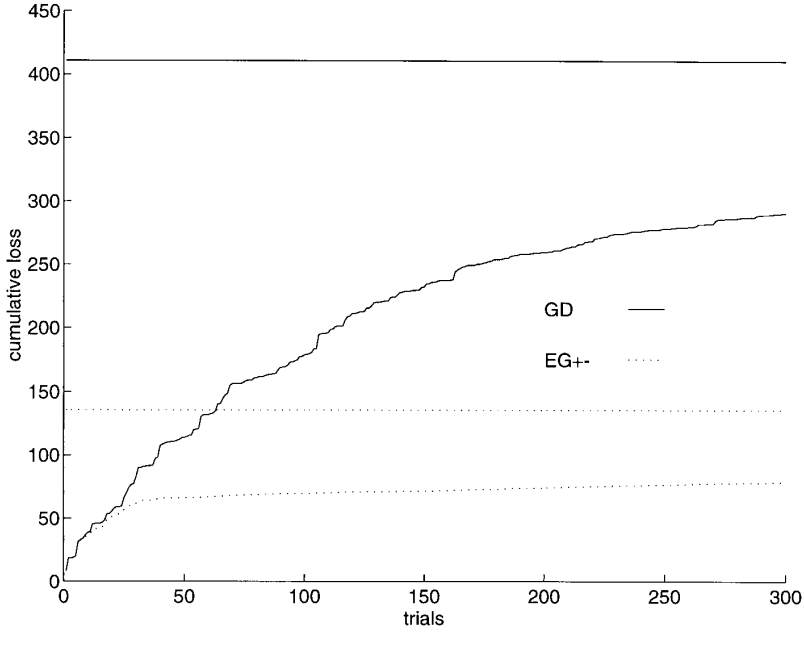

FIG. 6. Cumulative losses of GD (solid line) and EG$^{\pm}$ (dotted line), with their upper bounds, for instances $\mathbf{x}_t \in \{-1, 1\}^{100}$ and target $\mathbf{u}=(-1, 1, -1, 0, ..., 0)$, and noise rate 0.2.

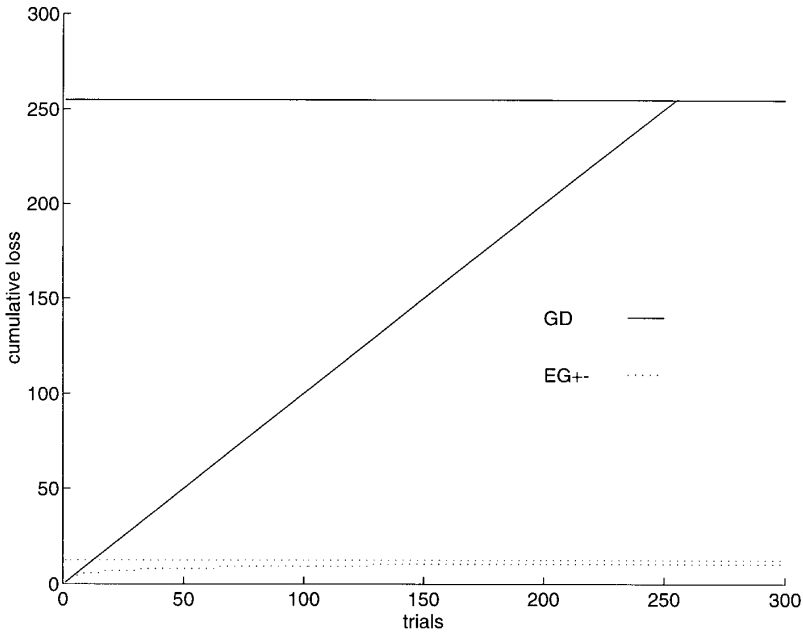

FIG. 7. Cumulative losses of GD (solid line) and EG$^{\pm}$ (dotted line), with their upper bounds, with $\mathbf{u} = (1, 0, ..., 0)$ as the target and with rows of a 256×256 Hadamard matrix as instances.

shown that no algorithm that uses weight vectors of the form $\mathbf{w}_{t+1} = \sum_{i=1}^{t} a_i \mathbf{x}_i$ can have smaller loss in this situation (Littlestone *et al.*, 1995). This class of algorithms also includes a basic variant of weight decay, where an additional $\|\mathbf{w}_t\|_2^2$ error term is used as a penalty for large weights (Hinton, 1986).

According to a commonly accepted heuristic, the number of examples needed to learn linear functions is roughly proportional to the number of dimensions in the instances. The results presented here do not contradict this in any way. The number of examples required for the EG$^{\pm}$ algorithm to learn is much smaller than the number of dimensions, but this is because the target functions have only a few nonzero components. Since the GD algorithm cannot take advantage of this, it is outperformed by EG$^{\pm}$. In the experiments of Figs. 5 and 6, there were only three relevant ones among the 100 input variables. If the number of relevant components is increased, keeping the values of all the relevant weights equal, the losses of GD and EG$^{\pm}$ first approach each other. When there are about 25 relevant components, the losses of the algorithms are roughly the same. If the number of relevant components is increased above 25, the GD algorithm outperforms the EG$^{\pm}$ algorithm more and more clearly. Based on the forms of the loss bounds for GD and EG$^{\pm}$, we expect EG$^{\pm}$ to perform well even if most of the components of the target are not zero, but the weight is concentrated on a few components, and $\|\mathbf{u}\|_1$ is thus not much larger than $\|\mathbf{u}\|_2$.

9.4. Dense Target, Instances from the Unit Sphere

We now consider a case where the target $\mathbf{u}$ is dense, in the sense that every component of an instance $\mathbf{x}$ affects the value $\mathbf{u} \cdot \mathbf{x}$. For N dimensions, we choose the

target $\mathbf{u} = (1, ..., 1)$. We choose the instances $\mathbf{x}_t$ uniformly from the N-dimensional unit sphere $\{\mathbf{x} \in \mathbf{R}^N \mid \|\mathbf{x}\|_2 = 1\}$. Then $\|\mathbf{x}_t\|_2 = 1$ for all t.

