Learning of Depth Two Neural Networks with Constant Fan-in at the Hidden Nodes (extended abstract)

Peter Auer[*] and **Stephen Kwek**[†] and **Wolfgang Maass**[‡] and **Manfred K. Warmuth**[§]

Abstract

We present algorithms for learning depth two neural networks where the hidden nodes are threshold gates with constant fan-in. The transfer function of the output node might be more general: we have results for the cases when the threshold function, the logistic function or the identity function is used as the transfer function at the output node. We give batch and on-line learning algorithms for these classes of neural networks and prove bounds on the performance of our algorithms. The batch algorithms work for real valued inputs whereas the on-line algorithms assume that the inputs are discretized.

The hypotheses of our algorithms are essentially also neural networks of depth two. However, their number of hidden nodes might be much larger than the number of hidden nodes of the neural network that has to be learned. Our algorithms can handle such a large number of hidden nodes since they rely on multiplicative weight updates at the output node, and the performance of these algorithms scales only logarithmically with the number of hidden nodes used.

[*] Address: Department of Computer Science, University of California, Santa Cruz, CA 95064. E-mail: pauer@cse.ucsc.edu

[†] Address: Department of Computer Science, University of Illinois, Urbana, IL 61801. E-mail: kwek@cs.uiuc.edu.

[‡] Address: Institute of Theoretical Computer Science, Technische Universität Graz, Klosterwiesgasse 32/2, A-8010 Graz, Austria. E-mail: maass@igi.tu-graz.ac.at

[§] Address: Department of Computer Science, University of California, Santa Cruz, CA 95064. E-mail: manfred@cis.ucsc.edu.

1 Introduction

In this paper we elaborate on a technique to expand learning algorithms for single neurons to learning algorithms for depth two neural networks. This technique works for on-line learning algorithms for single neurons whose total loss bounds scale only logarithmically with the input dimension. Quite a number of such algorithms were found recently [Lit88, CBLW95, KW94, HKW96]. All of them rely on a multiplicative update scheme of the weights and these update schemes are motivated [KW94] by the minimum relative entropy principle of Kullback [KK92, Jum90].

The way we get a depth two neural network from a single neuron is the following. We expand a single neuron by replacing the input nodes of the neuron by hidden nodes which compute linear threshold functions of the inputs (see Figure 1). We only require that the fan-in of the hidden nodes is some constant d. Thus the neural networks we are considering have N inputs, k hidden nodes which calculate linear threshold functions of d of the N inputs, and an output node with some transfer function ϕ.

We will consider two types of learning models: an on-line model and a batch model. In the on-line model, [Lit88, Ang88] learning proceeds in trials. In each trial t an input pattern $\mathbf{x}_t$ is presented to the learner and the learner has to produce an output $\hat{y}_t$. Then the learner receives the desired output y_t and incurs a loss $L(y_t, \hat{y}_t)$ for some loss function[1] $L : \mathbf{R} \times \mathbf{R} \to [0, \infty)$. The performance of the on-line learner is measured by the total loss over all trials, compared with the total loss of the neural network which best fits the $(\mathbf{x}_t, y_t)$ pairs of all trials.

In the batch model [HLW94] the learner is given a training sequence of examples $\mathcal{S} = \langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\rangle$ which are drawn independently at random from some

[1] Examples of loss functions are the discrete loss $L(y,\hat{y}) = 1$ if $\hat{y} \neq y$ and $L(y,\hat{y}) = 0$ if $\hat{y} = y$, the square loss $L(y,\hat{y}) = (y-\hat{y})^2$, and the entropic loss $L(y,\hat{y}) = y \ln \frac{y}{\hat{y}} + (1-y) \ln \frac{1-y}{1-\hat{y}}$.

