Additive Versus Exponentiated Gradient Updates for Linear Prediction

Jyrki Kivinen*
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
jkivinen@cs.helsinki.fi

Manfred K. Warmuth†
Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064, USA
manfred@cse.ucsc.edu

Abstract

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 weight vector is updated by subtracting from it the gradient of the squared error made on a prediction multiplied by a parameter called the learning rate. The EG$^{\pm}$ uses the components of the gradient in the exponents of factors that are used in updating the weight vector multiplicatively. We present worst-case on-line loss bounds for EG$^{\pm}$ and compare them to previously known bounds for the GD algorithm. The bounds suggest that although the on-line losses of the algorithms are in general incomparable, EG$^{\pm}$ has a much smaller loss if only few of the input variables are relevant for the predictions. Experiments show that the worst-case upper bounds are quite tight already on simple artificial data. Our main methodological idea is using a distance function between weight vectors both in motivating the algorithms and as a potential function in an amortized analysis that leads to worst-case loss bounds. Using squared Euclidean distance leads to the GD algorithm, and using the relative entropy leads to the EG$^{\pm}$ algorithm.

1 Introduction

The goal of this research is to obtain strong worst-case loss bounds for simple on-line algorithms by using amortized analysis. Consider the problem of on-line learning of linear threshold functions. There are two simple algorithms for this problem: the classical Perceptron algorithm [Ros58] and the more recent Winnow developed by Littlestone [Lit88]. The algorithms are similar in that they maintain one weight per dimension and use a linear threshold function to predict. However, the updates of their weights are radically different. Of particular interest to us is the class of (monotone) k-literal disjunctions over N variables, where we expect N to be significantly larger than k. This class is a simple subclass of linear threshold functions: if the instances $\mathbf{x}$ are N-dimensional Boolean vectors, i.e., $\mathbf{x} \in \{0,1\}^N$, then the k-literal disjunction $x_{i_1} \vee x_{i_2} \vee \ldots \vee x_{i_k}$ corresponds to the linear threshold function $\mathbf{w} \cdot \mathbf{x} \geq \theta$, where $\mathbf{w}$ is a coefficient vector with $w_{i_1} = w_{i_2} = \cdots = w_{i_k} = 1$ and $w_j = 0$ for $j \notin \{i_1, \ldots, i_k\}$, and the threshold θ is 1.

The on-line algorithm Winnow is guaranteed to make at most $O(k + k\log(N/k))$ prediction mistakes [Lit88] when learning k-literal disjunctions. This mistake bound is optimal to within a constant factor since the Vapnik-Chervonenkis (VC) dimension [VC71, BEHW89] of the class of k-literal disjunctions is $\Omega(k + k\log(N/k))$ [Lit88] and this dimension is always a lower bound for the optimal mistake bound. On the other hand, an adversary can force the Perceptron algorithm to make $\Omega(N-k)$ mistakes [KW95]. Thus when k is small, the mistake bound of the Perceptron algorithm is exponential in the optimal mistake bound (in this case essentially the VC dimension). This may be seen as an instance of what is called the *curse of dimensionality* in the literature of neural networks and statistics [Hay93, DH73].

To understand the different behavior of both algorithms, we retreat to the simpler problem of linear regression. Now the algorithms simply predict with the scalar product of the weight vector and the input vectors rather than with a classification obtained by thresholding the scalar product. Again, there are two fundamentally different algorithms. The first algorithm is the classical *Gradient Descent* algorithm (GD), in this context also known as the Widrow-Hoff rule and Least Mean Squares algorithm [WS85, Hay93]. It can be motivated as a gradient descent on the instantaneous loss of the algorithm and corresponds to the Perceptron algorithm. In this paper we introduce several new algorithms that apply the basic ideas of Winnow to the linear regression problem. The most basic one is the *Exponentiated Gradient* (EG) algorithm, from which we obtain the main algorithm called EG$^{\pm}$ by a simple reduction. We prove worst-case loss bounds for both the GD and EG$^{\pm}$ algorithms using amortized analysis. This has already been done for GD [CBLW93]. The focus in this paper is to prove similar bounds for the new algorithm EG$^{\pm}$ and then contrast the bounds and algorithms.

Since the regression problem is continuous in nature, we can see connections that are less apparent in the thresholded classification problem. The first key observation we make is that both the GD and EG$^{\pm}$ algorithms can be motivated as approximate solutions to a minimization problem that has as one component the distance between the new and

*Supported by Emil Aaltonen Foundation and the Academy of Finland. This work was done while the author was visiting University of California, Santa Cruz.
†Supported by NSF grant IRI-9123692.

STOC '95, Las Vegas, Nevada, USA
© 1995 ACM 0-89791-718-9/95/0005..$3.50

old hypothesis of the algorithm. The solution depends on the choice of the distance function between the hypotheses, which in our case are N-dimensional real weight vectors. Using the squared Euclidean distance leads to the GD algorithm, and using the relative entropy leads to the EG$^{\pm}$ algorithm. The second key observation is that the distance function that motivates the update of the weight vector gets a second use as a potential function for the amortized analysis of the learning algorithm employing that update. In the full paper we derive a number of additional algorithms using this framework and prove loss bounds for them using similar techniques [KW94].

As with the Perceptron algorithm and Winnow, the performances of the GD and EG$^{\pm}$ algorithms are radically different, the EG$^{\pm}$ algorithm performing better when only few of the input variables are relevant. As a simple example, assume that the value to be predicted is the sum of k variables, out of a total of N variables, and the variables assume only values -1 and 1. This is the continuous-valued analogue of k-literal disjunctions. The total squared error of the predictions made by the EG$^{\pm}$ algorithm is $O(k^2 \log N)$. The GD algorithm can incur a total squared error of $\Omega(kN)$, which is much larger when $N \gg k$. Note again the logarithmic versus the linear growth in the input dimension N. Our bounds are worst-case bounds, but our experiments show that the worst-case bounds are quite tight even on simple random data. The bounds can be generalized to situations in which no linear combination of the input variables can explain the values to be predicted, either because of noise or because of nonlinear dependencies.

The general forms of the loss bounds for the algorithms are rather complicated, so in the introduction we only consider the basic ideas of the bounds. Consider a situation in which the learner is given instances $\mathbf{x}_t$, $t = 1, 2, \ldots$, and tries to predict outcomes y_t. Assume now that some coefficient vector $\mathbf{u}$ is a good predictor, i.e., the scalar product $\mathbf{u} \cdot \mathbf{x}_t$ in some sense tends to be close to the outcome y_t. Our loss bounds for both algorithms can now be stated in terms of the norm $||\mathbf{u}||$ of the good predictor and the largest norm $\max_t ||\mathbf{x}_t||$ of the instances. However, for different algorithms the bounds depend on different norms. The bounds for GD depend on a product $X_2 U_2$, where X_2 is an upper bound for the largest Euclidean norm $\max_t ||\mathbf{x}_t||_2$ of the instances and U_2 is an upper bound for the Euclidean norm $||\mathbf{u}||_2$. For the EG$^{\pm}$ algorithm, the bounds depend on the product $X_\infty U_1$, where X_∞ and U_1 are upper bounds for the L_∞ and L_1 norms, respectively. As the products $X_2 U_2$ and $X_\infty U_1$ are incomparable, we expect that also the performances of the algorithms are incomparable. Indeed, it is easy to construct cases in which the GD algorithm outperforms the EG$^{\pm}$ algorithm. However, we feel that these are less natural than the cases that favor EG$^{\pm}$ over GD.

Note that the above pairs of norms corresponding to both algorithms are dual norms [Roy63]. We can give a lower bound that contains the product of an arbitrary pair of dual norms. We conjecture that there are regression algorithms for arbitrary pairs of dual norms with loss bounds close to our lower bounds. Finding these algorithms and the distance functions that motivate them within our framework is one of the key open problems. However, we believe that the special cases discussed here for the pairs (L_2, L_2) and (L_∞, L_1) are the fundamental ones.