In Fig. 8, the cumulative losses of GD and EGV$^{\pm}$, with the respective upper bounds, have been plotted for $N = 20$ and noise rate 0. The GD algorithm clearly outperforms the EGV$^{\pm}$ algorithm, as we would expect from the discussion in Subsection 9.1.

To make the difference between the GD and EG$^{\pm}$ algorithms as clear as possible, we again consider nonrandom data. We choose the instances by going in order through the rows of an $N \times N$ unit matrix, for $N = 20$. Hence, the instance $\mathbf{x}_t$ has $x_{t,i} = 1$ if $i = (t-1) \bmod N + 1$, and $x_{t,i} = 0$ otherwise. As the target we use $\mathbf{u} = (1, ..., 1)$, and the noise rate is 0. The results are depicted in Fig. 9. The GD algorithm learns the correct weight u_i at trial i, and achieves perfect performance after trial N.

9.5. *Variants of the Algorithms*

Figure 10 shows the cumulative losses for the GD algorithm in the experiment of Fig. 6 with slightly differing learning rates. We see that the algorithm is robust with respect to small deviations in the learning rate, and that the learning rate obtained from Theorem 5.3 is close to optimal.

If the algorithm is given additional information about the target vector $\mathbf{u}$, its performance should improve. In Section 3 we considered in particular the restriction $\sum_i u_i = 1$. As discussed in Section 3, incorporating this restriction into the GD algorithm leads to the GP algorithm. (Here we do not restrict the weights of GP to be

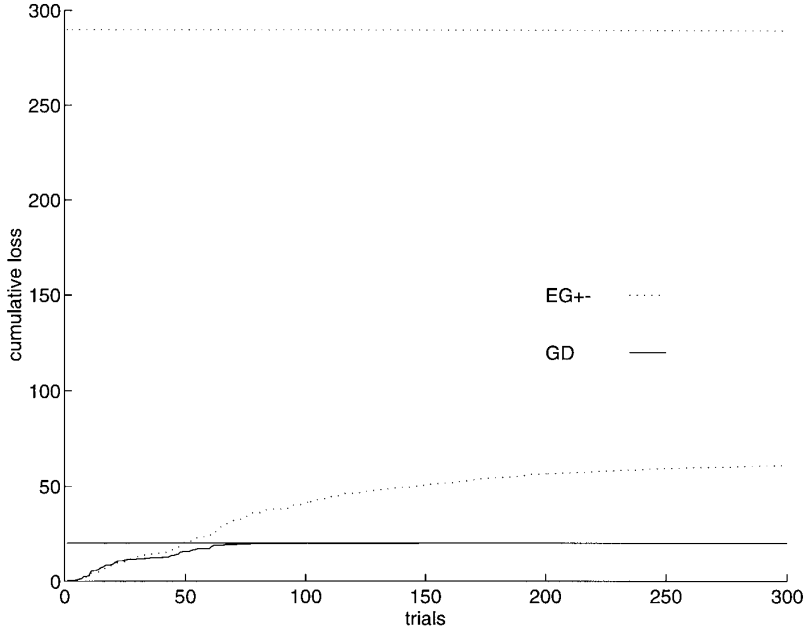

FIG. 8. Cumulative losses of GD and EGV$^{\pm}$, with their upper bounds, for target $\mathbf{u} = (1, ..., 1)$ and instances from the 20-dimensional unit sphere.

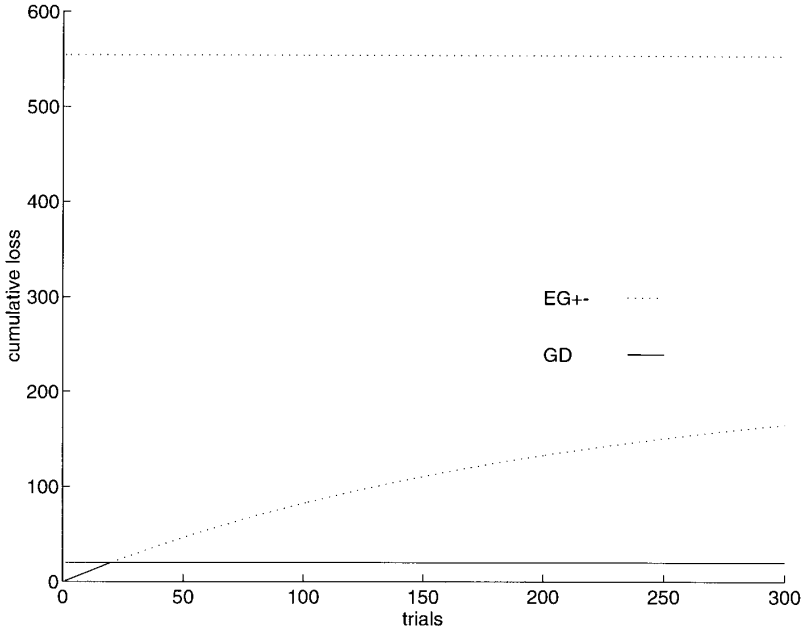

FIG. 9. Cumulative losses of GD and EG$^{\pm}$, with their upper bounds, for $\mathbf{u} = (1, ..., 1)$ as the target and with rows of the 20×20 unit matrix as instances.

