Relative Loss Bounds for Single Neurons

David P. Helmbold, Jyrki Kivinen, and Manfred K. Warmuth

***Abstract*— We analyze and compare the well-known gradient descent algorithm and the more recent exponentiated gradient algorithm for training a single neuron with an arbitrary transfer function. Both algorithms are easily generalized to larger neural networks, and the generalization of gradient descent is the standard backpropagation algorithm. In this paper we prove worst-case loss bounds for both algorithms in the single neuron case. Since local minima make it difficult to prove worst-case bounds for gradient-based algorithms, we must use a loss function that prevents the formation of spurious local minima. We define such a matching loss function for any strictly increasing differentiable transfer function and prove worst-case loss bounds for any such transfer function and its corresponding matching loss. For example, the matching loss for the identity function is the square loss and the matching loss for the logistic transfer function is the entropic loss. The different forms of the two algorithms' bounds indicates that exponentiated gradient outperforms gradient descent when the inputs contain a large number of irrelevant components. Simulations on synthetic data confirm these analytical results.**

***Index Terms*— Exponentiated gradient algorithm, generalized linear regression, matching loss function, relative loss bounds, sigmoid transfer function.**

I. Introduction

The basic element of a neural network, a *neuron*, takes in a number of real-valued input variables and produces a real-valued output. The input–output mapping of a neuron is defined by a *weight vector* $\boldsymbol{w} \in \boldsymbol{R}^N$, where N is the number of input variables, and a *transfer function* ϕ. When presented with the input vector $\boldsymbol{x} \in \boldsymbol{R}^N$, the neuron produces the output $\hat{y} = \phi(\boldsymbol{w} \cdot \boldsymbol{x})$, where $\boldsymbol{w} \cdot \boldsymbol{x} = \Sigma_{i=1}^N w_i x_i$. Thus, the weight vector regulates the influence of each input variable on the output, and the transfer function can produce nonlinearities in the input–output mapping. In particular, when the transfer function is the commonly used logistic function, $\phi(z) = 1/(1+e^{-z})$, the outputs are bounded between zero and one. On the other hand, if the outputs should be unbounded, then it is often convenient to use the identity function as the transfer function, in which case the neuron simply computes a linear mapping. In this paper we allow the transfer function to be any strictly increasing continuous function, a class that includes both the logistic function and the identity function, but not discontinuous (e.g., step) functions.

The goal of *learning* is to discover a weight vector $\boldsymbol{w}$ that produces a desirable input–output mapping. This is achieved by considering a sequence $S = ((\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_\ell, y_\ell))$ of *examples*, where for $t = 1, \cdots, \ell$ the value $y_t \in \boldsymbol{R}$ is the desired output for the input vector $\boldsymbol{x}_t$, possibly distorted by noise or other errors. We call $\boldsymbol{x}_t$ the tth *instance* and y_t the tth *outcome*. In what is often called batch learning, all ℓ examples are given at once and are available during the whole training session. As noise and other problems often make it impossible (or undesirable) to find a weight vector $\boldsymbol{w}$ satisfying $\phi(\boldsymbol{w} \cdot \boldsymbol{x}_t) = y_t$ for all t, one often introduces a *loss function* L, such as the *square loss* given by $L(y, \hat{y}) = (y - \hat{y})^2/2$, and finds a weight vector $\boldsymbol{w}$ that minimizes the empirical loss (or training error)

$$\mathrm{Loss}(\boldsymbol{w}, S) = \sum_{t=1}^{\ell} L(y_t, \phi(\boldsymbol{w} \cdot \boldsymbol{x}_t)). \tag{1}$$

With the square loss and identity transfer function $\phi(z) = z$, this is the well-known linear regression problem. When ϕ is the logistic function and L is the *entropic loss* given by $L(y, \hat{y}) = y \ln(y/\hat{y}) + (1-y)\ \ln((1-y)/(1-\hat{y}))$, this can be seen as a special case of logistic regression. (With the entropic loss, we assume $0 \le y_t, \hat{y}_t \le 1$ for all t, and use the convention $0\ \ln\ 0 = 0\ \ln\ (0/0) = 0$.)

In this paper we use an *on-line prediction* (or life-long learning) approach to the learning problem. One of the reasons that we focus on on-line learning is that good on-line algorithms can always be converted into good batch algorithms [12], [13], [17]. Instead of processing all the examples at once, the training algorithm begins with some fixed start vector $\boldsymbol{w}_1$, and produces a sequence of weight vectors, $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{\ell+1}$. Each new weight vector $\boldsymbol{w}_{t+1}$ is obtained by applying a simple *update rule* to the previous weight vector $\boldsymbol{w}_t$ and the single example $(\boldsymbol{x}_t, y_t)$. In the on-line prediction model, the algorithm uses its tth weight vector, or *hypothesis*, to make the *prediction* $\hat{y}_t = \phi(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)$. The training algorithm is then charged a loss $L(y_t, \hat{y}_t)$ for this tth *trial*. The performance of a training algorithm A that produces the weight vectors $\boldsymbol{w}_t$ on an example sequence S is measured by its total (cumulative) loss

$$\mathrm{Loss}(A, S) = \sum_{t=1}^{\ell} L(y_t, \phi(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)). \tag{2}$$

Our main results are bounds on the cumulative losses for two on-line prediction algorithms. One of these is the standard *gradient descent* (GD) algorithm. The other one, which we call EG$^{\pm}$ (for *exponentiated gradient*), also is based on the gradient

Manuscript received August 27, 1998; revised July 29, 1999. The work of D. P. Helmbold was supported by the NSF under Grant CCR 9700201. The work of J. Kivinen was supported by the Emil Aaltonen Foundation, the Academy of Finland, and the European Commission under Working Group NeuroCOLT2, EP27150. The work of M. K. Warmuth was supported by ONR under Grant NO0014-91-J-1162 and the NSF under Grant CCR 9700201.

D. P. Hembold and M. K. Warmuth are with the Department of Computer Science, University of California, Santa Cruz, Santa Cruz, CA 95064 USA.

J. Kivinen is with the Department of Computer Science, University of Helsinki, FIN-00014 Helsinki, Finland.

Publisher Item Identifier S 1045-9227(99)08834-7.

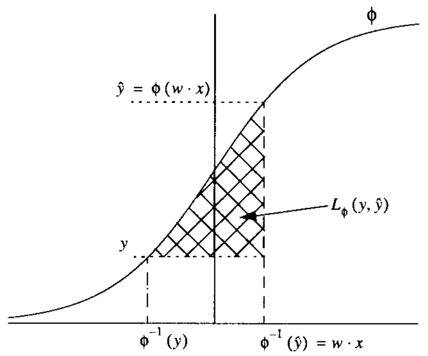


Fig. 1. The matching loss given in (3).

but uses it in a different manner than GD. The bounds are derived in a worst-case setting: we make no assumptions about how the instances are distributed or the relationship between each instance $\boldsymbol{x}_t$ and its corresponding outcome y_t. The approach we take is to compare the total losses, Loss(GD, S) and Loss(EG$^\pm$, S), to the least achievable empirical loss, $\inf_{\boldsymbol{w}}$ Loss($\boldsymbol{w}$, S) where Loss($\boldsymbol{w}$, S) is as in (1). By giving the bounds in such a relative form we can give meaningful results even in the total absence of statistical assumptions about how the examples are generated. If the least achievable empirical loss is high, the dependence between the instances and outcomes in S cannot be closely approximated by any neuron using the transfer function, so it is reasonable that the losses of the algorithms are also high. More interestingly, if some weight vector achieves a low empirical loss, then these relative bounds show that the losses of the algorithms are also low. Hence, although the algorithms always predict based on an initial segment of the example sequence, they must perform almost as well as the best fixed weight vector for the whole sequence.

The choice of loss function is crucial for the results that we prove. In particular, since we are using gradient-based algorithms, the empirical loss should not have spurious local minima. This can be achieved by using *matching* loss and transfer functions. The transfer function ϕ and loss function L_ϕ *match* if and only if

$$L_\phi(y, \hat{y}) = \int_{\phi^{-1}(y)}^{\phi^{-1}(\hat{y})} (\phi(z) - y)\, dz. \tag{3}$$

The definition of matching loss and transfer functions has a geometric interpretation (see Fig. 1). If L_ϕ and ϕ are matching loss and transfer functions then for $y < \hat{y}$ the value $L_\phi(y, \hat{y})$ is the area in the (p, q) plane below the curve $q = \phi(p)$, above the line $q = y$, and to the left of the line $p = \phi^{-1}(\hat{y})$. For example, if the transfer function is the logistic function, the matching loss function is the entropic loss, and if the transfer function is the identity function, the matching loss function is the square loss. When the loss function L used in the definitions (1) and (2) is the matching loss function L_ϕ for the transfer function ϕ, we indicate this by writing Loss$_\phi$ instead of just Loss.

Any increasing and differentiable transfer function has a matching loss function over its range. In Section II we show that a loss function L has a matching transfer function essentially whenever

$$\frac{\partial L(y, \hat{y})}{\partial \hat{y}} \frac{1}{(\hat{y} - y)}$$

is independent of y. Furthermore, we will will show in Section II that for any example sequence S, the mapping from $\boldsymbol{w}$ to Loss$_\phi(\boldsymbol{w}, S)$ is convex and hence has no spurious local minima. Note that using the logistic activation function with the square loss can lead to a very large number of local minima [1], [3], [4]. Even in the batch setting there are reasons to use the entropic loss with the logistic transfer function [19].

We bound the loss of the GD and EG$^\pm$ algorithms in Section IV. How much our bounds on the losses of the two algorithms exceed the least empirical loss depends on the maximum slope of the transfer function used. More interestingly, the bounds depend on various norms of the instances and the vector $\boldsymbol{w}$ for which the least empirical loss is achieved. As one might expect, neither of the algorithms is uniformly better than the other. The new EG$^\pm$ algorithm performs well when most of the input variables are irrelevant, i.e., when some weight vector $\boldsymbol{w}$ with $w_i = 0$ for most indexes i has a low empirical loss. On the other hand, the GD algorithm is better when the weight vectors with low empirical loss have many nonzero components, but the instances contain many zero components.

The bounds we derive concern only single neurons, and one often combines a number of neurons into a multilayer feedforward neural network. In particular, applying the gradient descent algorithm in the multilayer setting gives the standard backpropagation algorithm. Also the EG$^\pm$ algorithm, being gradient-based, can easily be generalized for multilayer feedforward networks. Although it seems unlikely that our loss bounds will generalize to multilayer networks, we believe that the intuition gained from the single neuron case will provide useful insight into the relative performance of the two algorithms in the multilayer case. Furthermore, the EG$^\pm$ algorithm is less sensitive to large numbers of irrelevant attributes. Thus it might be possible to avoid multilayer networks by introducing many new inputs, each of which is a nonlinear function of the original inputs. Multilayer networks remain an interesting area for future study.

Our work follows the path opened by Littlestone [15] with his work on learning thresholded neurons with sparse weight vectors. This paper builds directly on recent results about relative (or worst-case) loss bounds for linear regression [7], [13]. There are several other recent results in this area, including the generalization of the matching loss results to multidimensional outputs [14], and applying these techniques to classification [10], [9]. These new results are related to the fact that the matching loss functions considered in this paper are the one-dimensional special case of *Bregman divergences* [2], [8]. On the other hand, also nonmatching loss functions can be used with the GD and EG algorithms [5]. Yamanishi [21], [22] has used more involved algorithms and analysis techniques to achieve on-line loss bounds in terms of general loss functions and comparing against general parametric hypotheses.