We were surprised to find that the method of amortized analyses can be such a powerful tool in the comparative study of simple machine learning algorithms. We have successfully applied our framework to an unsupervised learning problem [HSSW95] and to temporal difference learning [SW94]. In each case our framework can be used to derive the algorithms as well as proving worst-case loss bounds for them. Whenever there is an algorithm based on Gradient Descent our method leads to a competitor that is derived using the relative entropy. For example, our method gives an alternative to the back-propagation algorithm for training feed-forward neural networks. Whenever there are local minima, it seems impossible to prove worst-case upper bounds. However, we have some preliminary results [HKW95] that show that our amortized analysis technique can give worst-case loss bounds for a single linear neuron with a sigmoid activation function. The latter problem does not have local minima if we use the entropic loss instead of the square loss.

We are looking for high dimensional natural problems that would bring out the advantages of the new algorithms. The logarithmic growth of the loss bounds with the dimension makes the following approach feasible. Try a large set of basis functions (non-linear functions of the original inputs) and do a pass over the data with EG$^{\pm}$ training one weight per basis function. Then exchange basis functions which have small remaining weight with new guesses of good basis functions and iterate. The new basis functions could, for example, be derived from basis functions that received a large weight in the last iteration. At the end of the paper we give an experimental comparison of the two competing algorithms for a case when the instances are expanded to a large set of basis functions and the target uses only a few of them.

We propose a systematic study of on-line learning algorithms using amortized analysis. The learning algorithms should be derived and analyzed using as few parameters as possible. We propose to use the potential function as the main parameter which encodes a way of searching through the solution space. The goal is to explain most simple learning algorithms by simply changing the potential function. For example, for a simple mixture problem [HSSW95] we were able to reverse-engineer the potential function deriving the algorithm corresponding to another statistical technique called Estimate Maximize. Going to a continuous domain led to the development of a unifying framework which lets us see new connections. It seems that all basic algorithms are incomparable and in the linear case our loss bounds can be used to predict which algorithm will do best.

The full version of this paper [KW94] contains additional theoretical and experimental results.

2 Framework of analysis

A *linear on-line prediction algorithm* is an algorithm that predicts real outputs from N-dimensional real inputs according to a certain protocol. In this protocol, the predictions are made in a sequence of ℓ *trials* for some positive integer ℓ. At the beginning of the trial sequence, the algorithm chooses some *start vector* $\mathbf{w}_1 \in \mathbf{R}^N$. Then at trial t, for $t = 1, \ldots, \ell$, the algorithm

1. receives the *instance* $\mathbf{x}_t \in \mathbf{R}^N$,
2. gives its *prediction* $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$,
3. receives an *outcome* y_t, and
4. chooses for the next trial the vector $\mathbf{w}_{t+1} \in \mathbf{R}^N$ that may depend on $\mathbf{w}_t$, $\mathbf{x}_t$, and y_t.

Thus, the weight vector $\mathbf{w}_t$ can be considered the algorithm's hypothesis before trial t for the best way of predicting the outcome as a linear combination of the components of the instance. The mapping that gives $\mathbf{w}_{t+1}$ as a function of $\mathbf{w}_t$, $\mathbf{x}_t$, and y_t is called the *update rule* of the algorithm.

For the purpose of the worst-case analysis we are interested in, the on-line prediction protocol can be considered a game between the prediction algorithm and an adversarial environment. The algorithm may choose its start vector and update rule, the enviroment may choose the instances and outcomes subject to some mild constraints. For scoring the game, we use a *loss function* L that is a mapping from $\mathbf{R}^2$ to $[0,\infty)$. In a trial sequence $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$, the total loss of the algorithm is defined to be $\mathrm{Loss}_L(A,S) = \sum_{t=1}^{\ell} L(y_t, \hat{y}_t)$. Analogously, the total loss of a fixed weight vector $\mathbf{u}$ is given 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 if the loss function is the square loss given by $L(y,z) = (y-z)^2$, which is the case for the main results of this paper. For the square loss, the value $\inf_{\mathbf{u}\in\mathbf{R}^N} \mathrm{Loss}(\mathbf{u},S)$ is the total loss of the solution to the linear least squares problem defined by the instances and outcomes. More generally, we are interested comparing the loss of the algorithm to the value $\inf_{\mathbf{u}\in\mathcal{U}} \mathrm{Loss}_L(\mathbf{u},S)$ for some *comparison class* $\mathcal{U} \subseteq \mathbf{R}^N$. We sometimes refer to the vectors $\mathbf{u} \in \mathcal{U}$ as *comparison vectors*. The value $\inf_{\mathbf{u}\in\mathcal{U}} \mathrm{Loss}_L(\mathbf{u},S)$ is the lowest loss that could be achieved if it were possible to choose in advance the best comparison vector for predicting. The algorithm cannot see the whole sequence in advance, and therefore incurs an *additional loss* $\mathrm{Loss}_L(A,S) - \inf_{\mathbf{u}\in\mathcal{U}} \mathrm{Loss}_L(\mathbf{u},S)$ as it searches for the best vector $\mathbf{u}$ and makes on-line predictions while it searches.

Our goal is to obtain a small additional loss for interesting comparison classes when the trial sequence satisfies some mild constraints. Recall that for $p \in (0,\infty)$, the L_p norm for vectors in $\mathbf{R}^N$ is defined by $||\mathbf{w}||_p = \left(\sum_{i=1}^N |w_i|^p\right)^{1/p}$; this is generalized for $p=\infty$ by $||\mathbf{w}||_\infty = \max_i |w_i|$. We take the comparison class $\mathcal{U}$ to consist of all the vector $\mathbf{u} \in \mathbf{R}^N$ that have L_p norm at most U, for some p and U. Similarly, we assume that the instances $\mathbf{x}_t$ have a bounded L_q norm for some q. We consider the case $p = q = 2$ and the case $p = 1$, $q = \infty$. These pairs of norms satisfy $1/p + 1/q = 1$ and are hence *dual* [Roy63].

In addition to norms, we use the *relative entropy* (also known as Kullback-Leibler distance)

$$d_{\mathrm{RE}}(\mathbf{u},\mathbf{s}) = \sum_{i=1}^{N} u_i \ln \frac{u_i}{s_i} \tag{1}$$

between vectors $\mathbf{u}$ and $\mathbf{s}$. Using the relative entropy requires that the vectors have only positive components, and the components of each vector sum to 1.

Subject to the norm constraint, the instances can be arbitrary. We make no assumptions about the process that generates them. Neither do we make assumptions about how the individual outcomes and instances relate to each other. If there is no reasonably accurate linear relationship between the instances $\mathbf{x}_t$ and the outcomes y_t, then the loss $\mathrm{Loss}_L(A,S)$ of the algorithm is expected to be high. However, in that case the loss $\inf_{\mathbf{u}\in\mathcal{U}} \mathrm{Loss}_L(\mathbf{u},S)$ of the best comparison vector will be high as well, so it is still feasible to try to achieve a small additional loss $\mathrm{Loss}_L(A,S) - \inf_{\mathbf{u}\in\mathcal{U}} \mathrm{Loss}_L(\mathbf{u},S)$. 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 applied in deriving probabilistic loss bounds for the prediction algorithm [Hay93, WS85].

The research reported in this paper was inspired by Littlestone [Lit88, Lit89], who proved worst-case on-line loss bounds for the case of thresholded linear functions. Recently, there has been some work on the special case where the comparison class consists of the N unit vectors $(0,\ldots,0,1,0,\ldots,0)$ [LW94, Vov90, CBFH+95, HKW94]. The immediate predecessors of this work are the papers by Cesa-Bianchi et al. [CBLW93] and Littlestone et al. [LLW91] on linear on-line prediction.