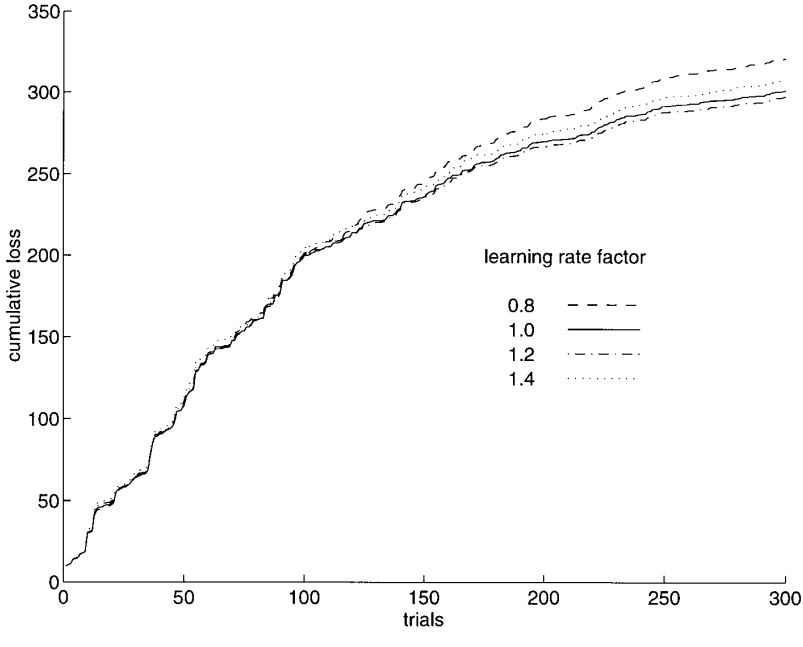

FIG. 10. Cumulative loss of GD with learning rate multiplied by 0.8, 1.0, 1.2, and 1.4.

nonnegative.) By Corollary 5.6, for any vector $\mathbf{u}$ such that $\sum_i u_i = 1$, the cumulative loss of GP is bounded by $(\sqrt{K} + U_2 V)^2$, where $K = \text{Loss}_L(\mathbf{u}, S)$, $U_2 = \|\mathbf{u}\|_2$, and $V = \max_t \|\mathbf{x}_t - \mathbf{avg}(\mathbf{x}_t)\|_2$. If the values $x_{t\,i}$ are large but, for each t, close to each other, then V can be much lower than $X_2 = \max_t \|\mathbf{x}_t\|_2$. Then we would expect GP perform better than GD does. We also have the EG algorithm, which can be thought of as EG$^{\pm}$ applied to the special case $\sum_i u_i = 1$ and $u_i \geqslant 0$ for all i. For EG we have by Theorem 5.10 the bound $(\sqrt{K} + U_1 R \sqrt{(\ln N)/2})^2$, where $R = \max_t(\max_i x_{t,i} - \min_i x_{t,i})$. Again, if the values $x_{t,i}$ are large but concentrated for each t, then EG is favored over EG$^{\pm}$.

Figure 11 shows the results of an experiment with large, concentrated values of $x_{t,i}$. There are 20 variables that attain values in $\{4, 6\}$, and the target vector has 3 non-zero components. There is no noise. The algorithms GPV and EG, which make use of the fact that $\sum_i u_i = 1$, clearly outperform the algorithms GDV and EG$^{\pm}$, which do not make use of this fact. If data like this were to appear in practice, one might want to subtract a constant from the input variables to avoid problems with large values. However, unless the value $\sum_i u_i$ is known, one cannot know how the outcomes should be transformed in order to maintain their linear relation to the instances.

We have also made experiments with the approximated EG algorithm introduced in Section 3. The advantage of the approximate algorithm is that it needs only addition and multiplications, and no exponentiation, in its update, and hence is computationally simpler. In situations considered here, the approximated and exact EG algorithms seem to have roughly the same learning performance. The learning rates suggested by the analysis of the EG and EG$^{\pm}$ algorithms also seem to work

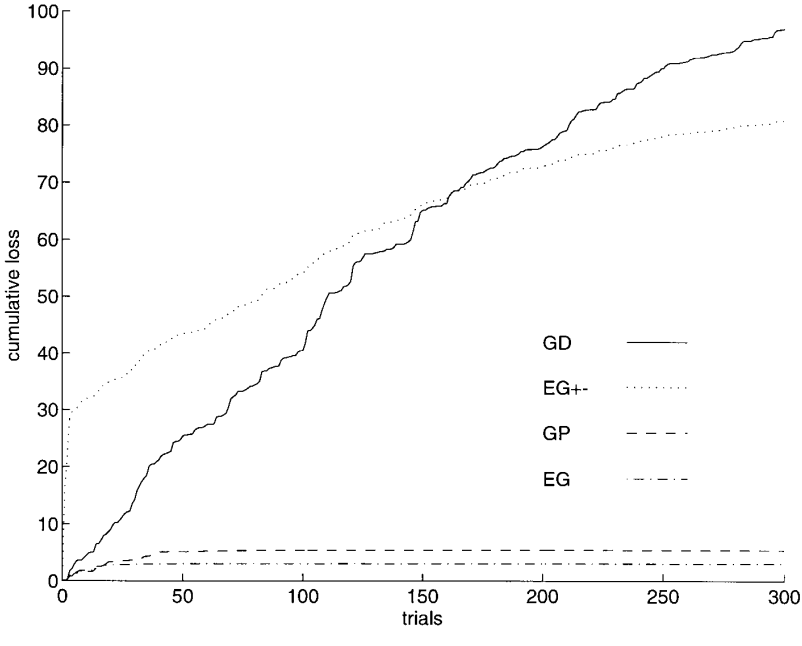

FIG. 11. Cumulative losses of GDV, EG$^{\pm}$, GPV, and EG for instances $\mathbf{x}_t \in \{4, 6\}^{20}$ and target $\mathbf{u} = (1/3, 1/3, 1/3, 0, ..., 0)$.

well for the corresponding approximated versions. Our experiments with the approximate algorithm have not been extensive, and we do not know if under some circumstances there are likely to be problems with weights going to zero.

We also performed some very preliminary experiments with the EGU algorithm. The algorithms seemed to work also in the case where some of the input variables are negative.