The next section gives additional examples of matching loss and transfer functions and provides sufficient conditions for a loss or transfer function to be part of a matching pair.

Section III describes and motivates the gradient descent and exponentiated gradient algorithms. Section IV contains the formal statements of our results and Section V has some supporting experiments. We conclude with a discussion of open problems in Section VI. The full proofs are given in the Appendix.

II. Matching Loss and Transfer Functions

When ϕ is continuous and strictly increasing, it has a continuous inverse ϕ^{-1}, and (3) defines a matching loss function L_ϕ. The matching loss function L_ϕ is then continuous and satisfies $L_\phi(y, y) = 0$ for all y and $L_\phi(y, \hat{y}) > 0$ for $y \neq \hat{y}$. If ϕ is not strictly increasing and thus not invertible, we of course cannot define a matching loss function L_ϕ by (3). The same is true if ϕ is strictly increasing but not continuous. In such cases we could try to define a loss by rewriting (3) as

$$L(\phi(a), \phi(\hat{a})) = \int_a^{\hat{a}} (\phi(z) - \phi(a))\, dz \tag{4}$$

but this would run into some problems. If ϕ is continuous but not strictly increasing, then there are values $\hat{a}_1 \neq \hat{a}_2$ such that $\phi(\hat{a}_1) = \phi(\hat{a}_2)$ but (4) would give $L(\phi(a), \phi(\hat{a}_1)) \neq L(\phi(a), \phi(\hat{a}_2))$. Thus, (4) does not give a well-defined loss $L(y, \hat{y})$. Further, for the case of strictly increasing but noncontinuous ϕ, consider the example $\phi(z) = z + \text{sign}(z)$ where $\text{sign}(z) = -1$ for $z < 0$, $\text{sign}(0) = 0$, and $\text{sign}(z) = 1$ for $z > 0$. Defining L by (4) now gives $L(\phi(-\epsilon), \phi(\epsilon)) \to 0$ as $\epsilon \to 0$, while the outputs $\phi(-\epsilon)$ and $\phi(\epsilon)$ remain two apart. Thus, we could have $L(y, \hat{y})$ approaching zero while $\hat{y}$ does not approach y.

We conclude that having a continuous and strictly increasing transfer function is necessary and sufficient for the matching loss defined by (3) to have the properties we find desirable. Of course, these restrictions disallow thresholded transfer functions. For threshold functions, the basic idea of (3) is still applicable, but the technicalities require a more sophisticated treatment than is presented here; see [9].

Consider now the reverse direction: when does a loss function L have a matching transfer function ϕ? Assume that $L = L_\phi$ for some ϕ, and make the additional assumption that L and ϕ are differentiable and the derivative $\phi'(z)$ is strictly positive. Then by differentiating both sides of (3) we obtain

$$\frac{\partial L(y, \hat{y})}{\partial \hat{y}} = (\hat{y} - y)\frac{d\phi^{-1}(\hat{y})}{d\hat{y}}$$

so for $y \neq \hat{y}$ we have

$$\frac{d\phi^{-1}(\hat{y})}{d\hat{y}} = \frac{\partial L(y, \hat{y})}{\partial \hat{y}} \frac{1}{(\hat{y} - y)}. \tag{5}$$

Since the left-hand side of (5) has no y-dependence, the right-hand-side must be independent of y as well. Thus, for fixed $\hat{y}$ the value

$$\frac{\partial L(y, \hat{y})}{\partial \hat{y}} \frac{1}{(\hat{y} - y)} \tag{6}$$

must be the same for all $y \neq \hat{y}$. If L satisfies this condition then we can obtain a unique matching transfer function ϕ by solving (5) (assuming that the right-hand side of (5) is positive and continuous). This characterization of loss functions for which a matching transfer function exists is not very intuitive, and we have omitted a closer inspection of various regularity conditions required. One example of a loss function where the value (6) depends on y, is the Hellinger loss defined by $L(y, \hat{y}) = ((y^{1/2} - \hat{y}^{1/2})^2 + ((1-y)^{1/2} - (1-\hat{y})^{1/2})^2)/2$. This technique shows that it does not have a differentiable matching transfer function. Table I contains several examples of matching loss and transfer functions.

TABLE I
Some Examples of Matching Loss and Transfer Functions

loss function $L_\phi(y, \hat{y})$	transfer function $\phi(z)$	bound Z on $\phi'(z)$
$(y - \hat{y})^2/2$	z	1
$y \ln \frac{y}{\hat{y}} + (1 - y) \ln \frac{1-y}{1-\hat{y}}$	$1/(1 + e^{-z})$	1/4
$\frac{1}{2}(y+1) \ln \frac{y+1}{\hat{y}+1} + \frac{1}{2}(y-1) \ln \frac{y-1}{\hat{y}-1}$	$\tanh z$	1/2
$(\hat{y} - y) \tan \hat{y} + \frac{1}{2} \ln \frac{1+\tan y}{1+\tan \hat{y}}$	$\arctan z$	1

To get more examples of matching loss functions, notice that (4) can be rewritten as $L(\phi(a), \phi(\hat{a})) = \Phi(\hat{a}) - \Phi(a) - \Phi'(a)(\hat{a} - a)$ where $\Phi' = \phi$. Thus, for any continuously differentiable and strictly convex function Φ, we can define a matching loss function for the transfer function $\phi = \Phi'$, and this transfer function is continuous and strictly increasing.

For the remainder of this paper, we assume that the transfer function ϕ is increasing and differentiable, and Z is a constant such that $0 < \phi'(z) \leq Z$ holds for all $z \in \boldsymbol{R}$. We call any such ϕ a *sigmoid* function. For the matching loss function L_ϕ defined by (3) we have

$$\frac{\partial L_\phi(y, \phi(\boldsymbol{w} \cdot \boldsymbol{x}))}{\partial w_i} = (\phi(\boldsymbol{w} \cdot \boldsymbol{x}) - y)x_i. \tag{7}$$

Treating $L_\phi(y, \phi(\boldsymbol{w} \cdot \boldsymbol{x}))$ for fixed $\boldsymbol{x}$ and y as a function of $\boldsymbol{w}$, we see that the Hessian $H(\boldsymbol{w})$ of the function is given by $H_{ij}(\boldsymbol{w}) = \phi'(\boldsymbol{w} \cdot \boldsymbol{x})x_i x_j$. Then $\boldsymbol{v}^{\mathrm{T}} H(\boldsymbol{w})\boldsymbol{v} = \phi'(\boldsymbol{w} \cdot \boldsymbol{x})(\boldsymbol{v} \cdot \boldsymbol{x})^2$, so H is positive semidefinite. Hence, for an arbitrary fixed S the total empirical loss $\text{Loss}_\phi(\boldsymbol{w}, S)$, as defined in (1), as a function of $\boldsymbol{w}$ is convex and thus has no spurious local minima.

Notice that if $\phi(a)$ is bounded for $a \in \boldsymbol{R}$, then the inverse ϕ^{-1} must have an unbounded derivative. From (5) we can then see that also the loss $L_\phi(y, \hat{y})$ will have an unbounded derivative with respect to $\hat{y}$.

III. The GD and EG Algorithms

This section describes how the GD training algorithm and the new exponentiated gradient training algorithm update the neuron's weight vector. We first describe the GD algorithm. Recall that the algorithm's prediction at trial t is $\hat{y}_t = \phi(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)$, where $\boldsymbol{w}_t$ is the current weight vector and $\boldsymbol{x}_t$ is the input vector. By (7), performing gradient descent in weight space on the loss incurred in a single trial leads to the update rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta(\hat{y}_t - y_t)\boldsymbol{x}_t. \tag{8}$$

The parameter η is a positive *learning rate*. To get the best possible performance, one must carefully choose the learning rate η. Note that the weight vector $\boldsymbol{w}_t$ of GD always satisfies $\boldsymbol{w}_t = \boldsymbol{w}_1 + \Sigma_{i=1}^{t-1} a_i \boldsymbol{x}_i$ with the scalar coefficients $a_i = -\eta(\hat{y}_t - y_t)$. Typically, one uses the zero start vector $\boldsymbol{w}_1 = \mathbf{0}$.

A more recent training algorithm, called the EG algorithm [13], uses the same gradient in a different way. This algorithm makes multiplicative (rather than additive) changes to the weight vector, and the gradient appears in the exponent. The basic version of the EG algorithm includes an additional normalization of the weight vector, so the update is given by

$$w_{t+1,i} = \frac{w_{t,i}\ \exp(-\eta(\hat{y}_t - y_t)x_{t,i})}{\sum_{j=1}^{N} w_{t,j}\ \exp(-\eta(\hat{y}_t - y_t)x_{t,j})}. \tag{9}$$

The start vector is usually chosen to be uniform, $\boldsymbol{w}_1 = (1/N, \cdots, 1/N)$. Notice that it is the logarithms of the weights produced by the EG training algorithm (rather than the weights themselves) that are essentially linear combinations of the past examples.

Because of the normalization in the update, the weight vector $\boldsymbol{w}_t$ of the EG algorithm always belongs to the $N-1$ dimensional probability simplex

$$P_{N-1} = \left\{ \boldsymbol{w} \in \boldsymbol{R}^N \left| \sum_{i=1}^{N} w_i = 1, 0 \le w_i \le 1 \text{ for all } i \right. \right\}.$$

We call such vectors *probability vectors*.

Although there might be situations in which we could expect to find a good weight vector from the probability simplex, in general it is not desirable to constrain the weight vectors to be nonnegative and sum to one. Hence, following [13] we introduce the EG algorithm with positive and negative weights EG$^{\pm}$ that simulates negative weights and relaxes the constraint that they sum to one. In addition to the learning rate η, the EG$^{\pm}$ algorithm has a *scaling factor* $U > 0$ as a parameter.

We define the behavior of EG$^{\pm}$ on a sequence of examples $S = ((\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_\ell, y_\ell))$ to be the same as the EG algorithm's behavior on a transformed example sequence $S' = ((\boldsymbol{x}'_1, y_1), \cdots, (\boldsymbol{x}'_\ell, y_\ell))$ where $\boldsymbol{x}' = (Ux_1, \cdots, Ux_N, -Ux_1, \cdots, -Ux_N)$. The EG algorithm here uses the uniform start vector $(1/(2N), \cdots, 1/(2N))$ and learning rate supplied by the EG$^{\pm}$ algorithm. At each time time t the N-dimensional weight vector $\boldsymbol{w}$ of EG$^{\pm}$ is defined in terms of the $2N$-dimensional weight vector $\boldsymbol{w}'$ of EG as

$$w_{t,i} = U(w'_{t,i} - w'_{t,N+i}).$$

Thus EG$^{\pm}$ with scaling factor U can learn any weight vector $\boldsymbol{w} \in \boldsymbol{R}^N$ with $\Sigma_i\ |w_i| < U$ by having the embedded EG algorithm learn the appropriate $2N$-dimensional (nonnegative and normalized) weight vector $\boldsymbol{w}'$. Note that for the uniform start vector $\boldsymbol{w}'_1$ of the underlying EG algorithm, the actual start vector $\boldsymbol{w}_1$ of the EG$^{\pm}$ algorithm is again the zero vector.