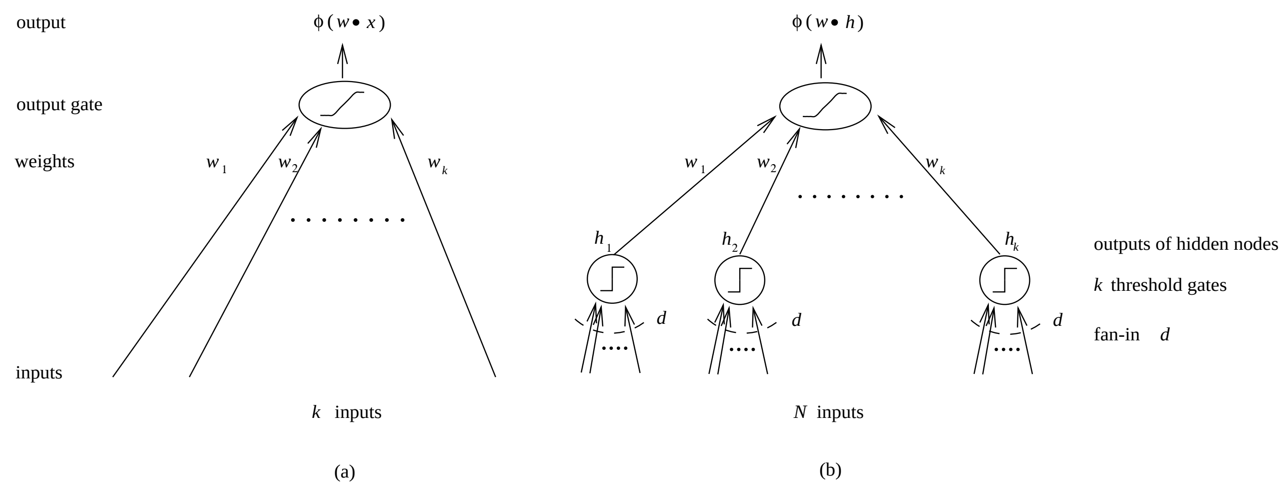

Figure 1: (a) The single neuron. (b) The neuron expended into a depth two neural network.

fixed but unknown distribution. Based on this training sequence the learner has to produce a hypothesis, and the performance of the learner is measured by the expected[2] loss of its hypothesis on an unseen example, compared with the expected loss of the neural network which performs best with respect to the fixed but unknown distribution.

We note here that our batch as well as our on-line leaning algorithms are *agnostic* learning algorithms [Hau92, KSS92], in the sense that they make no assumptions whatsoever about the target concept to be learned. Instead, we compare their performance with the performance of the best hypothesis for this unknown target from a comparison or "touchstone" class. In our case these touchstone classes are classes of depth two neural networks.

Now we describe our technique to transform a learning algorithm for a single neuron into a learning algorithm for depth two neural networks of the type described above. Assume we have an on-line learning algorithm A for a single neuron with k inputs and transfer function ϕ (see Figure 1a). When learning the depth two neural network we would like to use this algorithm A to learn the weights from the hidden nodes to the output node. This leaves us with the problem of obtaining the outputs of the hidden nodes which are determined by their weights. The main idea of our technique is to increase the number of hidden nodes such that each possible linear threshold function of the inputs is calculated by one of the hidden nodes. (Observe that each node has to calculate a threshold function of only d out of the N inputs.) Then the weights to the (now very many) hidden nodes can be fixed and only the weights from the hidden nodes to the output node have to learned, which can be done using algorithm A.

The problem with this approach is that the performance of the learning algorithm for the single neuron might degrade dramatically when the number of inputs is enlarged by too much. Thus our technique will work only for learning algorithms whose performance scales very moderately with the input dimension. We will make use of three algorithms for single neurons for which this is the case. All these algorithms use a multiplicative weight update and their performance scales logarithmically in the number of inputs. Our technique gives us learning algorithms for depth two neural networks with very reasonable performance bounds and polynomial run-time (with the fixed fan-in d in the exponent).

Whereas the application of the above technique is quite straightforward for the batch model there are additional difficulties for the on-line model. In order to keep the run-time reasonable it is not possible to deal with all the candidate hidden nodes individually. Instead they have to be collected into groups such that all nodes in a group "behave alike". Then one has to deal only with a relatively small number of groups which gives the required speedup. This grouping technique was developed by Maass and Warmuth [MW95] who called it "virtual weights". For its application the exact number of nodes in a group has to be known. Since in our case this number seems to be computationally expensive to calculate we had to extend the "virtual weights" technique by using an approximation for the number of nodes in a group. This approximation can be calculated from the volume of a polytope which is associated with the group under consideration.

[2] The expectation is taken over the random draw of the unseen example as well as over the m independent random draws that produced training sequence.

1.1 Related result

There are a number of previous related results for learning depth two neural networks in the PAC model which is a model closely related to the batch model considered here [HLW94]. As in our paper the resources of the algorithms scale exponentially in the fan-in of the hidden nodes. Bshouty et al. [BGM$^+$96] gave a noise-tolerant PAC algorithm for learning arbitrary boolean functions of s halfspaces of fixed fan-in d. Similarly, Koiran [Koi94] gave a PAC learning algorithm for neural networks of depth two of the form considered here (Figure 1b) with the identity transfer function. Maass [Maa93] gives a PAC learning algorithm for the case when the transfer function is a threshold function. Maass' algorithm also works for fixed depth neural networks with piecewise polynomial activation functions and a constant number of analog inputs.