9.6. *Expanding the Instances*

Our next experiment illustrates the use of linear function learning to learn nonlinear target functions by the means of expanding the instances in such a way that the target function becomes linear for the expanded inputs (see Boser *et al.*, 1992). For example, let $B(\mathbf{x}, q)$, for $q = 1, 2, 3, \ldots$, be a vector that has as its components all monomials over the variables x_i, up to degree q. Thus we have, e.g., $B((x_1, x_2, x_3), 2) = (1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2)$. Then every polynomial of degree at most q over the variables x_i can be written as $\mathbf{u} \cdot B(\mathbf{x}, q)$ for some coefficient vector $\mathbf{u}$. Accordingly, polynomials of degree at most q can be learned as linear functions by using the expanded instances $B(\mathbf{x}, q)$ instead of the original instances $\mathbf{x}$ as the input to the algorithm. If the original instances have N components, then the expanded instances have $O(N^q)$ components. However, if the target polynomial has only few terms, then the target vector $\mathbf{u}$ has only few nonzero components, and the EG$^{\pm}$ algorithm can still perform well.

Figure 12 shows the results of an experiment with expanded instances. The original instances have been chosen uniformly from $\{-1, 1\}^8$, and an expanded

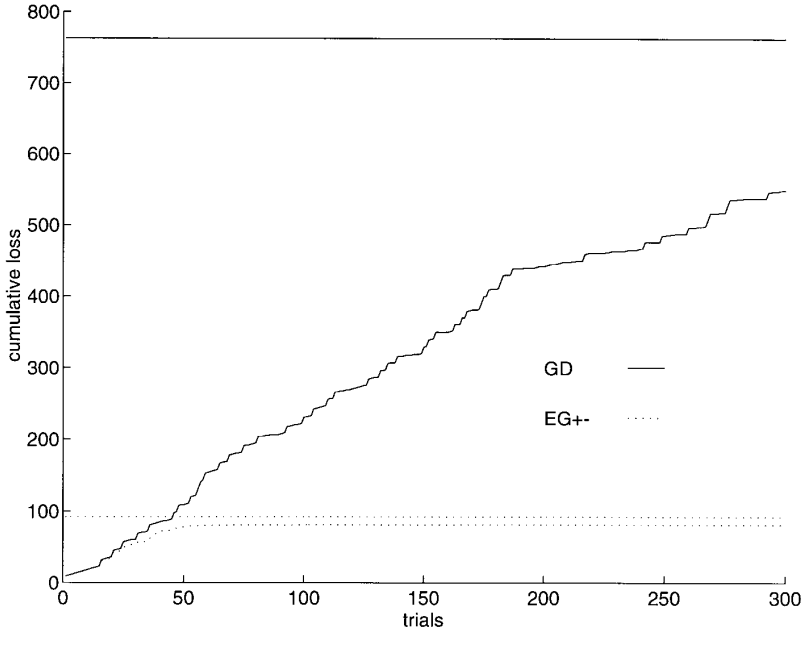

FIG. 12. Cumulative losses of GD and EG$^{\pm}$, with their upper bounds, for sparse target and expanded instances.

instance consist of the products of the components of the original instance. Since the components $x_{t,i}$ are from $\{-1, 1\}$, we do not consider products that include the same variable more than once. Hence, there are 256 products. We have chosen the target polynomial $x_2x_3x_4 + x_2x_2x_3x_6 + x_1x_2x_3x_4x_6x_7x_8$, with three terms, which for the encoding we use gives the target coefficient vector $\mathbf{u}$ with $u_{59} = u_{96} = u_{251} = 1$ and $u_i = 0$ for $i \notin \{59, 96, 251\}$. The noise rate has been set to 0. Figure 12 is qualitatively similar to Fig. 6. Hence, in the case of a sparse target and instances from the unit cube, the advantage of EG$^{\pm}$ over GD does not depend on the input variables being independent, which was the case for Fig. 6 but not for Fig. 12.

In real-world data, there might be some truly independent variables, possibly together with other variables with correlations and more complicated dependencies. The experiment suggests that in such data, the new variables generated as the products of the few original independent variables cause the GD algorithm similar difficulties as a large number of independent random variables, although the introduced new variables are not truly independent. Hence, if the instances are expanded, results may be similar to those described in Subsection 9.3 even if the number of independent random variables in the original instances is small.

When we expand the instances, the time for predicting and computing the updated weight vector usually becomes linear in the dimension of the expanded instances. For simplicity, we consider the case $\mathbf{x}_t \in \{-1, 1\}^N$ and ignore products that contain some variable more than once. In the expansion method discussed above, the expanded instances $B(\mathbf{x}, q)$ then have $\binom{N}{q} = O(N^q)$ dimensions. Thus, the expense of the prediction and update time restricts our choice of q. However, the EG$^{\pm}$ algorithm still generalizes well when the target is sparse. If the original instances are in $\{-1, 1\}$, then the components of $B(\mathbf{x}, q)$ are also in $\{-1, 1\}$. If the target $\mathbf{u}$ over the expanded domain has exactly k components in $\{-1, 1\}$ and its remaining components are zero, then (using the notation of Subsection 9.1) we get the following norms: $U_2 = \sqrt{k}$, $U_1 = k$, $X_2 = \binom{N}{\leqslant q}^{1/2}$, and $X_\infty = 1$. In the noise-free case, this leads the total loss bounds $U_2^2 X_2^2 = k\binom{N}{\leqslant q} = O(kN^q)$ for GD and $2U_1^2 X_\infty^2 \ln(2\binom{N}{\leqslant q}) = O(k^2 q \log N)$ for EG$^{\pm}$.

Assume that our goal is to obtain a hypothesis with instantaneous loss at most ε. If we take the bounds given by the reductions in Section 8 to be fair indicators of the actual instantaneous losses of the algorithms, then an algorithm with a total loss bound T leads to an algorithm with instantaneous loss T/t after $O(t)$ examples. Thus, GD would require $O(k\binom{N}{\leqslant q}/\varepsilon)$ examples and EG$^{\pm}$ would require $O(k^2 q \log N/\varepsilon)$ examples. As seen before, when k is small the EG$^{\pm}$ algorithm has good generalization performance. However, this is only useful if the time $O(\binom{N}{\leqslant q})$ is not prohibitive.