The key concept in understanding the motivation behind the GD and EG algorithms, and later the proofs for our relative loss bounds, is that of a *distance function* d. We require that the distance $d(\boldsymbol{u}, \boldsymbol{w})$ between two different weight vectors $\boldsymbol{u}$ and $\boldsymbol{w}$ is always strictly positive, and $d(\boldsymbol{u}, \boldsymbol{u}) = 0$ for any $\boldsymbol{u}$. The main distance functions we need here are the squared Euclidean distance $d(\boldsymbol{u}, \boldsymbol{w}) = ||\boldsymbol{u} - \boldsymbol{w}||_2^2/2$ and, for probability vectors only, the *relative entropy* (or Kulback–Leibler divergence)

$$d_{\mathrm{RE}}(\boldsymbol{u}, \boldsymbol{w}) = \sum_{i=1}^{N} u_i\ \ln\ \frac{u_i}{w_i}. \tag{10}$$

Our general method for deriving on-line update rules is to define the new weight vector by a minimization problem that involves some loss function L and distance function d. After seeing an example $(\boldsymbol{x}_t, y_t)$, the algorithm should modify its weight so that its loss would be smaller if the same example were encountered again. A distance term is used as a kind of regularizer to ensure that the updated weight vector stays close to the previous one.

Following [13] we suggest the update rule

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ (d(\boldsymbol{w}, \boldsymbol{w}_t) + \eta L(y_t, \phi(\boldsymbol{w} \cdot \boldsymbol{x}_t))) \tag{11}$$

where η is a positive parameter controlling the relative importance of the distance term versus the loss term. The above minimization is, in general, hard to solve in closed form. Thus we approximate the loss term by its first-order Taylor expansion around $\boldsymbol{w}_t$. To obtain the Taylor approximation, we apply (7) to compute the gradient of $L_\phi(y, \phi(\boldsymbol{w} \cdot \boldsymbol{x}))$ with respect to $\boldsymbol{w}$. This results in the following easier to compute update:

$$\begin{aligned} \boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ &(d(\boldsymbol{w}, \boldsymbol{w}_t) + \eta(L(y_t, \phi(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)) \\ &+ ((\phi(\boldsymbol{w}_t \cdot \boldsymbol{x}_t) - y_t)\boldsymbol{x}_t \cdot (\boldsymbol{w} - \boldsymbol{w}_t)))). \end{aligned}$$

The distance function can now be viewed as governing the step size of the gradient descent. Intuitively, the distance function estimates how rapidly the gradient of the loss changes as one moves away from $\boldsymbol{w}_t$. (See [11] and [13] for further discussion on the approximation step.) With the squared Euclidean distance this leads to the GD update (8). When d is the relative entropy then the EG update (9) results when the weight vectors are constrained to lie in P_{N-1}. The same distance function that motivates an algorithm is used in an amortized analysis to obtain the relative loss bounds for the algorithm.

IV. Relative Loss Bounds for GD and EG

We now proceed to the worst-case loss bounds for the algorithms GD and EG$^{\pm}$. In the following we assume that ϕ is a given sigmoid function, with Z an upper bound on its derivative so that $0 < \phi'(z) < Z$ for all $z \in \boldsymbol{R}$. We also let L_ϕ denote the matching loss function for ϕ as given in (3), and write $\mathrm{Loss}_\phi(\boldsymbol{w}, S)$ and $\mathrm{Loss}_\phi(A, S)$ for the empirical loss of a weight vector $\boldsymbol{w}$ and the total loss of an algorithm A for $L = L_\phi$ as defined in in (1) and (2). Our ultimate goal is to prove that on any sequence S the cumulative loss $\mathrm{Loss}_\phi(A, S)$ is not much higher than the empirical loss $\mathrm{Loss}_\phi(\boldsymbol{u}, S)$ for any fixed weight vector $\boldsymbol{u} \in \boldsymbol{R}^N$ against which we might be reasonably comparing our algorithm. Here we consider it reasonable to compare an algorithm against any weight vector within a given distance from the algorithm's start vector $\boldsymbol{w}_1$.

In the absence of any prior information one would typically choose a zero start vector, so this amounts to considering vectors with their norm bounded by some parameter $U > 0$. The three norms we use are the Euclidean length $||\boldsymbol{x}||_2$, the 1-norm $||\boldsymbol{x}||_1 = \Sigma_{i=1}^N |x_i|$, and the ∞-norm $||\boldsymbol{x}||_\infty = \max_i |x_i|$.

Before giving our main results (Theorem 1 for the GD algorithm and in Theorem 2 for the EG$^\pm$ algorithm) we explain the general form shared by the bounds for both algorithms. For each algorithm, we have two bounds. The simpler bounds have the form

$$\text{Loss}_\phi(A, S) \leq a\text{Loss}_\phi(\boldsymbol{u}, S) + b \tag{12}$$

where a is a small positive constant (two or 4/3), and the term b depends on the norms of the instances and the norms of the weight vectors $\boldsymbol{u}$ for which the inequality is being applied. This already shows that asymptotically the losses of the algorithms are within a multiplicative constant of the loss of the best fixed predictor. However, we can show an even stronger result: the constant a can actually be reduced to one at the expense of some lower order terms. Thus, the stronger form of our bounds look like

$$\text{Loss}_\phi(A, S) \leq \text{Loss}_\phi(\boldsymbol{u}, S) + c\text{Loss}_\phi(\boldsymbol{u}, S)^{1/2} + d \tag{13}$$

where the constants c and d depend on the norms of the instances and best weight vector $\boldsymbol{u}$. As we shall see, both the bounds of type (12) and of type (13) are valid only for specific values of the learning rate η, but finding these values is much simpler for the simpler bounds (12).

We have bounds in both the simpler form (12) and the stronger form (13) for both the GD and EG$^\pm$ algorithms. Therefore, we must examine the constants in the bounds in order to compare the performance of the algorithms. Since the bounds on the two algorithms use different norms in their constants, we can use the bounds to indicate which situations will tend to favor one algorithm over the other. We will compare the algorithms from this point of view after formally stating our bounds.

Theorem 1: Let GD be the gradient descent algorithm for transfer function ϕ and loss function L_ϕ, with start vector $\boldsymbol{w}_1$ and learning rate η. Consider a trial sequence $S = ((\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_\ell, y_\ell))$ such that for some $X > 0$ we have $||\boldsymbol{x}_t||_2 \leq X$ for $1 \leq t \leq \ell$. If we choose the learning rate $\eta = 1/(2X^2Z)$, then for any choice of start vector $\boldsymbol{w}_1$ and for all $\boldsymbol{u} \in \boldsymbol{R}^N$ we have

$$\text{Loss}_\phi(\text{GD}, S) \leq 2(\text{Loss}_\phi(\boldsymbol{u}, S) + (||\boldsymbol{u} - \boldsymbol{w}_1||_2 X)^2 Z). \tag{14}$$

Furthermore, for arbitrary positive constants K and U, if we choose the learning rate $\eta = (\sqrt{z^2 + z} - z)/(X^2 Z)$ where $z = U^2X^2Z/(2K)$, then for all $\boldsymbol{u} \in \boldsymbol{R}^N$ such that $\text{Loss}_\phi(\boldsymbol{u}, S) \leq K$ and $||\boldsymbol{u} - \boldsymbol{w}_1||_2 \leq U$ hold we have

$$\text{Loss}_\phi(\text{GD}, S) \leq \text{Loss}_\phi(\boldsymbol{u}, S) + UX\sqrt{2KZ} + 2(UX)^2 Z. \tag{15}$$

In particular, in the noise-free case with $\text{Loss}_\phi(\boldsymbol{u}, S) = 0$ for some $\boldsymbol{u}$ we see that choosing $\eta = 1/(2X^2Z)$ gives the loss bound $\text{Loss}_\phi(\text{GD}, S) \leq 2(||\boldsymbol{u} - \boldsymbol{w}_1||_2 X)^2 Z$.

Notice that if one applies Theorem 1 to the identity transfer function and square loss, one gets somewhat worse constant coefficients than those known earlier for that special case [7], [13]. This is because the square loss has some special properties that allow us to do a tighter optimization in choosing the learning rate.

The full proof of Theorem 1 is given in the Appendix. Here we give a brief outline of the main methods, which are direct generalizations of those applied for the case of identity transfer function and square loss [7], [13]. As in the motivation for the algorithms (Section III), the distance function d plays a key role in the analysis. For the GD algorithm, we again use the squared Euclidean distance $d(\boldsymbol{u}, \boldsymbol{w}) = \frac{1}{2}||\boldsymbol{u} - \boldsymbol{w}||_2^2$. The first goal is to prove an invariant over t and $\boldsymbol{u}$ of the form

$$\begin{aligned}\alpha\gamma L_\phi(y_t, \boldsymbol{w}_t \cdot \boldsymbol{x}_t) - \gamma L_\phi(y_t, \boldsymbol{u} \cdot \boldsymbol{x}_t)\\ \leq d(\boldsymbol{u}, \boldsymbol{w}_t) - d(\boldsymbol{u}, \boldsymbol{w}_{t+1})\end{aligned} \tag{16}$$

where α and γ are positive parameters that depend on the learning rate η. This invariant implies that for all vectors $\boldsymbol{u}$, whenever α times the loss of the algorithm is greater than the loss of $\boldsymbol{u}$ then the algorithm updates its weight vector so that it gets closer to $\boldsymbol{u}$. Intuitively, the net effect of the updates over a long sequence of trials is that the weight vector of the algorithm gets close to every $\boldsymbol{u}$ that has a low loss on that particular trial sequence. More formally, by summing (16) for $t = 1, \cdots, \ell$ (and observing that $d(\boldsymbol{u}, \boldsymbol{w}_{\ell+1}) \geq 0$) we obtain the bound

$$\text{Loss}_\phi \leq \frac{\text{Loss}_\phi(\boldsymbol{u}, S)}{\alpha} + \frac{d(\boldsymbol{u}, \boldsymbol{w}_1)}{\alpha\gamma}.$$

The right-hand side of this bound depends on η through the parameters α and γ, and based on what we know (or assume) about the values $\text{Loss}_\phi(\boldsymbol{u}, S)$ and $d(\boldsymbol{u}, \boldsymbol{w}_1)$ we can attempt to minimize the right-hand side by choosing an appropriate value for η.

We now state our upper bounds on the EG$^\pm$ algorithm. In addition to the learning rate η, this algorithm also takes the scaling parameter U. The greater the value U, the higher the loss bounds will be. On the other hand, the relative loss bounds are only guaranteed to hold for vectors $\boldsymbol{u}$ with $||\boldsymbol{u}||_1 \leq U$.