In contrast we have results for the batch as well as for the on-line model and our results are quite general in the sense that we give a reduction from learning algorithms for single neurons to learning algorithms for depth two neural networks.

1.2 Organization of the paper

In Section 2 we describe the basic ideas we will use to transform algorithms for single neurons into algorithms for depth two neural networks. The main questions are which hidden nodes should be generated and how can they be maintained efficiently. The first part of Section 3 gives general considerations about the proofs for our transformed on-line algorithms, Section 3.1 states the results, Section 3.2 describes the actual transformation of an on-line learning algorithm for a single neuron, and Section 3.3 contains the analysis of the transformed algorithm. Sections 3.4 and 3.5 sketch the transformation of two other on-line learning algorithms for single neurons. Section 4 contains our results and proof sketches for the batch model.

2 The hidden nodes

In this section we describe which hidden nodes are maintained by the learning algorithm. We disregard the weights from the hidden nodes to the output node but concentrate on the hidden nodes which are represented by the weights from the inputs to the hidden nodes. Since we restrict the fan-in of the hidden nodes to be at most d, each hidden node computes a linear threshold function of the inputs where besides the bias at most d of the weights are non-zero. It is also worthwhile to mention that the correct classification of the input patterns will be of no concern for the construction of the hidden nodes.

Our goal is to have one hidden node for each threshold function. But observe that there is no need to distinguish between threshold functions which coincide on all input patterns seen so far. There is simply no evidence which could tell the learner to prefer one over the other threshold function when they behave identically on the seen input patterns. Therefore we have to construct only one hidden node for each class of threshold functions which calculate the same values for the input patterns seen so far.

For the batch model the situation is particularly simple since all training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ are given in advance. To calculate the representatives for each class of threshold functions we have to consider the possible classifications of the input patterns which can be realized by a hidden node. Since the fan-in is at most d the function calculated by a hidden node can be decomposed into a projection $p : \mathbf{R}^N \to \mathbf{R}^d$ and a linear threshold function $h : \mathbf{R}^d \to \{0, 1\}$ defined as follows:

$$h(z_1, \ldots, z_d) = \begin{cases} 1 & \text{if } c_0 + \sum_{i=1}^{d} c_i \cdot z_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{c} = (c_0, \ldots, c_d) \in \mathbf{R}^{d+1}$. Now observe that for some fixed $\mathbf{z} = (z_1, \ldots, z_d)$ the weights which correspond to a threshold function with $h(\mathbf{z}) = 1$ are given by the halfspace $\{\mathbf{c} \in \mathbf{R}^{d+1} : \mathbf{c} \cdot (1, \mathbf{z}) \geq 0\}$ of the $(d+1)$-dimensional weight space. Thus, for a fixed projection p, the hyperplanes $\{\mathbf{c} \in \mathbf{R}^{d+1} : \mathbf{c} \cdot (1, p(\mathbf{x}_\tau)) \geq 0\}$, $\tau = 1, \ldots, m$, divide the weight space into polyhedra such that these polyhedra represent all possible linear threshold functions on the points $p(\mathbf{x}_1), \ldots, p(\mathbf{x}_m)$. A lemma from computational geometry [Ede87] states that the number of polyhedra in which $\mathbf{R}^{d+1}$ is dissected by m hyperplanes is at most $\sum_{i=0}^{d+1} \binom{m}{i} \leq m^{d+1}$ and that these polyhedra can be computed efficiently. Since for each of the $\binom{N}{d}$ projections $p : \mathbf{R}^N \to \mathbf{R}^d$ the corresponding $(d+1)$-dimensional weight space is divided by the corresponding hyperplanes the number of necessary hidden nodes is upper bounded by $\binom{N}{d} m^{d+1}$. The weights of the hidden nodes are given by any choice of points from the corresponding polyhedra.

Even though the loss bounds of our algorithms scale logarithmically with the number of hidden nodes, the time bounds of the algorithms are proportional to the number of hidden nodes and thus exponential in the fan-in d. Therefore we assume throughout this paper that d is constant (and small), and for $d = 2$ or $d = 3$ our algorithm might actually be practical. The exponential growth in d is not surprising, since if the time bounds were polynomial in d then one of our algorithms would lead to a polynomial, agnostic PAC learning algorithm for DNF formulas, using hypotheses more general than DNF formulas. The problem of finding a polynomial PAC learning algorithm for DNF formulas has been open for a long time now, even if the algorithm is allowed to ask membership queries in addition to receiving random examples (and we would write a very different paper if we could solve it).