When the number of examples is small, the prediction and update time of the GD algorithm can be significantly improved from the straightforward $O(\binom{N}{\leqslant q})$. If the start vector is also of the form $B(\mathbf{s}, q)$ then the tth weight vector $\mathbf{w}_t$ is a linear combination of $B(\mathbf{s}, q)$ and $B(\mathbf{x}_1, q)$, ..., $B(\mathbf{x}_{t-1}, q)$. Updating is done by adding a new component to the linear combination, and computing the prediction $\mathbf{w}_t \cdot \mathbf{x}_t$ amounts to computing t dot products of the form $B(\mathbf{a}, q) \cdot B(\mathbf{b}, q)$, where $\mathbf{a}, \mathbf{b} \in \mathbf{R}^N$. Using dynamic programming, such a dot product can be computed in time $O(qN)$. Thus,

for the tth example the total prediction and update time becomes $O(qNt)$ instead of $O(\binom{N}{\leq q})$. In the special case $q = N$, computing one of the dot products can be further sped up to $O(N)$ even though each N-dimensional instance is expanded to 2^N components. This is achieved by simply using the equality

$$B(\mathbf{a}, N) \cdot B(\mathbf{b}, N) = \prod_{i=1}^{N} (1 + a_i b_i).$$

Thus, if $q = N$, then the total time the tth example is $O(tN)$ instead of $O(\binom{N}{\leq q})$. We know of no way to speed up the prediction and update for the EG$^{\pm}$ algorithm.

Thus, the GD algorithm seemingly has an advantage. However, as argued above, this algorithm can require as many as $\Omega(\binom{N}{\leq q})$ examples, and only during the first few trials can the GD algorithm save time by using the above methods. For large values of t, the update and prediction time $O(tN)$ for trial t would be larger than the time $O(\binom{N}{\leq q})$ obtained by simply maintaining one weight for each of the dimensions of the expanded instances. In summary, the update and prediction times of GD can be reduced, but algorithm might use so many examples that the speed-up becomes useless.

10. CONCLUSIONS

The following are the key methods used in this paper.

1. We use worst-case bounds for the total loss for evaluating on-line learning algorithms. The bounds are expressed as a function of the loss of the best linear predictor.

2. We introduce a common framework for deriving learning algorithms based on the trade-off between the distance traveled from the current weight vector and a loss function. Different distance functions lead to radically different algorithms. This framework has been adapted recently (Helmbold *et al.*, 1996b and 1996c) to an unsupervised setting.

3. The distance function also serves in a second role as a potential function in proving worst-case loss bounds by amortized analysis (Cormen *et al.*, 1990). The bounds are first expressed as a function of the learning rate and various norms of the instances and target vectors, as well as the loss of the target vector. Good loss bounds are then obtained by carefully tuning the learning rate.

In this paper we are clearly championing the EG$^{\pm}$ algorithm derived from the relative entropy distance measure. The use of this distance measure is motivated by the *Minimum Relative Entropy Principle* of Kullback (Kapur and Kesavan, 1992; Jumarie, 1990). The resulting new algorithm EG$^{\pm}$ learns very well when the target is sparse and the components of the instances are in a small range. Such situations naturally arise if we perform nonlinear predicting by first expanding the instances to include the values of some nonlinear basis functions and then predict using linear functions of the expanded instances. Since the loss of the EG$^{\pm}$ algorithm increases

only logarithmically in the number of irrelevant input variables, it is possible to have a good generalization performance even if the number of basis functions, that is the number of dimensions in the expanded instances, significantly exceeds the number of training examples. As one possible heuristic, we suggest guessing a reasonable set of basis functions and then iteratively replacing the functions that receive a small weight, and are thus not used, with new hopefully more useful functions. Cross-validation can be used to avoid overfitting.

Even for the single linear neuron we have been able to prove worst-case loss bounds (in terms of the loss of the best linear predictor) only for the square loss. Ideally we would like to have loss bounds for other standard loss functions such as the relative entropy loss. It would also be interesting to find new distance measures that would lead to new linear prediction algorithms, for which the loss bounds depend on other pairs of dual norms than the pairs (L_1, L_∞) and (L_2, L_2), which correspond to the algorithms $EG^{\pm}$ and GD, respectively.

The bounds for GD are provably optimal. However, we still need matching lower bounds for the exponentiated gradient algorithms EG and $EG^{\pm}$. The bounds for EGU still need to be generalized to allow for negative components in the instances.

Applying gradient descent in multilayer sigmoid networks leads to the well-known back-propagation algorithm. The exponentiated gradient algorithms can similarly be generalized to obtain a new exponentiated back-propagation algorithm. As a long-term research goal, we suggest developing a whole family of algoritms derived using the relative entropy as a distance measure. Many of the tradional neural network algorithms belong to the gradient descent family of algorithms that in our framework can be derived using the squared Euclidean distance. This family includes the Perceptron algorithm for thresholded linear functions, the GD algorithm for linear functions, the standard back-propagation algorithm for multilayer sigmoid networks, and the Linear Least Squares algorithm for fitting a line to data points. The new family includes, respectively, the Winnow algorithm (Littlestone, 1988), the $EG^{\pm}$ algorithm, the exponentiated back-propagation algorithm, and an algorithm for fitting a line to data points so that the relative entropy of the coefficient vector is minimized. The new family uses a new bias, which favors sparse weight vectors. We have observed that in the case of linear regression, this leads to improved performance in high dimensional problems if the target weight vector is sparse. We also expect to see similar behavior in more general settings.

Recently, Helmbold *et al.* (1995a) were able to prove worst-case loss bounds for single *sigmoided* linear neurons when the tanh function is used as the sigmoid function and the loss function is the relative entropy loss. In this case, worst-case loss bounds can be obtained for the algorithms from the gradient descent and exponentiated gradient family.