3 The algorithms

For simplicity, we introduce the algorithms only for the square loss, although they can easily be generalized. In all the algorithms we consider here, the update is based on the gradient of the loss at trial t. We define for this gradient a shorthand notation by

$$\nabla_t = \nabla_{\mathbf{w}_t} L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t) \ . \tag{2}$$

For the square loss we therefore have

$$\nabla_{t,i} = \frac{\partial (y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2}{\partial w_{t,i}} = 2(\hat{y}_t - y_t) x_{t,i} \tag{3}$$

where $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$. Notice that for any loss function, the vector ∇_t is parallel to the instance $\mathbf{x}_t$. The other basic ingredient of the updates we consider is a positive *learning rate* η_t, which typically depends on some norm of the instance $\mathbf{x}_t$ and some parameters fixed at the beginning of the trial sequence.

One of the simplest learning algorithm is the Gradient Descent (GD) algorithm for linear prediction, also known as the Widrow-Hoff algorithm and the Least Mean Squares algorithm. Using the gradient ∇_t given in (3), the update of the GD algorithm can be written as $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_t$, or componentwise as

$$w_{t+1,i} = w_{t,i} - \eta_t \nabla_{t,i} \ . \tag{4}$$

Thus, the weight vector moves to the direction of the steepest descent of the loss by an amount that depends on the learning rate. Note that if the initial weight vector of the algorithm is $\mathbf{w}_1$, we have $\mathbf{w}_m = \mathbf{w}_1 + \sum_{t=1}^{m-1} a_t \mathbf{x}_t$ for some real coefficients a_t. Typically, one could choose the zero start vector $\mathbf{w}_1 = \mathbf{0}$. The choice of the learning rates η_t can have a great effect to the performance of the algorithm. We shall discuss this in the context of our upper bounds in Section 5. A simple example could be $\eta_t = 1/(4||\mathbf{x}_t||_2)$.

In this paper we contrast the GD algorithm with a new on-line prediction algorithm, which we call the *Exponentiated Gradient* (EG) algorithm. This basic algorithm uses weight vectors $\mathbf{w}_t$ for which $w_{t,i} > 0$ for all i and $\sum_{i=1}^N w_{t,i} = 1$. After the basic EG algorithm we consider variants without these restrictions, most importantly one which we call EG$^{\pm}$. The EG algorithm is closely related to the algorithm of Littlestone et al. [LLW91]. The update rule of the EG algorithm is given by

$$w_{t+1,i} = \frac{w_{t,i} e^{-\eta_t \nabla_{t,i}}}{\sum_{j=1}^N w_{t,j} e^{-\eta_t \nabla_{t,j}}} \ . \tag{5}$$

Thus, the ith component $\nabla_{t,i}$ of the gradient defined in (2) now appears in the exponent of the factor that multiplies $w_{t,i}$, and the weights are normalized to sum to 1.

A typical start vector could be the uniform vector $\mathbf{w}_1 = (1/N, \ldots, 1/N)$. Again, the learning rates η_t shall be discussed in Section 5. A very basic setting would be $\eta_t = 2/(3R_t)$ where $R_t = \max_i x_{t,i} - \min_i x_{t,i}$ gives the range of the components of the tth instance.

The components $w_{m,i}$ of the weight vector $\mathbf{w}_m$ satisfy $\ln w_{m,i} = \ln w_{1,i} + \sum_{t=1}^{m-1} a_t x_{t,i} - b$, where the a_t are again some real coefficient as for the GD algorithm and the value b gives the proper normalization. Hence, the EG algorithm is similar to applying the GD algorithm to the logarithms of the weights and maintaining the normalization of the weights. However even if we ignore the normalization, the EG algorithm is not the same as gradient descent on the logarithmically parameterized weights, since the predictions are made using the weights and not their logarithms. Another comparison between the algorithms can be made by observing that for the GD algorithm, the change caused by the update in the differences between the weights can be written as $w_{t+1,i} - w_{t+1,j} = (w_{t,i} - w_{t,j}) - \eta_t(\nabla_{t,i} - \nabla_{t,j})$. For the EG algorithm, it is more useful to consider the ratios of the weights and write $w_{t+1,i}/w_{t+1,j} = (w_{t,i}/w_{t,j})e^{-\eta_t(\nabla_{t,i} - \nabla_{t,j})}$.

The EG algorithm maintains the normalization property $\sum_{i=1}^{N} w_{t,i} = 1$ and, in addition, keeps all the weights $w_{t,i}$ positive provided that they are positive initially. These constraints are of course inappropriate in many situations. To allow the algorithm to use unnormalized weight vectors, one can simply omit the normalization factor from the update rule (5) and apply the update $w_{t+1,i} = w_{t,i}e^{-\eta_t \nabla_{t,i}}$. This leads to a quite reasonable algorithm, called EGU, that is analyzed in the full version of the paper [KW94] and is closely related to the algorithm Winnow of [Lit88]. However, even this unnormalized algorithm never changes the sign of a weight. Therefore, we introduce an algorithm which we call EG$^{\pm}$. This algorithm is basically EG with a simple standard transformation for the instances.

To understand the idea of EG$^{\pm}$, consider a vector $\mathbf{u} \in \mathbf{R}^N$ with $||\mathbf{u}||_1 \leq U$ for some given parameter $U > 0$. For $i = 1, \ldots, N$, let $u_i^+ = u_i$ if $u_i > 0$ and $u_i^+ = 0$ otherwise, and $u_i^- = -u_i$ if $u_i < 0$ and $u_i^- = 0$ otherwise. Define a vector $\mathbf{u}' \in [0, U]^{2N}$ by setting $u'_i = u_i^+ + (U - ||\mathbf{u}||_1)/(2N)$ and $u'_{N+i} = u_i^- + (U - ||\mathbf{u}||_1)/(2N)$ for $i = 1, \ldots, N$. We call this $\mathbf{u}'$ the *norm U representation* for $\mathbf{u}$. If we now define $\mathbf{u}'' = \mathbf{u}'/U$, we have $u''_i \geq 0$ for all i and $||\mathbf{u}''||_1 = 1$, so $\mathbf{u}''$ is in the class of possible hypotheses for EG. Further, if for $\mathbf{x} \in \mathbf{R}^N$ we define $\mathbf{x}' \in \mathbf{R}^{2N}$ by $x'_i = Ux_i$ and $x'_{N+i} = -Ux_i$ for $i = 1, \ldots, N$, we have $\mathbf{u} \cdot \mathbf{x} = \mathbf{u}'' \cdot \mathbf{x}'$ for all $\mathbf{x}$.

We now define EG$^{\pm}$ to be the algorithm that at trial t

1. receives the tth instance $\mathbf{x}_t$ and gives the vector $\mathbf{x}'_t = (Ux_{t,1}, \ldots, Ux_{t,N}, Ux_{t,1}, \ldots, Ux_{t,2N})$ as the tth instance to the EG algorithm,
2. gives as its tth prediction $\hat{y}_t$ the tth prediction of the EG algorithm (i.e. $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}'_t$, where $\mathbf{w}_t$ is the current weight vector of EG), and
3. receives the tth outcome y_t and gives it as the tth outcome to the EG algorithm which updates its current weight vector $\mathbf{w}_t$.

Note that in addition to the start vector and learning rates for the EG algorithm, there is the additional parameter U that needs to be specified.