In the on-line model the hidden nodes are maintained similarly but there are two additional difficulties. First, the examples are not known in advance but are given to the learner one by one. Thus the number of hidden nodes could not be fixed in advance but would have to be increased during the learning process. Second, it is generally harder to learn in the on-line model than it is in the batch model. Consider for example the concept class of initial intervals of $[0,1]$. This class can be realized by neural networks of the type considered in this paper with just a single hidden node of fan-in one. Whereas initial intervals can be easily learned in the batch model, an unbounded loss can be forced for any on-line learner since the on-line learner has to exactly identify the (real-valued) boundary of the initial interval. The problem of perfect precision in threshold gates is usually circumvented by making additional assumptions about the weights of the threshold functions to be learned and the input patterns. These assumptions boil down to assuming that using the "correct" weights the values computed by the threshold gates do not change when the inputs are slightly perturbed. Since this is equivalent to assuming discretized inputs we require that for the on-line model the inputs are integers from $\mathbf{Z}_P = \{-P, \ldots, P\}$.

The assumption of only discretized inputs solves the second problem mentioned above and it also helps to solve the first problem. It is known that for inputs from $\mathbf{Z}_P$ there are at most $\binom{N}{d} \cdot (dP)^{O(d)}$ linear threshold functions with fan-in d [MT92]. Thus we could fix the number of hidden nodes by maintaining that many hidden nodes. Unfortunately, for large P, this would result in a *very* unreasonable run time of the algorithm. Instead we use "virtual weights" [MW95], which means that we group hidden nodes into blocks when they compute the same values for all input patterns seen so far. We are able to do this because the weights from the hidden nodes to the output node are the same for all hidden nodes in such a block. Thus the weighted sum of the hidden nodes $\sum_j w_j h_j$ can be replaced by the weighted sum over the blocks $\sum_B w_B \cdot |B| \cdot h_B$ where w_B denotes the common weight of all the nodes in block B, $|B|$ denotes the number of nodes in B, and h_B denotes the output computed by all the nodes in B. Thus, instead of summing over all linear threshold functions and updating all weights, the algorithm has to sum only over a relatively small number of blocks and updates only one weight per block. Since different blocks correspond to linear threshold functions which give different values for the input patterns seen so far, the blocks are given by the polyhedra in which the weight space is dissected by the hyperplanes corresponding to the input patterns seen so far. Thus, after t trials (input patterns) the number of blocks is at most $\binom{N}{d} t^{d+1}$.

Note, that in contrast to the algorithm for the batch model where each polyhedron corresponded to a single hidden node, in the on-line model a polyhedron corresponds to a block of hidden nodes. Thus a new example does not increase the number of hidden nodes but only the number of blocks into which the hidden nodes are grouped. This is quite essential because the algorithms which learn the weights from the hidden nodes to the output node cannot deal with a growing number of hidden nodes.

Finally, there is still a difficulty in the algorithm for the on-line algorithm which is the calculation of $|B|$, the number of threshold functions or hidden nodes in a block. In general it seems to be very complicated to calculate this number exactly. Therefore we replace the exact number of nodes in a block by the volume of the polyhedron which corresponds to this block. This means that we only approximate the original weighted sum of the hidden nodes which has to be taken into account when calculating the loss bounds.

3 Algorithms for the on-line model

In this section, we present our on-line algorithms for learning depth two neural networks which are based on the ideas described in the previous section. We will start with three learning algorithms for single neurons and transform them into learning algorithms for depth two neural networks. For an overview of the results see Section 3.1. For an example of the transformation from a learning algorithm for single neurons to a learning algorithm for depth two neural networks see Section 3.2. In the remaining of this section we give some more motivation and a more abstract description of this transformation.