APPENDIX

Proof of Lemma 5.14. Let $\mathbf{x}_t \in [0, X]^N$ be given. We first estimate the progress $d_{\text{reu}}(\mathbf{u}, \mathbf{w}_t) - d_{\text{reu}}(\mathbf{u}, \mathbf{w}_{t+1})$, when $w_{t+1,i} = w_{t,i}\beta^{x_{t,i}}$ with $\beta > 0$. We have

$$d_{\mathrm{reu}}(\mathbf{u}, \mathbf{w}_{t+1}) - d_{\mathrm{reu}}(\mathbf{u}, \mathbf{w}_t) = \sum_{i=1}^{N} (w_{t+1,i} - w_{t,i}) + \sum_{i=1}^{N} u_i \ln \frac{w_{t,i}}{w_{t+1,i}}$$
$$= \sum_{i=1}^{N} w_{t,i}(\beta^{x_{t,i}} - 1) - \sum_{i=1}^{N} u_i x_{t,i} \ln \beta.$$

By applying the bound $\alpha^z \leqslant 1 - z(1-\alpha)$, which holds for $\alpha > 0$ and $0 \leqslant z \leqslant 1$, with $z = x_{t,i}/X$ and $\alpha = \beta^X$, we obtain

$$d_{\mathrm{reu}}(\mathbf{u}, \mathbf{w}_{t+1}) - d_{\mathrm{reu}}(\mathbf{u}, \mathbf{w}_t) \leqslant \sum_{i=1}^{N} w_{t,i} \frac{x_{t,i}}{X}(\beta^X - 1) - \sum_{i=1}^{N} u_i x_{t,i} \ln \beta \qquad \text{(A.1)}$$
$$= \mathbf{w}_t \cdot \mathbf{x}_t \frac{\beta^X - 1}{X} - \mathbf{u} \cdot \mathbf{x}_t \ln \beta. \qquad \text{(A.2)}$$

By substituting $\beta = e^{2\eta(y_t - \hat{y}_t)}$ into (A.2) we see that for proving (5.26) it is sufficient to show $G(\mathbf{w}_t \cdot \mathbf{x}_t, \hat{y}_t, y_t, \mathbf{u} \cdot \mathbf{x}_t) \leqslant 0$, where (omitting the subspcript t)

$$G(q, \hat{y}, y, r) = q \frac{e^{2X\eta(y - \hat{y})} - 1}{X} - 2r\eta(y - \hat{y}) + a(y - \hat{y})^2 - b(y - r)^2.$$

Further, since the estimate $\alpha^z \leqslant 1 - z(1-\alpha)$ is tight for $z = 0$ and $z = 1$, in the case $\mathbf{x}_t = (0, X) \in \mathbf{R}^2$ it is also necessary to show $G(\mathbf{w}_t \cdot \mathbf{x}_t, \hat{y}_t, y_t, \mathbf{u} \cdot \mathbf{x}_t) \leqslant 0$.

Recall that the prediction $\hat{y}_t$ of the EGU($\mathbf{s}$, Y, η) algorithm is given by $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$ if $\mathbf{w}_t \cdot \mathbf{x}_t \leqslant Y$ holds; otherwise $\hat{y}_t = Y$. Thus, we need to prove $G(q, \hat{y}, y, r) \leqslant 0$ for $\hat{y} = q$, and for $0 \leqslant \hat{y} = Y < q$. By the assumptions of the lemma, we also have $0 \leqslant y \leqslant Y$. Clearly $G(q, \hat{y}, y, r)$ is nonincreasing in q for $\hat{y} \geqslant y$. Hence, in the case $\hat{y} = Y < q$ the condition $G(q, \hat{y}, y, r) \leqslant 0$ follows if $G(Y, Y, y, r) \leqslant 0$ holds. Thus, without loss of generality we consider only $0 \leqslant \hat{y} = q \leqslant Y$.

By differentiating $G(\hat{y}, \hat{y}, y, r)$ with respect to r we see that for fixed $\hat{y}$ and y, the value $G(\hat{y}, \hat{y}, y, r)$ is maximized when

$$r = y + \frac{\eta}{b}(\hat{y} - y) = \left(1 - \frac{\eta}{b}\right) y + \frac{\eta}{b} \hat{y}.$$

Note that for $0 \leqslant \eta \leqslant b$, this value of r is between y and $\hat{y}$. We have $G(\hat{y}, \hat{y}, y, y + \eta(\hat{y} - y)/b) = H(\hat{y}, y)$ where

$$H(\hat{y}, y) = \frac{\hat{y}}{X}(e^{2X\eta(y - \hat{y})} - 1) - 2\eta y(y - \hat{y}) + \left(a + \frac{\eta^2}{b}\right)(y - \hat{y})^2.$$

To obtain (5.26), it is now sufficient to prove that for values of a and b as in the statement of the lemma we have $H(\hat{y}, y) \leqslant 0$ for $0 \leqslant \hat{y} \leqslant Y$ and $0 \leqslant y \leqslant Y$. We first see that for $H(\hat{y}, y) \leqslant 0$ to hold it is necessary that the values a and b satisfy the conditions of the lemma. We then see that these conditions are also sufficient, which is the main part of the claim.

We have

$$\frac{\partial H(\hat{y}, y)}{\partial y} = 2\hat{y}\eta e^{2X\eta(y-\hat{y})} - 2\eta(y-\hat{y}) - 2\eta y + \left(2a + 2\frac{\eta^2}{b}\right)(y-\hat{y}),$$

so for $y = \hat{y}$ we have $H(\hat{y}, y) = \partial H(\hat{y}, y)/\partial y = 0$. Hence, a necessary condition for having $H(\hat{y}, y) \leqslant 0$ for values of y close to $\hat{y}$ is that the second derivative

$$\frac{\partial^2 H(\hat{y}, y)}{\partial y^2} = 4\hat{y}X\eta^2 e^{2X\eta(y-\hat{y})} - 4\eta + 2a + 2\frac{\eta^2}{b}$$

is nonpositive for $y = \hat{y}$. Hence, we need $Q(y) = (4Xy + 2/b)\eta^2 - 4\eta + 2a \leqslant 0$ for $0 \leqslant y \leqslant Y$. In this range, $Q(y)$ is clearly maximized for $y = Y$. By differentiating, the value of η that minimizes $Q(Y)$ is seen to be $b/(1 + 2XYb)$. For this value of η, we have $Q(Y) \leqslant 0$ if and only if $a \leqslant b/(1 + 2XYb)$.