4 Motivating the algorithms

Both 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 $\mathbf{w}_t$, it is a reasonable goal for the algorithm to minimize

$$M(\mathbf{w}_{t+1}) = d(\mathbf{w}_{t+1}, \mathbf{w}_t) + \eta_t L(y_t, \mathbf{w}_{t+1} \cdot \mathbf{x}_t) \ ,$$

where $d(\mathbf{w}_{t+1}, \mathbf{w}_t)$ 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_t represents the importance of correctiveness compared to the importance of conservativeness. The minimization problem would be solved by setting $\partial M(\mathbf{w}_{t+1})/\partial w_{t+1,i} = 0$ for all i or, equivalently, by setting

$$\frac{\partial d(\mathbf{w}_{t+1}, \mathbf{w}_t)}{\partial w_{t+1,i}} + \eta_t x_{t,i} \left(\frac{\partial L(y_t, z)}{\partial z} \right)_{z = \mathbf{w}_{t+1} \cdot \mathbf{x}_t} = 0 \qquad (6)$$

for all i. Solving the equations (6) for $w_{t+1,i}$ may be difficult, but it becomes simpler if we approximate the left-hand side by evaluating the derivative of $L(y_t, z)$ at $z = \mathbf{w}_t \cdot \mathbf{x}_t$ instead of at $z = \mathbf{w}_{t+1} \cdot \mathbf{x}_t$. Thus, we use the gradient evaluated at the old weight vector $\mathbf{w}_t$, defined as ∇_t in (2), to replace the gradient evaluated at the new, still unknown, weight vector $\mathbf{w}_{t+1}$. For the squared Euclidean distance $d(\mathbf{w}_{t+1}, \mathbf{w}_t) = \frac{1}{2}||\mathbf{w}_{t+1} - \mathbf{w}_t||_2^2$, solving the approximated equation easily gives the GD algorithm. The EG algorithm is obtained in a straightforward manner by using for $d(\mathbf{w}_{t+1}, \mathbf{w}_t)$ the relative entropy $d_{\mathrm{RE}}(\mathbf{w}_{t+1}, \mathbf{w}_t)$ defined in (1). The relative entropy assumes that all the components $w_{t,i}$ and $w_{t+1,i}$ are positive and satisfy $\sum_i w_{t+1,i} = \sum_i w_{t,i} = 1$. The positiveness of the weights follows directly from the update we obtain. The constraint $\sum_i w_{t+1,i} = 1$ is observed in the usual way by adding a term with a Lagrange multiplier to the function M to be minimized.

Note that neither the squared Euclidean distance nor the relative entropy is a metric. However, they do have the properties $d(\mathbf{w}, \mathbf{w}) = 0$ and $d(\mathbf{w}, \mathbf{s}) > 0$ for $\mathbf{s} \neq \mathbf{w}$.

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. Amari's [Ama94, Ama95] 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 use of a distance measure for obtaining worst-case loss bounds was pioneered by Littlestone's [Lit89] analysis of Winnow, which also employs a variant of the relative entropy. This idea is explained in more detail in Section 5.

5 Worst-case loss bounds

5.1 The basic proof method

In the most basic case, to prove a worst case loss bound for an on-line linear algorithm, we would define a potential function $V(\mathbf{w})$ for weight vectors. We call the potential difference $V(\mathbf{w}_t) - V(\mathbf{w}_{t+1})$ the *progress* made by the algorithm at trial t. We assume that the value $V(\mathbf{w})$ is nonnegative and bounded from above by some value V_0. If we can then prove

that for all possible combinations of an old weight vector $\mathbf{w}_t$, instance $\mathbf{x}_t$, and outcome y_t, the updated weight vector $\mathbf{w}_{t+1}$ of the algorithm A satisfies $L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t) \leq V(\mathbf{w}_t) - V(\mathbf{w}_{t+1})$, we obtain $\mathrm{Loss}_L(A,S) \leq V(\mathbf{w}_1) - V(\mathbf{w}_{\ell+1}) \leq V_0$. Thus, if the total amount of progress is bounded and at each trial the loss is bounded by the progress, we get a bound for the total loss.

Of course, we cannot define a potential function V for which the above holds in general, since the loss of the algorithm is expected to depend on the loss of the best comparison vector $\mathbf{u} \in \mathcal{U}$. In the special case that $\mathrm{Loss}_L(\mathbf{u},S) = 0$ holds for some $\mathbf{u} \in \mathcal{U}$, we could take $V(\mathbf{w}) = c\,d(\mathbf{u},\mathbf{w})$ where $c > 0$ and d is some distance measure. Since we use the potential function only for analyzing the algorithm, it does not matter that the vector $\mathbf{u}$ is not known in advance. Consider now the general case, when even the best comparison vector incurs some loss over the trial sequence. Our basic idea is to simultaneously consider $V(\mathbf{w}) = c\,d(\mathbf{u},\mathbf{w})$ for all comparison vectors $\mathbf{u} \in \mathcal{U}$. We call $d(\mathbf{u},\mathbf{w}_t) - d(\mathbf{u},\mathbf{w}_{t+1})$ the algorithm's progress towards $\mathbf{u}$ at trial t. Ideally, we would wish to have for all comparison vectors $\mathbf{u}$ the algorithm's additional loss compared with $\mathbf{u}$, given by $L(y_t, \mathbf{w}_t \cdot \mathbf{x}_t) - L(y_t, \mathbf{u} \cdot \mathbf{x}_t)$, bounded from above by its progress towards $\mathbf{u}$. This turns out to be not quite possible, but we can prove for all $\mathbf{u}$ the bound

$$aL(y_t, \mathbf{w}_t \cdot \mathbf{x}_t) - bL(y_t, \mathbf{u} \cdot \mathbf{x}_t) \leq d(\mathbf{u},\mathbf{w}_t) - d(\mathbf{u},\mathbf{w}_{t+1}) \quad (7)$$

where a and b are suitably chosen constants with $0 < a < b$. Hence, at trial t the algorithm makes progress towards $\mathbf{u}$ if $\mathbf{u}$ predicted at that trial notably better than the algorithm did. On the other hand, on any given trial some vectors $\mathbf{u}$ make a larger loss than the algorithm, and the algorithm can make even negative progress towards those $\mathbf{u}$.

To control the parameters a and b in (7) it turns out to be convenient to introduce a new parameter c and functions f and g such that for $b = g(c)$, the condition (7) holds for all $\mathbf{u}$ if and only if $a \leq f(c)$. Proving (7) for the optimal values $a = f(c)$ and $b = g(c)$ is the main technical problem in the analysis. When this has been achieved, adding the bounds (7) for $t = 1, \ldots, \ell$ with $a = f(c)$ and $b = g(c)$ yields

$$\begin{aligned} \mathrm{Loss}_L(A,S) &\leq \frac{g(c)}{f(c)}\mathrm{Loss}_L(\mathbf{u},S) + \frac{d(\mathbf{u},\mathbf{w}_1) - d(\mathbf{u},\mathbf{w}_{\ell+1})}{f(c)} \\ &\leq \frac{g(c)}{f(c)}\mathrm{Loss}_L(\mathbf{u},S) + \frac{d(\mathbf{u},\mathbf{w}_1)}{f(c)} \end{aligned} \quad (8)$$

for all $\mathbf{u} \in \mathcal{U}$. The bound for the algorithm compared to $\mathbf{u}$ is then obtained by choosing the value c for which the right-hand side of (8) is minimized. Since everything holds for an arbitrary $\mathbf{u} \in \mathcal{U}$ from the comparison class, the final upper bound is obtained by taking an infimum over $\mathbf{u} \in \mathcal{U}$.

5.2 Upper bounds for GD

In the full paper we show that the earlier results of Cesa-Bianchi et al. [CBLW93] for the GD algorithm fit well into our general framework for proving loss bounds. We cite here their main result for comparison with our results on the new algorithm EG$^{\pm}$.