Theorem 2: Let EG$^\pm$ be the EG algorithm with positive and negative weights for transfer function ϕ and loss function L_ϕ, with start vector $\boldsymbol{w}_1$, scaling factor U, and learning rate η. Consider a trial sequence $S = ((\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_\ell, y_\ell))$ such that for some $X > 0$ we have $||\boldsymbol{x}_t||_\infty \leq X$ for $1 \leq t \leq \ell$. If we choose the learning rate $\eta = 1/(4(UX)^2Z)$ then for all $\boldsymbol{u} \in \boldsymbol{R}^N$ with $||\boldsymbol{u}||_1 \leq U$ we have

$$\text{Loss}_\phi(\text{EG}^\pm) \leq \tfrac{4}{3}\,\text{Loss}_\phi(\boldsymbol{u}, S) + 4(UX)^2 Z\, \ln(2N). \tag{17}$$

Furthermore, for arbitrary positive constants K and D, if we choose the learning rate $\eta = (\sqrt{z^2 + z} - z)/((UX)^2 Z)$ where $z = (UX)^2 ZD/K$, then for all $\boldsymbol{u} \in \boldsymbol{R}^N$ such that $||\boldsymbol{u}||_1 \leq U$ and $\text{Loss}_\phi(\boldsymbol{u}, S) \leq K$ hold we have

$$\begin{aligned}\text{Loss}_\phi(\text{EG}^\pm) \leq{}& \text{Loss}_\phi(\boldsymbol{u}, S) + 2UX\sqrt{KZ\,\ln(2N)}\\ &+ 4(UX)^2 Z\,\ln(2N).\end{aligned} \tag{18}$$

Again, consider the noise-free special case: If $\text{Loss}_\phi(\boldsymbol{u}, S) = 0$ where $||\boldsymbol{u}||_1 \leq U$, then examining the limit $K \to 0$ shows that the learning rate $\eta = 1/(2(UX)^2 Z)$

gives the loss bound $\mathrm{Loss}_\phi(\mathrm{EG}^\pm) \le 4(UX)^2 Z \ \ln(2N)$. As with GD, for the identity transfer function and square loss it is possible to obtain somewhat sharper constant coefficients by employing techniques particular to that special case [13].

The full proof of Theorem 2 is given in the Appendix. The bound given for $\mathrm{EG}^\pm$ is obtained by a straightforward reduction from a similar bound for EG. The proof for EG follows the same basic line that was sketched for GD after Theorem 1. The difference is that the relative entropy (10) replaces the Euclidean distance as the distance function. Hence, the details of obtaining the invariant (16) are different from the case of GD. After obtaining (16), the proofs for EG and GD proceed similarly.

Notice that an upper bound for the norms of the instances is needed to determine the learning rates η that lead to the simpler set of bounds, (14) and (17) and the $\mathrm{EG}^\pm$ algorithm also needs to know the scaling factor U beforehand. The stronger bounds (15) and (18) are more problematical in this respect, as the learning rate depends not only on the norms of the instances and the vectors $\boldsymbol{u}$ we wish to compare against, but it also depends on the value K bounding the loss $\mathrm{Loss}_\phi(\boldsymbol{u}, S)$ of some comparison vector. We would not usually expect to know the exact loss of any comparison vector beforehand, and if we choose the upper bound K too conservatively, it will adversely affect the bound we achieve. In particular, to actually get a bound of the form (13) from (15) or (18) we need to have our upper bound K within a constant factor of the actual loss $\mathrm{Loss}_\phi(\boldsymbol{u}, S)$. In practice, it would probably make more sense to tune the learning rate using one of the usual methods instead of trying to compute it using the theorems. We should just consider it a bonus if our theoretical results guide us a little in the tuning task. In certain limited cases it has been possible to show that based on worst-case upper bounds one can in an on-line manner systematically update the learning rate so that one achieves almost as good a result as by using the optimal learning rate from the beginning [6], [7].

It is important to realize that we bound the total loss of the algorithms over *any* adversarially chosen sequence of examples where the input vectors satisfy the norm bound. Although we state the bounds in terms of loss on the data, they imply that the algorithms must also perform well on new unseen examples, since the bounds still hold when an adversary adds these additional examples to the end of the sequence. A formal treatment of this appears in several places [13], [16]. Furthermore, in contrast to standard convergence proofs [18], we bound the loss on the *entire* sequence of examples instead of studying the convergence behavior of the algorithm when it is arbitrarily close to the best weight vector.

Theorems 1 and 2 give some insight into whether GD or $\mathrm{EG}^\pm$ is likely to be better in particular situations. The bounds for the $\mathrm{EG}^\pm$ algorithm grow with the maximum component of the input vectors and the one-norm of the best weight vector from the comparison class. On the other hand, the loss bounds for the GD algorithm grow with the two-norm (Euclidean length) of both vectors. Thus when the best weight vector is sparse, having few significant components, and the input vectors are dense, with several similarly sized components, the bound for the $\mathrm{EG}^\pm$ algorithm is better than the bound for the GD algorithm. To make the comparison more concrete, consider the noise-free situation with $\mathrm{Loss}_\phi(\boldsymbol{u}, S) = 0$ for some $\boldsymbol{u}$, so that the bounds simplify. Assume $\boldsymbol{x}_t \in \{-1, 1\}^N$ and $\boldsymbol{u} \in \{-1, 0, 1\}^N$ with only k nonzero components in $\boldsymbol{u}$. We can then take $X_2 = \sqrt{N}$, $X_\infty = 1$, $||\boldsymbol{u}||_2 = \sqrt{k}$, and $||\boldsymbol{u}||_1 = k$. The loss bounds bounds (15) and (18) become $4k^2 Z \ln(2N)$ for $\mathrm{EG}^\pm$ and $2kNZ$ for GD, so for $N \gg k$ the bounds for the $\mathrm{EG}^\pm$ algorithm are clearly better. On the other hand, the GD algorithm's bounds has the advantage when each input vector is sparse and the best weight vector is dense, with all its components having roughly the same magnitude. For example, if the inputs $\boldsymbol{x}_t$ are the rows of an $N \times N$ unit matrix and $\boldsymbol{u} \in \{-1, 1\}^N$, then $X_2 = X_\infty = 1$, $||\boldsymbol{u}||_2 = \sqrt{N}$, and $U = N$. In this case, the upper bounds become $4N^2 Z \ \ln(2N)$ for $\mathrm{EG}^\pm$ and $2NZ$ for GD, so here the bound for GD is better.

Of course, a comparison of the upper bounds is meaningless unless the bounds are known to be reasonably tight. Our experiments with artificial random data suggest that the upper bounds are not tight but still give us a good idea on which algorithm performs better in certain conditions. In particular, the experimental evidence confirms that for large N $\mathrm{EG}^\pm$ is much better than GD when the best weight vector is sparse. See Section V for details.

The bounds we give in this paper are very similar to the bounds obtained obtained earlier [13] for the comparison class of linear functions and the square loss. There it was first observed how the relative performances of the GD and $\mathrm{EG}^\pm$ algorithms relate to the norms of the input vectors and the best weight vector in the linear case. It should be noted that the bounds of [13] have better constants than what would result from applying Theorems 1 and 2 to the identity transfer function. This is because the special properties of the square loss and identity function allow some additional fine-tuning in the proofs. More recently, Cesa-Bianchi [5] has analyzed these algorithms in terms of arbitrary convex loss functions that do not necessarily match the transfer function. This results in bounds with essentially a $\sqrt{\ell}$ factor, depending on the length ℓ of the trial sequence, replacing in bounds (15) and (18) the factor $\sqrt{K}$ depending on an upper bound K for the loss of some weight vector $\boldsymbol{u}$. Thus, the bounds are not directly comparable to the ones we have here. Vovk [20] has bounds where this factor is decreased to $\log\ \ell$, but he only has bounds analogous to the GD bounds where the upper bound U_2 for the two-norm of the weight vectors is used.

In the linear case it has also been possible to obtain lower bounds [7], [13] that are closely related to the upper bounds we have. In particular, these lower bounds include products of norms of the input and weight vectors. This is quite natural, since simply multiplying the inputs or weights by a constant would cause a corresponding increase in the square losses. For loss functions such as the relative entropy, the loss $L(y, \hat{y})$ is bounded for fixed $\hat{y}$, so intuitively one would expect that the loss of the algorithm remains bounded even if, say, the norms of the instances are unbounded. This intuition is indeed correct at least in some special cases, as the following example shows.

Assume again that we are in the noise-free case, i.e., $y_t = \phi(\boldsymbol{u} \cdot \boldsymbol{x}_t)$ where ϕ is the logistic function. Then, in particular, given an example $(\boldsymbol{x}_t, y_t)$ we can easily obtain the value $\boldsymbol{u} \cdot \boldsymbol{x}_t = \phi^{-1}(y_t)$. Consider now the following simple algorithm: If $\boldsymbol{x}_t$ is in the span of the previous input vectors, i.e., $\boldsymbol{x}_t = \Sigma_{i=1}^{t-1} a_i \boldsymbol{x}_i$ for some scalars a_i, it is possible to make the exactly correct prediction $\hat{y}_t = \phi(\boldsymbol{u} \cdot \boldsymbol{x}_t)$ by using the fact $\boldsymbol{u} \cdot \boldsymbol{x}_t = \Sigma_{i=1}^{t-1} a_i \boldsymbol{u} \cdot \boldsymbol{x}_i$. In the case that $\boldsymbol{x}_t$ is not in the span of the previous input vectors, we simply predict $\hat{y}_t = \frac{1}{2}$. Now the algorithm incurs loss only when the input vector is not in the span of the previous ones, which can happen at most N times. Since $L_\phi(y, \frac{1}{2}) \le \ln\, 2$ for $0 \le y \le\, 1$, the loss in each case is at most $\ln\, 2$, and we see that in the noise-free case our simple algorithm achieves a loss at most $N\, \ln\, 2$ regardless of the norms of $\boldsymbol{u}$ and $\boldsymbol{x}_t$.

V. Simulation Results

As indicated by the discussion in Section IV, the relative loss bounds suggest that the EG$^\pm$ algorithm should outperform GD when the target is sparse but the instances dense. Conversely, if the target is dense but the instances sparse, the bounds suggest that GD will outperform EG$^\pm$. There are at least two reasons why these conclusions might not hold in practice.

First, the loss bounds hold over *all* trial sequences; in other words, they are worst-case bounds by nature. In practice, although the environment generating the trials may be less than helpful, it almost certainly is not trying to foil specific algorithms. This makes it important to relate the worst-case bounds to the performance of the algorithms in less adversarial settings.

Second, our bounds are only *upper bounds* for the actual losses. As long as we lack matching lower bounds, we have no guarantee that the bounds are tight. Thus, the difference we have observed between the algorithms might be just an artifact of our proof technique. For the linear case some good lower bounds are known [7], [13], but for nonlinear transfer functions such bounds have not been found. In particular, as suggested by the discussion of a simple alternative algorithm at the end of Section IV, it is not clear whether the norms of the input and weight vectors are really necessary for nonlinear transfer functions.

To address these concerns, we present some experiments on simple artificially generated data. Obviously, such experiments will not really tell us how the algorithms would perform in actual applications. On the other hand, since we have control over how the data is generated, making conclusions about the factors affecting the performance of the algorithms is much simpler than with data from real applications. The goal of the experiments shown here is to see how well the predictions based on the upper bounds match the actual behavior of the algorithms. In particular, we wish to observe at least qualitatively how the losses of the different algorithms behave as a function of the input dimensionality N for large N. To facilitate easy comparison between the algorithms, the experiments were performed with noise-free data. In a very noisy environment, the effects of noise would dominate the total losses of both algorithms, making the differences between the algorithms less obvious. Possible differences in noise tolerance between the algorithms are not the subject of these simple experiments.