We have now seen that $a \leqslant b/(1 + 2XYb)$ is a sufficient condition for $G(\hat{y}, \hat{y}, y, r) \leqslant 0$ in the special case that y is close to $\hat{y}$. Before proving that the condition is also sufficient in the general case, we show that it is necessary for the claim of the lemma to hold. Let $y_t = Y - \varepsilon$ for a small positive value ε. Supposing $a > b/(1 + 2XYb)$, the preceding argument shows that $H(Y, y_t) > 0$ holds. The value $r = y_t + \eta(Y - y_t)/b$ is positive. Therefore, for any nonzero instance $\mathbf{x}_t \in [0, \infty)^N$ we can find a comparison vector $\mathbf{u}$ and a weight vector $\mathbf{w}$ such that $\mathbf{u} \cdot \mathbf{x}_t = r$ and $\mathbf{w}_t \cdot \mathbf{x}_t = Y$. In this case we have, by the preceding argument, $G(\mathbf{w}_t \cdot \mathbf{x}_t, \hat{y}_t, y_t, \mathbf{u} \cdot \mathbf{x}_t) > 0$. If we choose the particular instance $\mathbf{x}_t = (0, X)$, then the bound (A.1) holds as an equality, and hence $G(\mathbf{w}_t \cdot \mathbf{x}_t, \hat{y}_t, y_t, \mathbf{u} \cdot \mathbf{x}_t) > 0$ implies that (5.26) does not hold for this trial.

We now let $\eta = a = b/(1 + 2XYb)$ and see that these choices indeed give us $H(\hat{y}, y) \leqslant 0$ for $0 \leqslant y \leqslant Y$ and $0 \leqslant \hat{y} \leqslant Y$. As explained in the preceding part of the proof, this is sufficient for proving the lemma. For the special case $\hat{y} = 0$ we then obtain $H(0, y) = -2XYb^2y^2/(1 + 2XYb)^2 \leqslant 0$. Assume now $\hat{y} > 0$. The third derivative

$$\frac{\partial^3 H(\hat{y}, y)}{\partial y^3} = 8\hat{y}X^2\eta^3 e^{2X\eta(y-\hat{y})}$$

is then strictly positive for all y. As our choices for a and η in the case $\hat{y} = y \leqslant Y$ imply

$$\frac{\partial^2 H(\hat{y}, y)}{\partial y^2} = \frac{2b}{1 + 2XYb}\left(\frac{1 + 2Xyb}{1 + 2XYb} - 1\right) \leqslant 0,$$

we must have $\partial^2 H(\hat{y}, y)/\partial y^2 \leqslant 0$ for all $y \leqslant \hat{y}$. Therefore, since $H(\hat{y}, y) = \partial H(\hat{y}, y)/\partial y = 0$ holds for $y = \hat{y}$, we have $H(\hat{y}, y) \leqslant 0$ for $y \leqslant \hat{y}$.

In the special case that $\partial^2 H(\hat{y}, y)/\partial y^2 = 0$ holds for $y = \hat{y}$, the positiveness of the third derivative implies that $H(\hat{y}, y) > 0$ holds for $y > \hat{y}$ and, in particular, $H(\hat{y}, Y) > 0$. If the second derivative $\partial^2 H(\hat{y}, y)/\partial y^2$ is strictly negative for $y = \hat{y}$,

then $H(\hat{y}, y)$ as a function of y has a local maximum at $y = \hat{y}$. The positiveness of the third derivative implies that the second derivative can attain value 0 at most once. Hence, the function cannot have another local maximum in the range $\hat{y} < y < Y$, since between these two zeroes of the derivative $\partial H(\hat{y}, y)/\partial y$ there would have to be a third, at a local minimum, and hence two zeroes of the second derivative. Therefore, $H(\hat{y}, y)$ obtains its maximum value for $\hat{y} \leqslant y \leqslant Y$ either at $y = \hat{y}$ or at $y = Y$. Thus, it remains to verify that $H(\hat{y}, Y) \leqslant 0$ holds. We have

$$\frac{\partial H(\hat{y}, Y)}{\partial \hat{y}} = \frac{e^{2X\eta(Y-\hat{y})} - 1}{X} - 2\hat{y}\eta e^{2X\eta(Y-\hat{y})} + 2\eta Y - \left(2a + 2\frac{\eta^2}{b}\right)(Y - \hat{y})$$

$$\frac{\partial^2 H(\hat{y}, Y)}{\partial \hat{y}^2} = -4\eta e^{2X\eta(Y-\hat{y})} + 4\hat{y}X\eta^2 e^{2X\eta(Y-\hat{y})} + 2a + 2\frac{\eta^2}{b}$$

$$\frac{\partial^3 H(\hat{y}, Y)}{\partial \hat{y}^3} = 4X\eta^2 e^{2X\eta(Y-\hat{y})}(3 - 2\hat{y}X\eta).$$

Thus, regardless of the choice of η and a, we get $H(\hat{y}, Y) = \partial H(\hat{y}, Y)/\partial \hat{y} = 0$ for $\hat{y} = Y$, and to prove $H(\hat{y}, Y) \leqslant 0$ for $0 \leqslant \hat{y} \leqslant Y$ it is sufficient to prove $\partial^2 H(\hat{y}, Y)/\partial \hat{y}^2 < 0$ for $0 < \hat{y} < Y$. For our particular choices of η and a, we obtain $\partial^2 H(\hat{y}, Y)/\partial \hat{y}^2 = 0$ for $\hat{y} = Y$, and

$$\frac{\partial^3 H(\hat{y}, Y)}{\partial \hat{y}^3} = 4Xb^2 \frac{(6Y - 2\hat{y})\, Xb + 3}{(1 + 2XYb)^3} \exp\left(\frac{2Xb(Y - \hat{y})}{1 + 2XYb}\right) > 0.$$

Hence, $\partial^2 H(\hat{y}, Y)/\partial \hat{y}^2 < 0$ holds for $0 \leqslant \hat{y} < Y$, and we have $H(\hat{y}, Y) \leqslant 0$ for $\hat{y} \leqslant Y$. ∎

ACKNOWLEDGMENTS

We thank Nicolò Cesa-Bianchi, David P. Helmbold, and Yoram Singer for their comments. We also thank John Denker for inspiring us to use dimension analysis for checking update rules and learning rates.