Theorem 1 *For a trial sequence $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$, let X be an upper bound such that $||\mathbf{x}_t||_2 \leq X$ holds for all t. If the* GD *algorithm uses an arbitrary start vector $\mathbf{w}_1$ and learning rates given by $\eta_t = 1/(4||\mathbf{x}_t||_2^2)$, we have for any vector $\mathbf{u}$ the bound*

$$\mathrm{Loss}(\mathrm{GD},S) \leq 2\left(\mathrm{Loss}(\mathbf{u},S) + ||\mathbf{u} - \mathbf{w}_1||_2^2 X^2\right) . \quad (9)$$

Further, let K and U be arbitrary constants and define a comparison class by $\mathcal{U} = \{\mathbf{u} \in \mathbf{R}^N : \mathrm{Loss}(\mathbf{u},S) \leq K$ and $||\mathbf{u} - \mathbf{w}_1||_2 \leq U\}$. Now for the learning rate

$$\eta_t = \frac{1}{||\mathbf{x}_t||_2^2} \frac{UX}{2\sqrt{K} + 2UX} . \quad (10)$$

the GD *algorithm has for any $\mathbf{u} \in \mathcal{U}$ the loss bound*

$$\mathrm{Loss}(\mathrm{GD},S) \leq \mathrm{Loss}(\mathbf{u},S) + 2\sqrt{K}UX + ||\mathbf{u} - \mathbf{w}_1||_2^2 X^2 . \quad (11)$$

The bound (11) is a strong worst-case upper bound. The ratio $\mathrm{Loss}(\mathrm{GD},S)/\inf_{\mathbf{u}\in\mathcal{U}}\mathrm{Loss}(\mathbf{u},S)$ approaches 1 as $\inf_{\mathbf{u}\in\mathcal{U}}\mathrm{Loss}(\mathbf{u},S)$ approaches ∞, assuming the parameter K is $\inf_{\mathbf{u}\in\mathcal{U}}\mathrm{Loss}(\mathbf{u},S)$ (or somewhat higher). Further, we get good upper bounds for the two leading coefficients in the additional loss $\mathrm{Loss}(\mathrm{GD},S) - \inf_{\mathbf{u}\in\mathcal{U}}\mathrm{Loss}(\mathbf{u},S)$. Our lower bound in Theorem 5 implies that these constants are, in fact, optimal. The less refined bound (9) is obtained without prior knowledge of K or U.

5.3 Upper bounds for EG

The full papers contains many loss bounds for the algorithm EG and its variants. The main result is the following upper bound for the EG$^{\pm}$ algorithm.

Theorem 2 *Let $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$ be a trial sequence and X a bound such that $||\mathbf{x}_t||_\infty \leq X$ holds for all t. Let $\mathbf{u} \in \mathbf{R}^N$ be an arbitrary weight vector with $||\mathbf{u}||_1 \leq U$. If the algorithm* EG$^{\pm}$ *uses the uniform start vector and the learning rates given by $\eta_t = 1/(3U^2||\mathbf{x}_t||_\infty^2)$, we have*

$$\mathrm{Loss}(\mathrm{EG}^{\pm},S) \leq 3\left(\mathrm{Loss}(\mathbf{u},S) + U^2X^2\ln 2N\right) . \quad (12)$$

Further, let K and U be arbitrary constants and define a comparison class by $\mathcal{U} = \{\mathbf{u} \in \mathbf{R}^N : \mathrm{Loss}(\mathbf{u},S) \leq K$ and $||\mathbf{u}||_1 \leq U\}$. Now for the learning rate

$$\eta_t = \frac{1}{||\mathbf{x}_t||_\infty^2} \cdot \frac{X\sqrt{\ln 2N}}{U\sqrt{2K} + 2U^2X\sqrt{\ln 2N}} . \quad (13)$$

the EG$^{\pm}$ *algorithm has for any $\mathbf{u} \in \mathcal{U}$ the loss bound*

$$\begin{aligned} \mathrm{Loss}(\mathrm{EG}^{\pm},S) \leq\ & \mathrm{Loss}(\mathbf{u},S) + 2UX\sqrt{2K\ln 2N} \\ & + 2U^2X^2\ln 2N . \end{aligned} \quad (14)$$

Bounds similar to (12), but with worse constants, were already proved by Littlestone et al. [LLW91] for their algorithm. Here we have also the stronger bound (14), in which the ratio $\mathrm{Loss}(\mathrm{EG}^{\pm},S)/\inf_{\mathbf{u}\in\mathcal{U}}\mathrm{Loss}(\mathbf{u},S)$ approaches 1 as $\inf_{\mathbf{u}\in\mathcal{U}}\mathrm{Loss}(\mathbf{u},S)$ approaches ∞. Notice that as with the GD algorithm, making full use of the tighter bound (14) requires good values for the parameters K and U.

There are tighter and more complicated formulations of the bounds (12) and (14) that use the actual relative entropy $d_{\mathrm{RE}}(\mathbf{u}'/U, \mathbf{s})$ where $\mathbf{s} \in \mathbf{R}^{2N}$ is the start vector of the EG algorithm simulated by EG$^{\pm}$ and $\mathbf{u}' \in \mathbf{R}^{2N}$ the norm U representation of $\mathbf{u}$. The appearance of the factor $\ln 2N$ in the bounds is due to our use of the bound $\sum_{i=1}^{2N}(u_i'/U)\ln((u_i'/U)/s_i) \leq \ln 2N$ for the relative entropy for the uniform start vector $\mathbf{s} = (1/2N, \ldots, 1/2N)$.

The following lemma is the main step that leads to Theorem 2.

Lemma 3 *Let $\mathbf{w}_t$ be the weight vector of* EG *before trial t in a trial sequence $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$, and let $\mathbf{u} \in [0,1]^N$ be a vector with $\sum_i u_i = 1$. Consider an arbitrary trial t, and let $R = \max_i x_{t,i} - \min_i x_{t,i}$. For any constants a and b such that $0 < a \leq 2b/(2 + R^2 b)$, and the learning rate $\eta_t = 2b/(2 + R^2 b)$, we have*

$$\begin{aligned} & a(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - b(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \\ & \quad \leq \; d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_t) - d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_{t+1}) \; . \end{aligned} \tag{15}$$

Proof We have $w_{t+1,i} = w_{t,i}\beta_t^{x_{t,i}} / \sum_j w_{t,j}\beta_t^{x_{t,j}}$ where $\beta_t = e^{2\eta_t(y_t - \hat{y}_t)}$. We can then write

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

Hence, (15) can be written $F(\mathbf{w}_t, \mathbf{x}_t, \mathbf{w}_t \cdot \mathbf{x}_t, y_t, \mathbf{u} \cdot \mathbf{x}_t, \beta_t) \leq 0$ where

$$\begin{aligned} F_N(\mathbf{w}, \mathbf{x}, \hat{y}, y, r, \beta) \; &= \; \ln \sum_{i=1}^N w_i \beta^{x_i} - r \ln \beta \\ & \quad + a(y - \hat{y})^2 - b(y - r)^2 \end{aligned} \tag{16}$$

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

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

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

Let now $\sum_{i=1}^N w_i = 1$ and $\mathbf{w} \cdot \mathbf{x} = \hat{y}$. Using the above gives us

$$\ln \sum_{i=1}^N w_{t,i}\beta^{x_{t,i}} \leq B \ln \beta + \ln \left(1 - \frac{\hat{y} - B}{R}\left(1 - \beta^R\right)\right) \; .$$

Hence, we get $F_N(\mathbf{w}, \mathbf{x}, \hat{y}, y, r, \beta) \leq G(\hat{y}, y, r, \beta)$, where