We now describe the experimental setup in more detail. The input dimension N took on four different values, 100, 200, 400, and 800. For each data set used in the experiment a *target* weight vector $\boldsymbol{u} \in \boldsymbol{R}^N$ was fixed and a set of m input vectors $\boldsymbol{x}_t \in \boldsymbol{R}^N$ generated randomly. The hyperbolic tangent was chosen as the transfer function. As the data was meant to be noise-free, the desired output $\boldsymbol{y}_t$ was then given by $y_t = \tanh(\boldsymbol{u} \cdot \boldsymbol{x}_t)$, the standard sigmoid when the labels lie in the interval [−1,+1]. The number m of input vectors was 15 000 in every experiment. Even with 800 input features, the simplicity of the algorithms allowed repeating the experiments as many times as needed without problems with computation time.

Two kinds of data sets were used, differing in the choice of the target $\boldsymbol{u}$ and the inputs $\boldsymbol{x}_t$. In the data sets with a *sparse target*, five components of the target $\boldsymbol{u}$ were randomly chosen to have value 1 or -1, and the remaining $N-5$ components were set to zero. Thus, the targets were extremely sparse; only five out of the hundreds of features were relevant. The instances were chosen independently of each other from the uniform distribution over $\{-1, 1\}^N$. In other words, each feature in each input vector received randomly either the value one or the value -1.

Another possibility for the sparse target experiments would have been to choose the number of relevant features to be some fixed proportion of the total number of features, say $0.05N$. However, in these experiments we wished to test the prediction made based on the upper bounds that for a *fixed* number k of relevant variables, the loss of GD grows linearly in N, while the loss of EG grows only logarithmically in N. To see why this is an interesting scenario, consider the standard trick of doing nonlinear regression by applying a linear regression algorithm in a high-dimensional feature space where the features are some nonlinear functions of the original input variables. For example, to fit polynomials of degree three over n variables x_i, one would fit a linear function to the $O(n^3)$ features $x_i x_j x_k$, $1 \le i, j, k \le n$. In such situations, the dimensionality of the feature space can get quite high while the set of relevant feature remains small.

In contrast, every component was relevant in the data sets with a *dense target*. Each component of the target $\boldsymbol{u}$ was randomly chosen to be one or -1. However, for these data sets the input vectors were made sparse by choosing independently for each input vector five components that randomly received value one or -1, while the remaining $N-5$ input variables were set to zero. Thus, at any given time an input variable had only a $5/N$ chance of having a nonzero value.

This gave a total of eight different distributions for generating the input data, sparse and dense versions for each of the four different input dimensions. From each of these distributions, 20 data sets were generated independently of each other. The first ten data sets were always used to find the optimal learning rates for the GD and EG$^\pm$ algorithms for the input distribution in question. The actual performance of the algorithms was then measured by running them on the remaining ten data sets using the optimal learning rate. Here

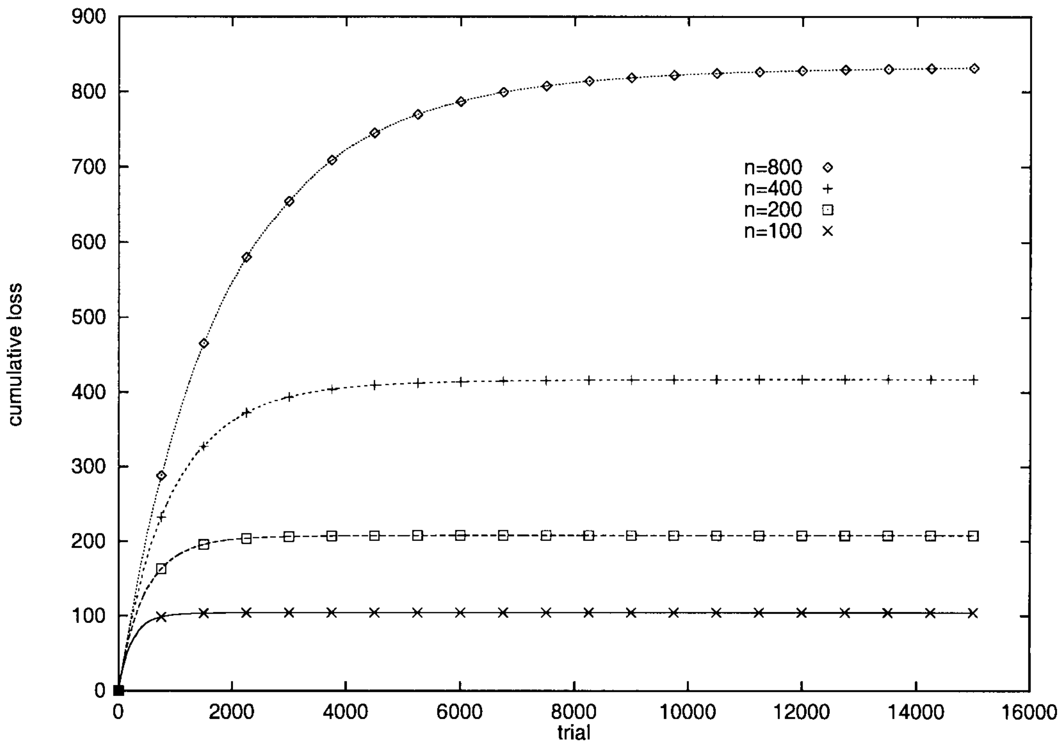


Fig. 2. GD with sparse target.

the optimal learning rate was taken to be the one that achieved the least total loss (averaged over the ten data sets) over the sequence of 15 000 input vectors.

It is well known that tuning the learning rate can have a huge effect on the performance of an algorithm. (This was also true for the experiments described here.) The tuning method described above was chosen to make certain that any differences observed between the algorithms would not result from a failure to tune the learning rates properly. In practice, limited availability of data might preclude such a brute-force approach. Actually, as one might expect, we found that the optimal learning rates did not differ much between different data sets generated from the same distribution, so using 20 data sets for each distribution may have been excessive.

Another possibility for tuning would have been to use the same learning rates that are used in the proof of the relative loss bounds. We also performed some experiments using these alternative learning rates. Later we discuss these experiments and compare the observed losses to the upper bounds given in the theorems. However, since our particular concern here is determining if the behavior predicted by our upper bound analysis is close to the algorithms' actual behavior, it seems inappropriate to base the learning rate on the theoretical analysis.

Let us now consider the experimental results. The results are presented in Figs. 2–5 as cumulative loss curves. Each curve shows how the cumulative loss of an algorithm increases as a function of the trial number as the algorithm is given more and more examples from a certain distribution. Thus, the curves start from zero as initially the algorithm has accumulated no loss, and since the data are noise-free the curve eventually levels out, showing that the algorithm has learned the target weight vector and makes no more errors. As explained above, each curve is an average over ten runs on independent data sets from one distribution.

For both sparse and dense targets there are separate figures for the two algorithms. This makes it easy to see how the performance of the algorithm degrades as the dimensionality of the input increases but the other parameters of the problem are kept the same. (Remember that each gap between two adjacent curves represents doubling the number of input variables.) Comparing the algorithms against each other on our data sets would be less interesting, since the data sets are completely artificial and the experiments clearly illustrate how we can give the advantage to one algorithm or the other by suitably modifying the parameters within our experimental setting. For similar reasons, the results have not been represented in a numerical form. The only really interesting quantitative question arising from these experiments is determining the actual functional form of the dependence between N and the total loss for the various algorithms, but that cannot be reliably answered without making more experiments with larger values of N. Thus, we prefer to stay on a qualitative level in our comparison.

Fig. 2 shows the curves for GD with sparse targets. Notice how doubling the input dimension roughly doubles the total loss, as expected. As explained above, each of the four curves is a result of running the algorithm on the optimal learning rate determined by experiments. For the input distributions used for generating these curves, the optimal learning rate turned out to

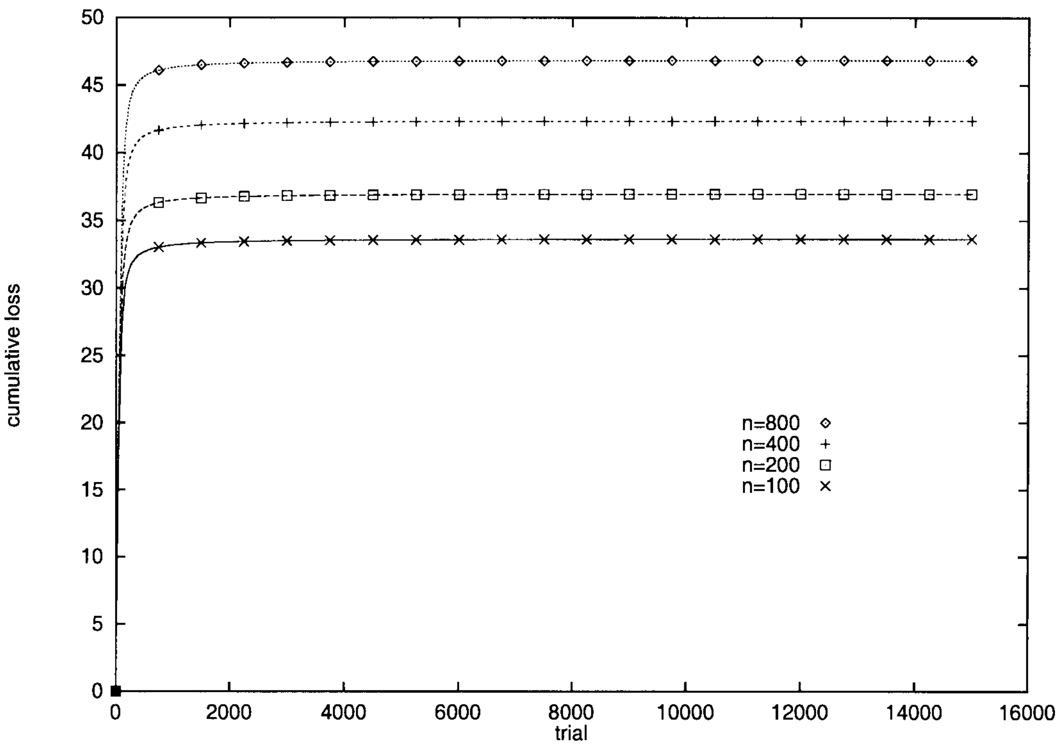


Fig. 3. EG with sparse target.