Received April 24, 1996; final manuscript received September 10, 1996

REFERENCES

Amari, S. (1994), "Information Geometry of the EM and em Algorithms for Neural Networks," Technical Report METR 94-4, Univ. of Tokyo.

Amari, S. (1995), The EM algorithm and information geometry in neural network learning, *Neural Comput.* **7**, 13–18.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992), A training algorithm for optimal margin classifiers *in* "Proceedings, 5th Annual Workshop on Computational Learning Theory," pp. 144–152, ACM Press, New York.

Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D. P., Schapire, R. E., and Warmuth, M. K. (1994), "How to Use Expert Advice," Technical Report UCSC-CRL-94-33, Univ. of California, Santa Cruz, Computer Research Laboratory. An extended abstract appeared *in* "Proceedings, 25th Annual ACM Symposium on the Theory of Computing," pp. 382–381, ACM Press, New York.

Cesa-Bianchi, N., Long, P., and Warmuth, M. K. (1996), Worst-case quadratic loss bounds for on-line prediction of linear functions by gradient descent, *IEEE Trans. Neural Networks* **7**, 604–619.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990), "Introduction to Algorithms," MIT Press, Cambridge, MA.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977), Maximum-likelihood from incomplete data via the EM algorithm, *J. Roy. Statist. Soc. Ser. B* **39**, 1–38.

Haussler, D., Kivinen, J., and Warmuth, M. K. (1994), "Tight Worst-Case loss bounds for Predicting with Expert Advice," Technical Report UCSC-CRL-94-36, Univ. California, Santa Cruz, Computer Research Laboratory. Partial results appeared *in* "EuroCOLT '93," pp. 109–120, Clarendon Press, Oxford, and *in* "EuroCOLT '95," pp. 69–83, Springer, Berlin. To appear in *IEEE Transactions on Information Theory.*

Haykin, S. (1991), "Adaptive Filter Theory," 2nd ed., Prentice–Hall, Englewood Cliffs, NJ.

Haykin, S. (1993), "Neural Networks: A Comprehensive Foundation," Macmillan, New York.

Helmbold, D. P., Kivinen, J., and Warmuth, M. K. (1996a), Worst-case loss bounds for sigmoided linear neurons, *in* "Advances in Neural Information Processing Systems 8," MIT Press, Cambridge, MA, pp. 309–315.

Helmbold, D. P., Schapire, R. E., Singer, Y., and Warmuth, M. K. (1996b), A comparison of new and old algorithms for a mixture estimation problem, *Mach. Learning*, to appear.

Helmbold, D. P., Schapire, R. E., Singer, Y., and Warmuth, M. K. (1996c), On-line portfolio selection using multiplicative updates *in* "Proceedings, 13th International Conference on Machine Learning," Morgan Kaufmann, San Francisco, pp. 243–251.

Helmbold, D. P., and Warmuth, M. K. (1995), On weak learning, *J. Comput. System Sci.* **50**, 551–573.

Hinton, G. E. (1986), Learning distributed representations of concepts, *in* "Proceedings, 8th Annual Conference of the Cognitive Science Society," pp. 1–12, Erlbaum, Hillsdale, NJ.

Jumarie, G. (1990), "Relative Information," Springer, New York.

Kapur, J. N., and Kesavan, H. K. (1992), "Entropy Optimization Principles with Applications," Academic Press, New York.

Kearns, M. J., Schapire, R. E., and Sellie, L. M. (1994), Toward efficient agnostic learning, *Mach. Learning* **17**, 115–142.

Littlestone, N. (1988), Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, *Mach. Learning* **2**, 285–318.

Littlestone, N. (1989), From on-line to batch learning *in* "Procedings, 2nd Annual Workshop on Computational Learning Theory," pp. 269–284, Morgan Kaufmann, San Mateo, CA.

Littlestone, N. (1989), "Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms," Ph.D. thesis, Technical Report UCSC-CRL-89-11, Univ. of California, Santa Cruz, Computer Research Laboratory.

Littlestone, N. (1991), Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow, *in* "Proceedings, 4th Annual Workshop on Computational Learning Theory," pp. 147–156, Morgan Kaufmann, San Mateo, CA.

Littlestone, N., Long, P. M., and Warmuth, M. K. (1995), On-line learning of linear functions, *J. Comput. Complexity* **5**, 1–23.

Littlestone, N., and Warmuth, M. K. (1994), The weighted majority algorithm, *Inform. and Comput.* **108**, 212–261.

Luenberger, D. G. (1984), "Linear and Nonlinear Programming," Addison–Wesley, Reading, MA.

Royden, H. (1963), "Real Analysis," Macmillan, New York.

Schapire, R. E., and Warmuth, M. K. (1994), On the worst-case analysis of temporal-difference learing algorithms, *in* "Proceedings, 11th International Conference on Machine Learning," pp. 266–274, Morgan Kaufmann, San Francisco; *Mach. Learning*, to appear.

Vovk, V. (1990), Aggregating strategies, *in* "Proceedings, 3rd Annual Workshop on Computational Learning Theory," pp. 371–383, Morgan Kaufmann, San Mateo, CA.

Widrow, B., and Stearns, S. (1985), "Adaptive Signal Processing," Prentice–Hall, Englewood Cliffs, NJ.