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

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

To obtain (15), it is now sufficient to show that $G(\hat{y}, y, r, \beta) \leq 0$ holds for all values of $\hat{y}$, y, and r, when $\beta = e^{2\eta(y - \hat{y})}$ with $\eta = \eta_t = 2b/(2 + R^2 b)$. Since the second derivative $\partial^2 G(\hat{y}, y, r, \beta)/\partial r^2 = -2b$ is negative, 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 $G(\hat{y}, y, r, e^{2\eta(y - \hat{y})}) \leq 0$ for $r = y + \eta(\hat{y} - y)/b$ implies $G(\hat{y}, y, r, e^{2\eta(y - \hat{y})}) \leq 0$ 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

$$\begin{aligned} H(\hat{y}, y) \; &= \; 2\eta B(y - \hat{y}) \\ & \quad + \ln \left(1 - \frac{\hat{y} - B}{R}\left(1 - e^{2\eta R(y - \hat{y})}\right)\right) \\ & \quad - 2\eta y(y - \hat{y}) + \left(a + \frac{\eta^2}{b}\right)(y - \hat{y})^2 \; . \end{aligned}$$

It remains to show that $H(\hat{y}, y) \leq 0$. We apply the bound $\ln(1 - q(1 - e^p)) \leq pq + p^2/8$, which can be shown to hold for all real values p and q. We get $H(\hat{y}, y) \leq S(\hat{y}, y)$ where

$$\begin{aligned} S(\hat{y}, y) \; &= \; 2\eta B(y - \hat{y}) + 2\eta R(y - \hat{y})\frac{\hat{y} - B}{R} \\ & \quad + \frac{1}{8}(2\eta R(y - \hat{y}))^2 \\ & \quad - 2\eta y(y - \hat{y}) + \left(a + \frac{\eta^2}{b}\right)(y - \hat{y})^2 \\ &= \; \frac{(y - \hat{y})^2}{2b}\left((2 + R^2 b)\eta^2 - 4b\eta + 2ab\right) \; . \end{aligned}$$

Therefore, we must show $Q(\eta) \leq 0$ 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 $Q(\eta) \leq 0$ if and only if $a \leq 2b/(2 + R^2 b)$. □

We now combine the bounds for the individual trials to give a bound for the total loss for EG. We introduce a new parameter c and later show how it can be suitably chosen to minimize the bound.

Lemma 4 *Let $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$ be an arbitrary trial sequence. For $t = 1, \ldots, \ell$, let $R_t = \max_i x_{t,i} - \min_i x_{t,i}$, and let R be an upper bound such that $R_t \leq R$ holds for all t. Let c be an arbitrary positive constant, and let $\eta_t = 2c/(R_t^2(2 + c))$ for all t. Then for any start vector $\mathbf{s} \in \mathbf{R}^N$ and comparison vector $\mathbf{u} \in \mathbf{R}^N$, we have*

$$\begin{aligned} \mathrm{Loss}(\mathrm{EG}, S) \; &\leq \; \left(1 + \frac{c}{2}\right)\mathrm{Loss}(\mathbf{u}, S) \\ & \quad + \left(\frac{1}{2} + \frac{1}{c}\right) R^2 d_{\mathrm{RE}}(\mathbf{u}, \mathbf{s}) \end{aligned} \tag{17}$$

for the EG *algorithm with the start vector* $\mathbf{s}$ *and learning rates η_t.*

Proof For $t = 1, \ldots, \ell$, let $b_t = c/R_t^2$ and $a_t = 2b_t/(2 + R_t^2 b_t) = 2c/(R_t^2(2 + c))$. Let $\mathbf{w}_t$ be the tth weight vector of EG on the trial sequence S with $\eta_t = a_t$. Then Lemma 3 gives us

$$\begin{aligned} & a_t(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - b_t(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \\ & \quad \leq \; d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_t) - d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_{t+1}) \end{aligned} \tag{18}$$

and hence

$$\begin{aligned} & \frac{2c}{2 + c}(y_t - \mathbf{w}_t \cdot \mathbf{x}_t)^2 - c(y_t - \mathbf{u} \cdot \mathbf{x}_t)^2 \\ & \quad \leq \; R_t^2 \left(d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_t) - d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_{t+1})\right) \\ & \quad \leq \; R^2 \left(d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_t) - d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_{t+1})\right) \; . \end{aligned} \tag{19}$$

By adding the bounds (19) for $t = 1, \ldots, \ell$ we get

$$\begin{aligned} & \frac{2c}{2 + c}\mathrm{Loss}(\mathrm{EG}, S) - c\mathrm{Loss}(\mathbf{u}, S) \\ & \quad \leq \; R^2 \left(d_{\mathrm{RE}}(\mathbf{u}, \mathbf{s}) - d_{\mathrm{RE}}(\mathbf{u}, \mathbf{w}_{\ell+1})\right) \leq R^2 d_{\mathrm{RE}}(\mathbf{u}, \mathbf{s}) \; , \end{aligned}$$

which is equivalent with (17). □

We now proceed to the proof of Theorem 2. In the proof we first reduce the desired bound to a bound for the EG algorithm and then get the bound for EG by choosing the parameter c in Lemma 4 suitably.

Proof of Theorem 2 We define a new trial sequence $S' = ((\mathbf{x}'_1, y_1), \ldots, (\mathbf{x}'_\ell, y_\ell))$ by setting $\mathbf{x}'_t = (Ux_{t,1}, \ldots, Ux_{t,N}, -Ux_{t,1}, \ldots, -Ux_{t,N})$. The algorithm EG$^\pm$ has been defined in such a way that the predictions produced by EG$^\pm$ on the trial sequence S are the same as those produced by EG on the trial sequence S' with the given start vectors and learning rates. In particular, $\mathrm{Loss}(\mathrm{EG}^\pm, S) = \mathrm{Loss}(\mathrm{EG}, S')$. We further note that for any $\mathbf{u} \in \mathbf{R}^N$ with $||\mathbf{u}||_1 \le U$ and the norm U representation $\mathbf{u}'$ of $\mathbf{u}$, we have $\mathrm{Loss}(\mathbf{u}'/U, S') = \mathrm{Loss}(\mathbf{u}, S)$, and that $\max_i x'_{t,i} - \min_i x'_{t,i} = 2U||\mathbf{x}_t||_\infty$.

We now apply Lemma 4 to the trial sequence S' and comparison vector $\mathbf{u}'/U$ with $R_t = 2U||\mathbf{x}_t||_\infty$. For the uniform vector $\mathbf{s}'$ we have $d_{\mathrm{RE}}(\mathbf{u}'/U, \mathbf{s}') \le \ln 2N$, so the bound (17) now yields

$$\mathrm{Loss}(\mathrm{EG}^\pm, S) \le \left(1 + \frac{c}{2}\right) \mathrm{Loss}(\mathbf{u}, S) + \left(2 + \frac{4}{c}\right) U^2 X^2 \ln 2N \ . \quad (20)$$

The bound (12) follows from (20) by choosing $c = 4$, which corresponds to the learning rates $\eta_t = 1/(3U^2||\mathbf{x}_t||_\infty^2)$.

To get the bound (14), assume further that $\mathrm{Loss}(\mathbf{u}, S) \le K$ holds. Then (20) implies

$$\mathrm{Loss}(\mathrm{EG}^\pm, S) \le \mathrm{Loss}(\mathbf{u}, S) + 2U^2X^2 \ln 2N + \frac{cK}{2} + \frac{4U^2X^2 \ln 2N}{c} \ . \quad (21)$$

Assume first $K > 0$. The right-hand side of (21) is minimized for $c = 2UX\sqrt{(2 \ln 2N)/K}$, and we get (14). According to Lemma 4, the value of c corresponds to the learning rate given in (13). The result for the special case $K = 0$ follows by a straightforward limit argument. □

5.4 An application to a probabilistic setting

We wish to illustrate our upper bounds by considering them in a simple probabilistic setting. Assume that $\mathbf{x}_t$ is a random variable such that $||\mathbf{x}_t||_\infty \le X$ with probability 1 for some known bound X. Let $\mathbf{u} \in \mathbf{R}^N$ be an unknown vector such that $||\mathbf{u}||_1 \le U$ holds for some known bound U. Assume that the outcome y_t is given by $y_t = \mathbf{u} \cdot \mathbf{x}_t + \epsilon_t$, where ϵ_t, $t = 1, \ldots, \ell$, are independent random variables with mean 0 and variance σ^2. The proof of Theorem 2 can easily be modified to yield the bound

$$\mathrm{E}\left[\mathrm{Loss}(\mathrm{EG}^\pm, S)\right] \le \ell\sigma^2 + 2UX\sigma\sqrt{2\ell \ln 2N} + 2UX \ln 2N$$

for the expected loss of the EG$^\pm$ algorithm when the trial sequence $S = ((\mathbf{x}_1, y_1), \ldots, (x_\ell, y_\ell))$ is drawn according to this distribution. An analogous result obviously holds for the GD algorithm.