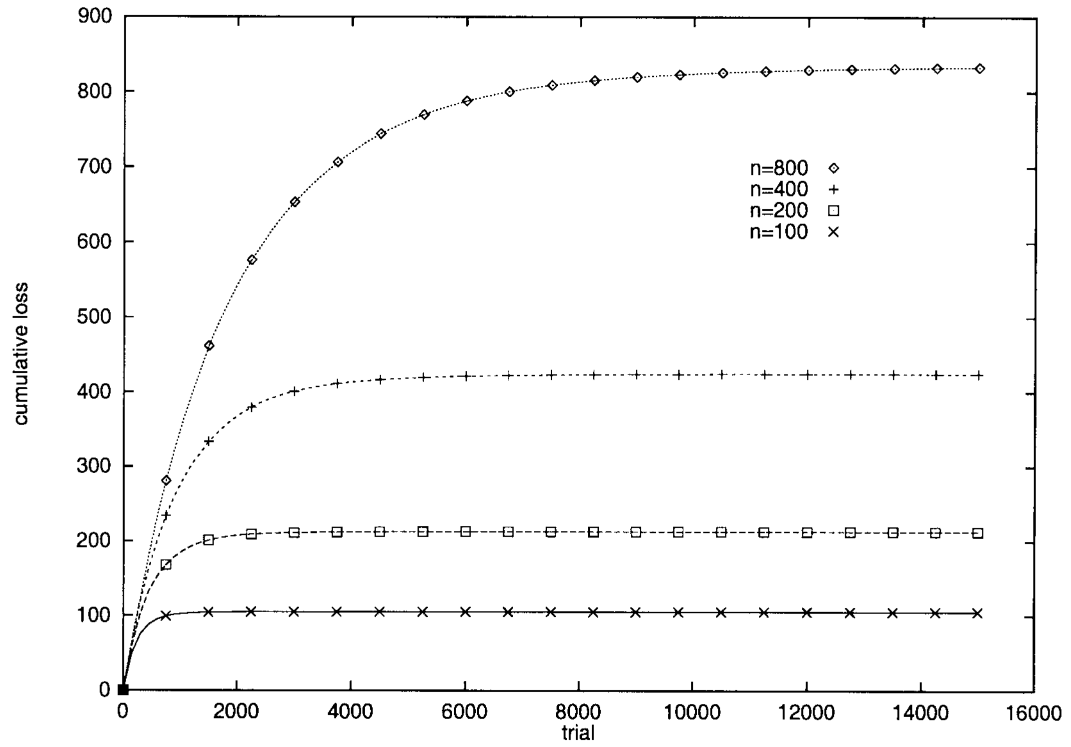


Fig. 4. GD with dense target.

be roughly three times the learning rate suggested by Theorem 1 (regardless of N). Using the the learning rate from Theorem 1 led to roughly twice as large losses as the ones shown here. Comparing further to the theoretical results, we notice that the upper bound given for the loss in Theorem 1 is about five times as large as the actual loss achieved using the learning rate from the theorem. Our conclusion from this is that the upper bounds of Theorem 1 give a good qualitative picture of the actual behavior of the algorithm on sparse targets, although the constant coefficients certainly are off.

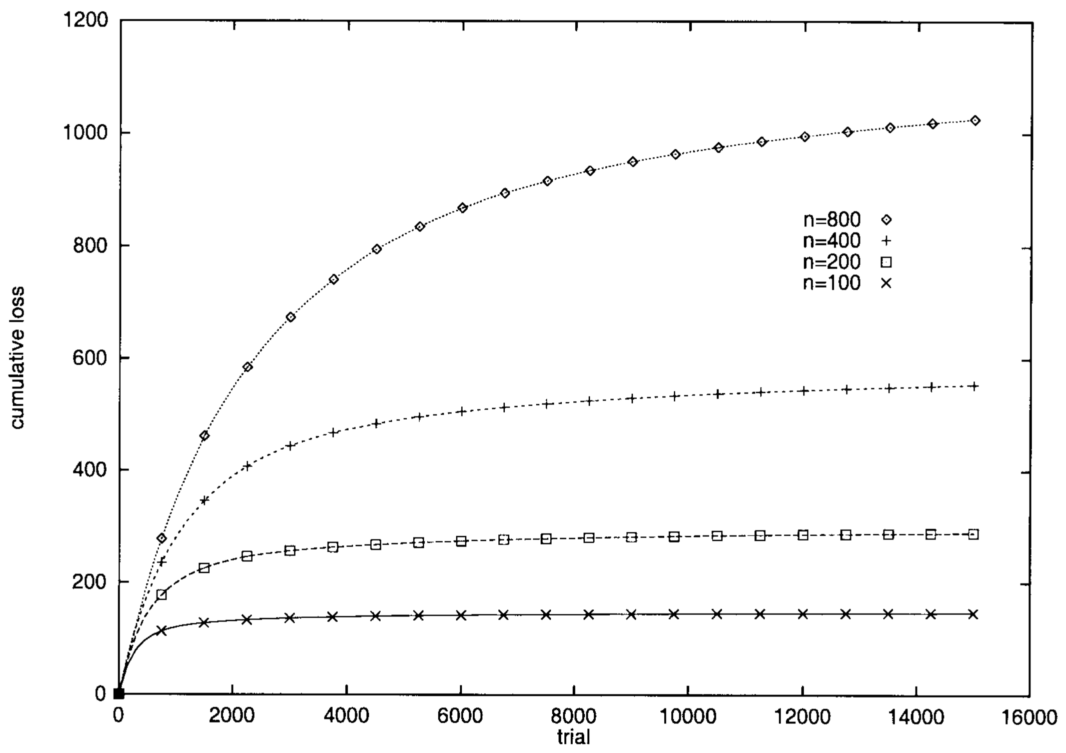


Fig. 5. EG with dense target.

Compare now Figs. 2 and 3, which shows the performance of EG$^\pm$ on the same data sets. It is obvious that there is a qualitative difference, in favor of EG$^\pm$, in how the algorithms react to increasing the number of input variables while keeping the number of relevant inputs fixed. This is what we expected from the upper bounds, and verifying that the difference actually exists at least in this experimental setup is the main conclusion we make from these simulations.

If we compare the performance of EG$^\pm$ as presented in Fig. 3 to Theorem 2, we again see that the predictions of the theorem are qualitatively reasonable but off by at least moderately large constant factors. Now the optimal learning rate is about 15 times the learning rate given in Theorem 2, and the upper bound given in Theorem 2 is about twice the actual loss the algorithm incurs using the learning rate from the theorem. However, the upper bounds of Theorem 2 exceed the losses seen in Fig. 2 for the actually optimal learning rate by a factor of about 15.

We now go to the dense targets, which we expect to favor GD. The cumulative loss curves for GD with dense targets are given in Fig. 4. Theorem 1 gives for GD exactly the same upper bounds with dense targets (and sparse inputs) as it gives with sparse targets (and dense inputs). Indeed, Figs. 2 and 4 are quite similar. Also the experiments with different learning rates show no differences between these two cases for GD.

Finally, Fig. 5 shows the results for EG$^\pm$ with dense targets. Comparing this with Fig. 4 we see that the difference between GD and EG$^\pm$ is much less clear than the upper bounds would suggest ($O(N)$ versus $O(N^2 \log N)$). It seems that the learning rates suggested by Theorem 2 are much too small, and get worse as N increases. For $N = 800$ the optimal learning rate turned out to be about 300 000 times the one suggested in the theorem. This naturally implies that the upper bounds of Theorem 2 are also ridiculously pessimistic for this case; for $N = 800$ they give a loss bound of roughly 4.7 million.

Thus, the main conclusion of our experiments is that EG$^\pm$ outperforms GD with sparse targets even with simple random data. The comparison with dense targets seems to be much less clear. We have no good explanation for the fact that the bounds for EG$^\pm$ with dense targets are quite bad, whereas for EG$^\pm$ with sparse targets and for GD with both dense and sparse targets the bounds seemed accurate within reasonable constant factors. However, a quick check of the more well-studied linear regression setting indicated that the bounds for EG$^\pm$ also seem quite pessimistic for the identity transfer function as well.

VI. Open Problems

Although the presence of local minima in multilayer networks makes it difficult to obtain worst case bounds for gradient-based algorithms, it may be possible to analyze slightly more complicated settings than just a single neuron. One such generalization would be neural networks with multiple output nodes but no hidden nodes. This is essentially the setting analyzed in [14], and includes for example multiclass logistic regression.

As the bounds for EG$^\pm$ depend only logarithmically on the input dimension, the following approach may be feasible. Instead of using a multilayer net, use a single (linear or sigmoided) neuron on top of a large set of basis functions that are arbitrarily chosen nonlinear functions of the original input variables. The logarithmic growth of the loss bounds in the number of such basis functions mean that large numbers of basis functions can be tried.

As noted above, the matching loss for the logistic transfer function is the entropic loss, so this pair does not create local minima. No bounded transfer function matches the square loss in this sense [1], [3], [4], and thus it seems impossible to get the same kind of strong loss bounds for a bounded transfer function and the square loss as we have for any (increasing and differentiable) transfer function and its matching loss function.

The bounds of this paper are worst-case bounds and our experiments on artificial random data indicate that the bounds may not be tight when the input values and best weights are large. However, we feel that the bounds do indicate the relative merits of the algorithms in different situations. Further research needs to be done to tighten the bounds and obtain useful lower bounds.

Appendix A
Proofs of Relative Loss Bounds

A.1. Obtaining Cumulative Loss Bounds from Progress Invariants

In this section we show how a progress invariant can be used to obtain cumulative loss bounds over a sequence of trials. Here, as in the rest of the Appendix, we assume that ϕ is an arbitrary sigmoid function and L_ϕ its matching loss function. Further, we assume that Z is an upper bound for the derivative of ϕ. Recall that by a distance function we mean any mapping d from $\boldsymbol{R}^N \times \boldsymbol{R}^N$ to $[0,\infty]$ such that $d(\boldsymbol{w},\boldsymbol{w})=0$ for all $\boldsymbol{w}$, and $d(\boldsymbol{w},\boldsymbol{w}')>0$ when $\boldsymbol{w}\neq\boldsymbol{w}'$.

Lemma 3: Let d be an arbitrary distance function. Let $\boldsymbol{w}_t$ be the weight vector of an algorithm A before trial t in a trial sequence $S=((\boldsymbol{x}_1,y_1),\cdots,(\boldsymbol{x}_\ell,y_\ell))$, and let $\boldsymbol{u}\in\boldsymbol{R}^N$ be arbitrary. For arbitrary positive constants α and γ, if for all t the condition

$$\begin{aligned}\alpha\gamma L_\phi(y_t,\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t)) - \gamma L_\phi(y_t,\phi(\boldsymbol{u}\cdot\boldsymbol{x}_t))\\ \le d(\boldsymbol{u},\boldsymbol{w}_t)-d(\boldsymbol{u},\boldsymbol{w}_{t+1})\end{aligned} \tag{19}$$

holds, then

$$\mathrm{Loss}_\phi(A,S)\le\frac{\mathrm{Loss}_\phi(\boldsymbol{u},S)}{\alpha}+\frac{d(\boldsymbol{u},\boldsymbol{w}_1)}{\alpha\gamma}. \tag{20}$$

Proof: Summing for $t=1,\cdots,\ell$ gives

$$\begin{aligned}&\alpha\gamma\mathrm{Loss}_\phi(A,S)-\gamma\mathrm{Loss}_\phi(\boldsymbol{u},S)\\ &\quad\le d(\boldsymbol{u},\boldsymbol{w}_1)-d(u,\boldsymbol{w}_{\ell+1})\le d(\boldsymbol{u},\boldsymbol{w}_1)\end{aligned}$$

and solving for $\mathrm{Loss}_\phi(A,S)$ yields (20). □

The following lemma will be helpful later when we tune the learning rate η in order to transform bounds of the form (20) into something more intuitive. First, by straightforward calculus we can see that the value of the expression $\sqrt{z^2+z}-z$ increases monotonically from zero to $1/2$ as z increases from zero to ∞. In particular, $0\le\sqrt{z^2+z}-z\le 1$ holds for all $z\ge 0$.