5.5 Lower bounds

The following lower bound applied with $p = q = 2$ shows that the upper bound (11) is tight. This special case was already noted by Cesa-Bianchi et al. [CBLW93].

Theorem 5 *Let $p, q \in \mathbf{R}_+ \cup \{\infty\}$ with $1/p + 1/q = 1$. Let A be an arbitrary on-line prediction algorithm, and let K, U, and X be arbitrary positive reals. Then for all positive integers N there are an instance $\mathbf{x}_t \in \mathbf{R}^N$ 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*

$$\mathrm{Loss}(A, S) \ge K + 2UX\sqrt{K} + (UX)^2 \ . \quad (22)$$

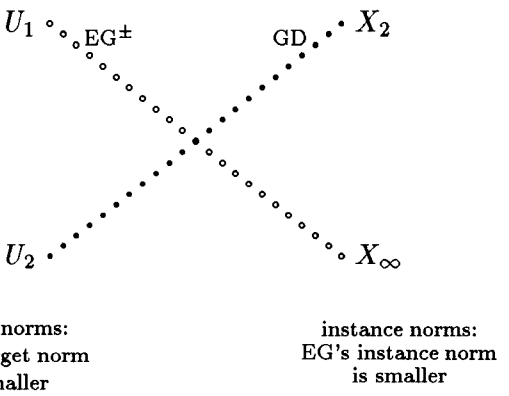

Figure 1: Schematic representation of the main factors affecting the loss bounds of the GD and EG$^\pm$ algorithms.

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

For the case $(p, q) = (1, \infty)$, the constants in the upper bound (14) do not match the constants in the lower bound (22). It is an open problem to find the optimal constants for the case $(p, q) = (1, \infty)$, or to find any upper bounds close to the lower bound (22) for the case when (p, q) is neither $(2, 2)$ nor $(1, \infty)$. The fact that Theorem 5 holds when L_p and L_q are dual norms is due to the basic property that if the norms are dual, the bound $|\mathbf{w} \cdot \mathbf{x}| \le ||\mathbf{w}||_p||\mathbf{x}||_q$ holds [Roy63].

6 Comparison of the algorithms

6.1 Comparison of the worst-case bounds

We now show that the different pairs of dual norms in the upper bounds for the GD and the 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 $\mathrm{Loss}(\mathbf{u}, S) = 0$. We can then take $K = 0$ in the bounds (11) and (14). 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 (11) simplifies to $\mathrm{Loss}(\mathrm{GD}, S) \le ||\mathbf{u}||_2^2 X_2^2$ and the bound (14) to $\mathrm{Loss}(\mathrm{EG}^\pm, S) \le 2||\mathbf{u}||_1^2 X_\infty^2 \ln 2N$.

Figure 1 illustrates the trade-offs between the different norms in the bounds. Recall that always $||\mathbf{w}||_\infty \le ||\mathbf{w}||_2 \le$

$||\mathbf{w}||_1$, 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_2U_2 and $X_\infty U_1$ are incomparable, the total effect can favor either GD or EG$^{\pm}$.

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 $\text{Loss}(\text{GD}, S) \leq kN$ and $\text{Loss}(\text{EG}^{\pm}, S) \leq 2k^2 \ln 2N$, so for $N \gg k$, the EG$^{\pm}$ algorithm has clearly the better bound. On the other hand, let $\mathbf{u} = (1, \ldots, 1)$, 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 $\text{Loss}(\text{GD}, S) \leq N$ and $\text{Loss}(\text{EG}^{\pm}, S) \leq N^2 \ln 2N$, so the GD algorithm has clearly the better bound. Thus, the bounds for GD and 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 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.

6.2 Experimental comparisons

We first consider a situation in which the worst-case upper bounds suggest EG$^{\pm}$ should outperform GD. We generated the instances uniformly at random from the set $\{-1,1\}^{100}$ of vertices of the 100-dimensional cube. We chose a comparison vector $\mathbf{u} = (-1, 1, -1, 0, \ldots, 0)$ with three nonzero and 97 zero components. We added roughly 10 % noise to the outcomes by generating for each t a variable r_t uniformly at random from the interval $[0.8, 1.2]$ and then setting $y_t = r_t \mathbf{u} \cdot \mathbf{x}_t$. Figure 3 shows how the cumulative losses $\sum_{t=1}^{m} (y_t - \hat{y}_t)^2$ for the algorithms GD and EG$^{\pm}$ increased as a function of the number m of trials in a typical trial sequence. The GD algorithm used a zero start vector. The EG$^{\pm}$ algorithm used a uniform start vector, which due to the reduction used in EG$^{\pm}$ leads to predictions being initially zero and thus corresponds to using a zero start vector. The learning rates were chosen according to (10) and (13) using the comparison vector $\mathbf{u}$. We feel that this gives a fair comparison between the algorithms, since our other experiments show that this setting leads to almost optimal learning rates. The figure also shows as horizontal lines the worst-case upper bounds for the losses. These bounds are based on slightly sharper versions of Theorems 1 and 2 [KW94].

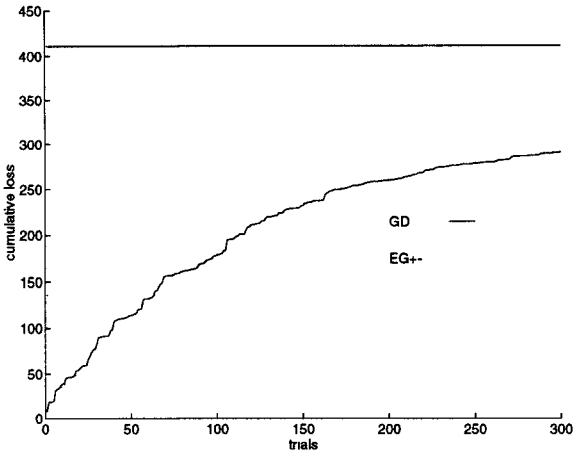

Figure 2: Cumulative losses of GD (solid line) and EG$^{\pm}$ (dotted line), with their upper bounds, for instances $\mathbf{x}_t \in \{-1,1\}^{100}$ and target $\mathbf{u} = (-1, 1, -1, 0, \ldots, 0)$, and 10 % noise.

From Figure 2 we see that the worst-case upper bounds are quite tight for the random data we consider. They become even tighter if we do not add noise to the outcomes. Obviously, the difference between the algorithms can be made arbitrarily large by increasing the number of dimensions. On the other hand, we can make experiments in which the GD algorithm outperforms the EG$^{\pm}$ algorithm, again by an arbitrary amount of loss depending on the number of dimension. This is the case for instance if the comparison vector $\mathbf{u}$ is chosen from $\{-1,1\}^N$ and the instances are from the N-dimensional Euclidean sphere. For these and other experiments, see the full paper [KW94].