Lemma 4: For all $z>0$, if $\alpha=\sqrt{z^2+z}-z$ then

$$\frac{\alpha}{1-\alpha}+\frac{z}{\alpha(1-\alpha)}\le 2\sqrt{z}+4z.$$

Proof: We can write $\alpha/(1-\alpha)+z/(\alpha(1-\alpha)) = 2z/(\sqrt{z(z+1)}-z)$. Therefore, it suffices to show that $S(z)\ge 1$ for all z, where

$$S(z)=\frac{(\sqrt{z}+2z)(\sqrt{z(z+1)}-z)}{z}.$$

This can be rewritten as $S(z)=\sqrt{z+1}-\sqrt{z}-2z+2\sqrt{z(z+1)}$. We then have

$$S'(z)=\frac{1}{2\sqrt{z+1}}-\frac{1}{2\sqrt{z}}+\sqrt{\frac{z+1}{z}}+\sqrt{\frac{z}{z+1}}-2.$$

We now rewrite $S'(z)$ as $T(\sqrt{z/(z+1)})$ where

$$\begin{aligned}T(r)&=\frac{\sqrt{1-r^2}}{2}-\frac{\sqrt{1-r^2}}{2r}+\frac{1+r^2}{r}-2\\ &=\frac{(1-r)^{3/2}}{r}\left(\sqrt{1-r}-\frac{\sqrt{1+r}}{2}\right).\end{aligned}$$

Since the only solutions to $T(r)=0$ are $r=1$ and $r=3/5$, the only solution for $S'(z)=0$ with $0<z<\infty$ is $z=9/16$. Since $S(9/16)=5/4>1$ and $\lim_{z\to 0+}S(z)=\lim_{z\to\infty}S(z)=1$, we have $S(z)\ge 1$ for $z\ge 0$. □

A.2. Bounds for the Gradient Descent Algorithm

Lemma 5: Let GD be the gradient descent algorithm for transfer function ϕ and loss function L_ϕ, with start vector $\boldsymbol{w}_1$ and learning rate η. Consider a trial sequence $S=((\boldsymbol{x}_1,y_1),\cdots,(\boldsymbol{x}_\ell,y_\ell))$ such that for some $X>0$ we have $\|\boldsymbol{x}_t\|_2\le X$ for $1\le t\le\ell$, and let $\boldsymbol{w}_t$ be the weight vector of GD before trial t in the sequence S. Then for all $\boldsymbol{u}\in\boldsymbol{R}^N$ we have

$$\begin{aligned}(\eta-\eta^2X^2Z)L_\phi(y_t,\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t))-\eta L_\phi(y_t,\phi(\boldsymbol{u}\cdot\boldsymbol{x}_t))\\ \le\tfrac{1}{2}\left(\|\boldsymbol{u}-\boldsymbol{w}_t\|_2^2-\|\boldsymbol{u}-\boldsymbol{w}_{t+1}\|_2^2\right).\end{aligned} \tag{21}$$

Proof: For the gradient descent update, we have $\boldsymbol{w}_{t+1}=\boldsymbol{w}_t+\eta(y_t-\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t))\boldsymbol{x}_t$, and

$$\begin{aligned}&\|\boldsymbol{u}-\boldsymbol{w}_t\|_2^2-\|\boldsymbol{u}-\boldsymbol{w}_{t+1}\|_2^2\\ &\quad=-2\eta(y_t-\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t))(\boldsymbol{w}_t\cdot\boldsymbol{x}_t-\boldsymbol{u}\cdot\boldsymbol{x}_t)\\ &\qquad-\eta^2\|\boldsymbol{x}_t\|_2^2(y_t-\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t))^2\\ &\quad\ge-2\eta(y_t-\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t))(\boldsymbol{w}_t\cdot\boldsymbol{x}_t-\boldsymbol{u}\cdot\boldsymbol{x}_t)\\ &\qquad-\eta^2X^2(y_t-\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t))^2.\end{aligned}$$

Hence, to prove (21) it is sufficient to show $F(y_t,\boldsymbol{w}_t\cdot\boldsymbol{x}_t,\boldsymbol{u}\cdot\boldsymbol{x}_t,\eta)\le 0$ where

$$\begin{aligned}F(y,p,r,\eta)&=(\eta-\eta^2X^2Z)L_\phi(y,\phi(p))-\eta L_\phi(y,\phi(r))\\ &\quad+\eta(y-\phi(p))(p-r)+\frac{\eta^2X^2}{2}(y-\phi(p))^2.\end{aligned}$$

We have

$$\begin{aligned}\frac{\partial F(y,p,r,\eta)}{\partial r}&=\eta(y-\phi(r))-\eta(y-\phi(p))\\ &=\eta(\phi(p)-\phi(r))\end{aligned}$$

and

$$\frac{\partial^2F(y,p,r,\eta)}{\partial r^2}=-\eta\phi'(r)\le 0.$$

Therefore, for fixed y, p, and η, the value $F(y,p,r,\eta)$ is maximized when $r = p$. Therefore, it is sufficient to show $G(y,p) \leq 0$ where

$$\begin{aligned} G(y,p) &= \frac{F(y,p,p,\eta)}{X^2\eta^2} \\ &= -ZL_\phi(y,\phi(p)) + \frac{(y-\phi(p))^2}{2}. \end{aligned}$$

We now apply the fact that

$$\frac{\partial L_\phi(y,\hat{y})}{\partial y} = \phi^{-1}(y) - \phi^{-1}(\hat{y}) \tag{22}$$

which follows directly from (3). This gives us

$$\frac{\partial G(y,p)}{\partial y} = -Z(\phi^{-1}(y) - p) + y - \phi(p).$$

Hence, for $y = \phi(p)$ we have both $G(y,p) = 0$ and $\partial G(y,p)/\partial y = 0$, and to prove $G(y,p) \leq 0$ it suffices to show that $\partial^2 G(y,p)/\partial y^2 \leq 0$ always holds. But the second derivative is always nonpositive since

$$\frac{\partial^2 G(y,p)}{\partial y^2} = -\frac{Z}{\phi'(\phi^{-1}(y))} + 1$$

and Z is an upper bound on ϕ'. This completes the proof. □

Proof of Theorem 1: From Lemma 5 with $\eta = 1/(2X^2Z)$, we have for all t the bound

$$\begin{aligned} &\frac{L_\phi(y_t,\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t))}{4X^2Z} - \frac{L_\phi(y_t,\phi(\boldsymbol{u}\cdot\boldsymbol{x}_t))}{2X^2Z} \\ &\quad\leq \frac{1}{2}(\|\boldsymbol{u}-\boldsymbol{w}_t\|_2^2 - \|\boldsymbol{u}-\boldsymbol{w}_{t+1}\|_2^2). \end{aligned}$$

Applying Lemma 3 with $\alpha = 1/2$ and $\gamma = 1/(2X^2Z)$ yields the bound (14).

We now show (15). From Lemma 5 and Lemma 3 with $\alpha = 1 - \eta X^2 Z$ and $\gamma = \eta$ we have

$$\begin{aligned} \text{Loss}(\text{GD},S) &\leq \frac{\text{Loss}_\phi(\boldsymbol{u},S)}{1-\eta X^2Z} + \frac{U^2}{2\eta(1-\eta X^2Z)} \\ &\leq \text{Loss}_\phi(\boldsymbol{u},S) + \frac{\eta X^2ZK}{1-\eta X^2Z} \\ &\quad + \frac{U^2X^2Z}{2\eta X^2Z(1-\eta X^2Z)} \\ &= \text{Loss}_\phi(\boldsymbol{u},S) + K\Bigg(\frac{\eta X^2Z}{1-\eta X^2Z} \\ &\quad + \frac{U^2X^2Z}{2K\eta X^2Z(1-\eta X^2Z)}\Bigg). \end{aligned}$$

Since $\eta = (\sqrt{z^2+z} - z)/(X^2Z)$ with $z = U^2X^2Z/(2K)$, the term multiplying K is in the form required by Lemma 4. Applying this lemma we get

$$\begin{aligned} \text{Loss}(\text{GD},S) &\leq \text{Loss}_\phi(\boldsymbol{u},S) + K(2\sqrt{z}+4z) \\ &= \text{Loss}_\phi(\boldsymbol{u},S) + UX\sqrt{2ZK} + 2U^2X^2Z \end{aligned}$$

as desired. □

A.3. Bounds for the Exponentiated Gradient Algorithm

Recall that for two probability vectors $\boldsymbol{u},\boldsymbol{w} \in P_{N-1}$ we write $d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w})$ for the relative entropy defined in (10).

Lemma 6: Let EG be the exponentiated gradient algorithm for transfer function ϕ and loss function L_ϕ, with start vector $\boldsymbol{w}_1 \in P_{N-1}$ and learning rate η. Consider a trial sequence $S = ((\boldsymbol{x}_1,y_1),\cdots,(\boldsymbol{x}_\ell,y_\ell))$ such that for some $R > 0$ we have $\max_i x_{t,i} - \min_i x_{t,i} \leq R$ for $1 \leq t \leq \ell$, and let $\boldsymbol{w}_t$ be the weight vector of EG before trial t in the sequence S. Then for all $\boldsymbol{u} \in P_{N-1}$ we have

$$\begin{aligned} &(\eta - \eta^2R^2Z/4)L_\phi(y_t,\phi(\boldsymbol{w}_t\cdot\boldsymbol{x}_t)) - \eta L_\phi(y_t,\phi(\boldsymbol{u}\cdot\boldsymbol{x}_t)) \\ &\quad\leq d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w}_t) - d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w}_{t+1}). \end{aligned} \tag{23}$$

Proof: Consider the update $w_{t+1,i} = w_{t,i}\beta^{x_{t,i}}/\Sigma_j\ w_{t,j}\beta^{x_{t,j}}$ for some $\beta > 0$. For $\beta = e^{\eta(y_t-\hat{y}_t)}$ this is the update of the EG algorithm. Then $d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w}_t) - d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w}_{t+1}) = \boldsymbol{u}\cdot\boldsymbol{x}_t \ln\beta - \ln \Sigma_{i=1}^N w_{t,i}\beta^{x_{t,i}}$. Let $B_t = \min_i x_{t,i}$, so $B_t \leq x_{t,i} \leq B_t + R$ holds for all i. For $\alpha \geq 0$ and $0 \leq z \leq 1$ we have the bound $\alpha^z \leq 1 - z(1-\alpha)$. Applying this bound with $\alpha = \beta^R$ and $z = (x_{t,i} - B_t)/R$ gives us

$$\begin{aligned} &d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w}_t) - d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w}_{t+1}) \\ &\quad\geq \boldsymbol{u}\cdot\boldsymbol{x}_t\ln\beta - B_t\ln\beta \\ &\qquad - \ln\left(1 - \frac{\boldsymbol{w}_t\cdot\boldsymbol{x}_t - B_t}{R}\,(1-\beta^R)\right). \end{aligned}$$

Hence, to prove (23) it suffices to show $F(y_t,\boldsymbol{w}_t\cdot\boldsymbol{x}_t,\boldsymbol{u}\cdot\boldsymbol{x}_t,\eta) \leq 0$ where

$$\begin{aligned} F(y,p,r,\eta) &= (\eta - \eta^2R^2Z/4)L_\phi(y,\phi(p)) - \eta L_\phi(y,\phi(r)) \\ &\quad - r\eta(y-\phi(p)) + B_t\eta(y-\phi(p)) \\ &\quad + \ln\left(1 - \frac{p-B_t}{R}\,(1-e^{R\eta(y-\phi(p))})\right). \end{aligned}$$

We then have

$$\frac{\partial F(y,p,r,\eta)}{\partial r} = \eta(y-\phi(r)) - \eta(y-\phi(p)) = \eta(\phi(p)-\phi(r))$$

and

$$\frac{\partial^2 F(y,p,r,\eta)}{\partial r^2} = -\eta\phi'(r) \leq 0.$$

Hence, the value $F(y,p,r,\eta)$ as a function of r is maximized when $r = p$. Therefore, it is sufficient to show $G(y,p,\eta) \leq 0$ where

$$\begin{aligned} G(y,p,\eta) &= F(y,p,p,\eta)/\eta \\ &= -\frac{\eta R^2Z}{4}\,L_\phi(y,\phi(p)) - (p-B_t)(y-\phi(p)) \\ &\quad + \frac{1}{\eta}\,\ln\left(1 - \frac{p-B_t}{R}\,(1-e^{\eta R(y-\phi(p))})\right). \end{aligned}$$

By applying (22) we get

$$\begin{aligned} \frac{\partial G(y,p,\eta)}{\partial y} &= -\frac{\eta R^2Z}{4}\,(\phi^{-1}(y)-p) - (p-B_t) \\ &\quad + \frac{(p-B_t)e^{\eta R(y-\phi(p))}}{1-(p-B_t)(1-e^{\eta R(y-\phi(p))})/R}. \end{aligned}$$

Hence, for $y = \phi(p)$ we have both $G(y,p,\eta) = 0$ and $\partial G(y,p,\eta)/\partial y = 0$, so to prove that $G(y,p,\eta) \leq 0$ always holds, it suffices to show that the second derivative

$\partial^2 G(y,p,\eta)/\partial y^2$ is always nonpositive. We can write

$$\begin{aligned}\frac{\partial^2 G(y,p,\eta)}{\partial y^2} &= -\frac{\eta R^2 Z}{4\phi'(\phi^{-1}(y))} + \frac{\eta R(p-B_t)e^{\eta R(y-\phi(p))}}{1-(p-B_t)(1-e^{\eta R(y-\phi(p))})/R} \\ &\quad - \frac{\eta(p-B_t)^2(e^{\eta R(y-\phi(p))})^2}{(1-(p-B_t)(1-e^{\eta R(y-\phi(p))})/R)^2} \\ &= \frac{-\eta R^2 Z/4}{\phi'(\phi^{-1}(y))} + \eta R^2(Q-Q^2)\end{aligned}$$

where

$$Q = \frac{(p-B_t)e^{\eta R(y-\phi(p))}}{R-(p-B_t)(1-e^{\eta(y-\phi(p))})}.$$

Since $Q - Q^2 \le 1/4$ holds for all real Q, and $Z \ge \phi'(\phi^{-1}(p)) > 0$, we have $\partial^2 G(y,p,\eta)/\partial y^2 \le 0$ as desired. □

Theorem 7: Let EG be the exponentiated gradient algorithm for transfer function ϕ and loss function L_ϕ, with start vector $\boldsymbol{w}_1 \in P_{N-1}$ and learning rate η. Consider a trial sequence $S = ((\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_\ell, y_\ell))$ such that for some $R > 0$ we have $\max_i x_{t,i} - \min_i x_{t,i} \le R$ for $1 \le t \le \ell$. If we choose the learning rate $\eta = 1/(R^2 Z)$ then for any choice of start vector $\boldsymbol{w}_1 \in P_{N-1}$ and all $\boldsymbol{u} \in P_{N-1}$ we have

$$\text{Loss}_\phi(\text{EG}) \le \tfrac{4}{3}\left(\text{Loss}_\phi(\boldsymbol{u},S) + R^2 Z d_{\text{RE}}(\boldsymbol{u},\boldsymbol{w}_1)\right). \quad (24)$$

Furthermore, for arbitrary positive constants K and D, if we choose the learning rate $\eta = 4(\sqrt{z^2+z} - z)/(R^2 Z)$ where $z = R^2 ZD/(4K)$, then for all $\boldsymbol{u} \in P_{N-1}$ such that $\text{Loss}_\phi(\boldsymbol{u},S) \le K$ and $\|\boldsymbol{u} - \boldsymbol{w}_1\|_2 \le U$ hold we have

$$\text{Loss}_\phi(\text{EG}) \le \text{Loss}_\phi(\boldsymbol{u},S) + R\sqrt{KZD} + R^2 ZD. \quad (25)$$

Proof: By Lemma 6, we can apply Lemma 3 with $\alpha = 3/4 = 1 - \eta R^2 Z/4$ and $\gamma = \eta$ to get the bound (24).

We now show (25). From Lemmas 6 and 3 with $\alpha = 1 - \eta R^2 Z/4$ and $\gamma = \eta$ we have

$$\begin{aligned}\text{Loss}(\text{EG},S) &\le \frac{\text{Loss}_\phi(\boldsymbol{u},S)}{1-\eta R^2 Z/4} + \frac{D}{\eta(1-\eta R^2 Z/4)} \\ &\le \text{Loss}_\phi(\boldsymbol{u},S) + \frac{\eta K R^2 Z/4}{1-\eta R^2 Z/4} \\ &\quad + \frac{R^2 ZD/4}{(1-\eta R^2 Z/4)\eta R^2 Z/4} \\ &= \text{Loss}_\phi(\boldsymbol{u},S) + K\left(\frac{\eta R^2 Z/4}{1-\eta R^2 Z/4}\right. \\ &\quad \left. + \frac{R^2 ZD/4}{K(1-\eta R^2 Z/4)\eta R^2 Z/4}\right).\end{aligned}$$

Since $\eta = 4(\sqrt{z^2+z} - z)/(R^2 Z)$ with $z = R^2 ZD/(4K)$, the term multiplying K is in the form required by Lemma 4. Applying this lemma, we get

$$\begin{aligned}\text{Loss}(\text{EG},S) &\le \text{Loss}_\phi(\boldsymbol{u},S) + 2K\sqrt{z} + 4Kz \\ &\le \text{Loss}_\phi(\boldsymbol{u},S) + R\sqrt{DZK} + R^2 ZD.\end{aligned}$$

as desired. □

Proof of Theorem 2: We use the usual reduction, where each component of the input, x_i is associated with two weights, w_i^+ and w_i^- and the effective weight of the component is $U(w_i^+ - w_i^-)$. This theorem then follows directly from Theorem 7 with $R = 2UX$. □

ACKNOWLEDGMENT

The authors wish to thank Mark Herbster for helpful discussions.

REFERENCES

[1] P. Auer, M. Herbster, and M. K. Warmuth, "Exponentially many local minima for single neurons," in *Advances in Neural Information Processing Systems 8*. Cambridge, MA: MIT Press, 1996, pp. 316–317.

[2] L. M. Bregman, "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming," *USSR Comput. Math. Phys.*, vol. 7, pp. 200–217, 1967.

[3] M. Budinich, "Some notes on perceptron learning," *J. Phys. A.: Math. Gen.*, vol. 26, pp. 4237–4247, 1993.

[4] M. Budinich and E. Milotti, "Geometrical interpretation of the back-propagation algorithm for the perceptron," *Phys. A*, vol. 185, pp. 369–377, 1992.

[5] N. Cesa-Bianchi, "Analysis of two gradient-based algorithms for on-line regression," in *Proc. 10th Annu. Wkshp Comput. Learning Theory*. New York: ACM, 1997, pp. 163–170.

[6] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth, "How to use expert advice," *J. ACM*, vol. 44, pp. 427–485, 1997.

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

[8] I. Csiszar, "Why least squares and maximum entropy? An axiomatic approach for linear inverse problems," *Ann. Statist.*, vol. 19, pp. 2032–2066, 1991.

[9] C. Gentile and M. K. Warmuth, "Linear hinge loss and average margin," in *Advances in Neural Information Processing Systems 11*. Cambridge, MA: MIT Press, 1999, pp. 225–231.

[10] A. J. Grove, N. Littlestone, and D. Schuurmans, "General convergence results for linear discriminant updates," in *Proc. 10th Annu. Conf. Comput. Learning Theory*. New York: ACM, 1997, pp. 171–183.

[11] D. Helmbold, R. E. Schapire, Y. Singer, and M. K. Warmuth, "A comparison of new and old algorithms for a mixture estimation problem," *Machine Learning*, vol. 27, pp. 97–119, 1997.

[12] D. Helmbold and M. K. Warmuth, "On weak learning," *J. Comput. Syst. Sci.*, vol. 50, pp. 551–573, 1995.

[13] J. Kivinen and M. K. Warmuth, "Additive versus exponentiated gradient updates for linear prediction," *Inform. Comput.*, vol. 132, pp. 1–64, 1997.

[14] ——, "Relative loss bounds for multidimensional regression problems," in *Advances in Neural Information Processing Systems 10*. Cambridge, MA: MIT Press, 1998, pp. 287–293.

[15] N. Littlestone, "Learning when irrelevant attributes abound: A new linear-threshold algorithm," *Machine Learning*, vol. 2, pp. 285–318, 1988.

[16] ——, "From on-line to batch learning," in *Proc. 2nd Annu. Wkshp. Comput. Learning Theory*. San Mateo, CA: Morgan Kaufmann, 1989, pp. 269–284.

[17] ——, "Mistake bounds and logarithmic linear-threshold learning algorithms," Ph.D. dissertation, University of California, Santa Cruz, Tech. Rep. UCSC-CRL-89-11, 1989.

[18] D. G. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984.

[19] S. A. Solla, E. Levin, and M. Fleisher, "Accelerated learning in layered neural networks," *Complex Syst.*, vol. 2, pp. 625–639, 1988.

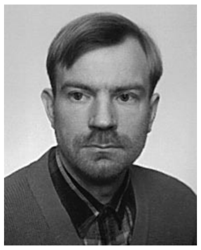
[20] V. Vovk, "Competitive on-line linear regression," in *Advances in Neural Information Processing Systems 10*. Cambridge, MA: MIT Press, 1998, pp. 364–370.

[21] K. Yamanishi, "On-line maximum likelihood prediction with respect to general loss functions," *J. Comput. Syst. Sci.*, vol. 55, pp. 105–118, 1997.

[22] ——, "Minimax relative loss analysis for sequential prediction algorithms using parametric hypotheses," in *Proc. 11th Annu. Conf. Comput. Learning Theory*. New York: ACM, 1998, pp. 32–43.

David P. Helmbold received the B.A. degree from the University of California, Berkeley, in 1981 and the Ph.D. degree in computer science from Stanford University, Stanford, CA, in 1987.

He is an Associate Professor at the University of California, Santa Cruz. His main research interests include computational learning theory, and he is currently working on on-line learning problems and boosting.



Jyrki Kivinen received the M.Sc. and Ph.D. degrees in computer science from University of Helsinki, Finland, in 1989 and 1992, respectively.

Since then he has held various research and teaching positions at University of Helsinki. His current research interests include machine learning, data mining, and neural networks.

Manfred K. Warmuth received the Ph.D. degree in computer science in 1981 from the University of Colorado, Boulder.

He is a Professor in the Computer Science Department at the University of California, Santa Cruz. His current research interests include machine learning, on-line learning, neural networks, and statistics.

Dr. Warmuth is an editor of the *Journal of Machine Learning* and is involved with the organization of the annual ACM Conference on Computational Learning Theory.