We feel that the situation that favors the 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 $f_t(\mathbf{x}_t)$ for some suitable chosen basis functions f_t. 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 [BGV92]. After such an expansion, one would expect relatively few of the new variables to be relevant for prediction. Assuming that the input variables are in the range $[-1, 1]$, this does not increase the L_∞ norms of the instances. If the outcomes are actually given by some degree k polynomial of the input variables, the loss of the EG$^{\pm}$ algorithm after the expansion of the instances would be $O(q \ln N)$. 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 $O(N^q)$. Note that if the original N input variables contain r irrelevant random input variables, these are expanded into $O(r^k)$ irrelevant pseudorandom variables, and our experiments show that such variables do increase the loss of the GD algorithm.

Figure 3 shows the cumulative losses $\sum_{t=1}^{m} (y_t - \hat{y}_t)^2$ as a function of the number m of trials for the algorithms GD and EG$^{\pm}$ in a typical experiment with expanded instances. We have taken $q = N = 8$. As we see, the loss curve of EG$^{\pm}$ flattens earlier, indicating that EG$^{\pm}$ learns faster. The total loss is also smaller for EG$^{\pm}$. The original instances have been chosen uniformly from $\{-1,1\}^8$, and an expanded instance

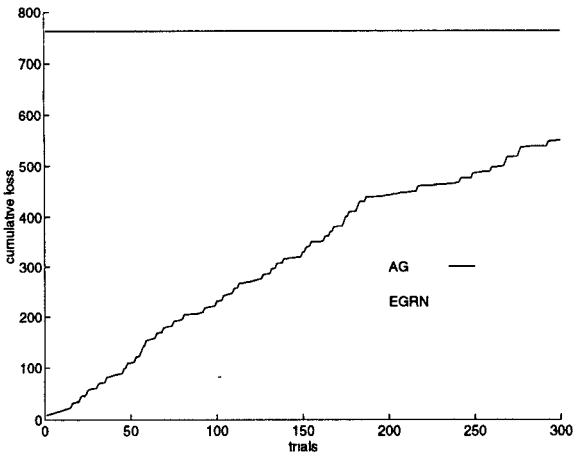

Figure 3: Cumulative losses of GD and EG$^{\pm}$, with their upper bounds, for sparse target and expanded instances.

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 outcomes have been chosen to have $y_t = \mathbf{u} \cdot \mathbf{x}_t$, with no noise.

6.3 Discussion

Our experiments show that already on simple artificial random data, the actual losses of the algorithms come close to their worst-case upper bounds. This is true both with and without noise in the outcomes. In particular, we have observed that the GD algorithm does suffer when there is a large number of irrelevant input variables. The number of examples it 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. This is easily seen to be true in the special case that the number of instances does not exceed N, and the instance vectors are orthogonal to each other. Since the weight vector of the GD algorithm satisfies $\mathbf{w}_m = \mathbf{w}_1 + \sum_{t=1}^{m-1} a_t \mathbf{x}_t$ for some scalars a_t, the prediction $\hat{y}_m = \mathbf{w}_m \cdot \mathbf{x}_m$ is in this case not at all affected by what the algorithm has seen at trials $1, \ldots, m-1$. This would also be true if we at each trial solved a linear least squares problem to find a weight vector that fits all the examples and has the least L_2 norm.

For the 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 EG$^{\pm}$ algorithm has a strong bias towards hypotheses with few relevant variables, so if only few variables are needed for prediction, then EG$^{\pm}$ avoids the curse of dimensionality. The GD algorithm and the linear least squares algorithm are biased towards hypotheses with small L_2 norm, and even if only few variables are relevant, they use all the dimensions in a futile search for a good predictor with a small norm.

7 Conclusion

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 fixed 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.

3. The distance function also serves in a second role as a potential function in proving worst-case loss bounds by amortized analysis. 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. It 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 predict nonlinearly 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.

Even for the linear single 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 log loss. It would also be interesting to find new distance measures that would lead to new 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. Our bounds for GD are provably optimal, but we still need matching lower bounds for the EG$^{\pm}$ algorithm.

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 algorithms 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 [Lit88], 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. In the linear case we have been able to verify that the new family indeed

performs better when the target is sparse. We expect to see similar behavior also in more general settings.

Acknowledgments

We wish to thank Nicolò Cesa-Bianchi, David P. Helmbold, and Yoram Singer for their comments and Robert E. Schapire for simplifying the proof of Lemma 3.

References

[Ama94] S. Amari. Information geometry of the EM and em algorithms for neural networks. Technical Report METR 94-4, University of Tokyo, Tokyo, 1994.

[Ama95] S. Amari. The EM algorithm and information geometry in neural network learning. *Neural Computation*, 7(1):13–18, January 1995.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, October 1989.

[BGV92] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. 5th Workshop on Computational Learning Theory*, pages 144–152. ACM Press, New York, NY, 1992.

[CBFH+95] N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, D. Haussler, R. E. Schapire, and M. K. Warmuth. How to use expert advice. Technical Report UCSC-CRL-95-19, Univ. of Calif. Computer Research Lab, Santa Cruz, CA, 1995. An extended abstract appeared in STOC '93.

[CBLW93] N. Cesa-Bianchi, P. Long, and M. Warmuth. Worst-case quadratic loss bounds for on-line prediction of linear functions by gradient descent. Technical Report UCSC-CRL-93-36, Univ. of Calif. Computer Research Lab, Santa Cruz, CA, 1993. An extended abstract appeared in COLT '93.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, NY, 1973.

[Hay93] S. Haykin. *Neural Networks: a Comprehensive Foundation.* Macmillan, New York, NY, 1993.

[HKW94] D. Haussler, J. Kivinen, and M. K. Warmuth. Tight worst-case loss bounds for predicting with expert advice. Technical Report UCSC-CRL-94-36, Univ. of Calif. Computer Research Lab, November 1994. Preliminary versions appeared in EuroCOLT '93 and EuroCOLT '95.

[HKW95] D. P. Helmbold, J. Kivinen, and M. K. Warmuth. Worst-case loss bounds for sigmoided linear neurons. Unpublished manuscript, 1995.

[HSSW95] D. P. Helmbold, R. E. Schapire, Y. Singer, and M. K. Warmuth. A comparison of new and old algorithms for a mixture estimation problem. Proc. 8th Workshop on Computational Learning Theory, July 1995 (To appear).

[KW94] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, Univ. of Calif. Computer Research Lab, June 1994. Retrievable from ftp.cse.ucsc.edu as /pub/tr/ucsc-crl-94-16.ps.Z .

[KW95] J. Kivinen and M. K. Warmuth. The Perceptron algorithm vs. Winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant. Proc. 8th Workshop on Computational Learning Theory, July 1995 (To appear).

[Lit88] N. Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, April 1988.

[Lit89] N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms.* PhD Thesis, Technical Report UCSC-CRL-89-11, University of California Santa Cruz, 1989.

[LLW91] N. Littlestone, P. M. Long, and M. K. Warmuth. On-line learning of linear functions. To appear in *Journal of Computational Complexity.* A preliminary version appeared in *Proc. 23rd Symposium on Theory of Computing*, pages 465–475. ACM Press, New York, NY, 1991.

[LW94] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, February 1994.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

[Roy63] H. Royden. *Real Analysis.* Macmillan, New York, NY, 1963.

[SW94] R. E. Schapire and M. K. Warmuth. On the worst-case analysis of temporal-difference learing algorithms. To appear in the special issue of *Machine Learning* on reinforcement learning. A preliminary version appeared in *Proc. 11th International Conference on Machine Learning*, pages 266–274, Morgan Kaufmann, San Francisco, CA, 1994.

[VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[Vov90] V. Vovk. Aggregating strategies. In *Proc. 3rd Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, San Mateo, CA, 1990.

[WS85] B. Widrow and S. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1985.