All the learning algorithms for single neurons we consider here maintain a weight vector $\mathbf{w}_t$ which after training is supposed to approximate the optimal weight vector $\mathbf{u}$. In each trial t the weight vector $\mathbf{w}_t$ is used to compute the output $\hat{y}_t$ of the learning algorithm for the input pattern $\mathbf{x}_t = (x_{t,1}, \ldots, x_{t,k})$. After receiving the desired output y_t the weights are updated multiplicatively, i.e. they are multiplied with some positive factors which depend on y_t, $\mathbf{x}_t$, and $\mathbf{w}_t$. Since the multiplicative update does not change the sign of a weight the learning algorithms have to maintain pairs of weights $w_{t,i}^+, w_{t,i}^- > 0$ where $w_{t,i}^+$ represents a positive weight for the i-th input coordinate and $w_{t,i}^-$ represents a negative weight. Then the predictions of the learning algorithms are given by $\hat{y}_t = \phi\left(\sum_{i=1}^k (w_{t,i}^+ - w_{t,i}^-) \cdot x_{t,i}\right)$ and the output of the neuron with optimal weight vector $\mathbf{u}$ can be written as $y_t^* = \phi\left(\sum_{i=1}^k (u_i^+ - u_i^-) \cdot x_{t,i}\right)$.

The analysis of the learning algorithms for single neurons relies on inequalities of the type

$$D(\mathbf{u}, \mathbf{w}_t) - D(\mathbf{u}, \mathbf{w}_{t+1}) \geq a \cdot L(y_t, \hat{y}_t) - b \cdot L(y_t, y_t^*). \quad (1)$$

Here D is a distance function measuring the distance of the current weight vector from the optimal weight vector $\mathbf{u}$, and a and b are appropriately chosen positive constants. For multiplicative updates entropy based distance functions D are used [KW94, Lit91]. The left hand side of the above inequality might be seen as the progress towards the optimal weight vector $\mathbf{u}$: if the loss of the algorithm is large compared to the loss of the optimal neuron then the progress of the weight vector towards $\mathbf{u}$ is large. For a sequence $\mathcal{S}$ of T trials the above equality is added over all trials, giving

$$D(\mathbf{u}, \mathbf{w}_1) - D(\mathbf{u}, \mathbf{w}_{T+1}) \geq a \cdot L_A(\mathcal{S}) - b \cdot L_{\mathbf{u}}(\mathcal{S}). \quad (2)$$

Here $L_A(\mathcal{S})$ denotes the total loss of algorithm A on sequence $\mathcal{S}$ and $L_{\mathbf{u}}(\mathcal{S})$ denotes the loss of the neuron with optimal weight vector $\mathbf{u}$. Solving for $L_A(\mathcal{S})$ gives an upper bound for the total loss of the algorithm.

In the modified algorithms for learning depth two neural networks we have in each trial a set of blocks $\mathcal{B}_t$ with volumes $\mathbf{vol}_t = (\mathrm{vol}_t(B))_{B \in \mathcal{B}_t}$ and weights $\mathbf{w}_t = (w_t(B))_{B \in \mathcal{B}_t}$. After receiving input pattern $\mathbf{x}_t$ some of the blocks might have to be split accordingly to which values the functions in these blocks give for input pattern $\mathbf{x}_t$. This is done by dissecting blocks with the hyperplane in the weight space corresponding to input pattern $\mathbf{x}_t$. We denote the new set of blocks by $\mathcal{B}_{t+1}$, the new volumes by $\mathbf{vol}_{t+1}$ and the corresponding weights[3] by $\tilde{\mathbf{w}}_t$. After receiving the desired output y_t the weights are updated to $\mathbf{w}_{t+1}$.

Whereas in the case of a single neuron the weight vector $\mathbf{w}_t$ was directly related to the optimal weight vector $\mathbf{u}$ this relation is more complicated for depth two neural networks. Since the weight $w_t(B)$ denotes the weights of the threshold functions in block B and $\mathrm{vol}(B)$ approximates the number of functions in block B, the total weight of block B is given by $w_t(B) \cdot \mathrm{vol}(B)$. The corresponding optimal weight $u(B)$ of block B can be calculated from the weights in the optimal depth two neural network. The optimal weight vector $\mathbf{u}$ for the connections between hidden nodes and output node assigns some weight to all the linear threshold functions represented by a hidden node and weight 0 to all the other linear threshold functions. This weight assignment can be extended to blocks: the weight $u(B)$ of a block B is just the sum of the weights of the functions in B.[4]

With these notations we can apply inequality (1) from the update of the single neuron. Let $\mathbf{w} \times \mathbf{vol} = (w(B) \cdot \mathrm{vol}(B))_{B \in \mathcal{B}}$ denote the vector of the total weights of the blocks in $\mathcal{B}$. Since in the update step of the modified algorithm the blocks are not changed we get

$$\begin{aligned} D(\mathbf{u}, \tilde{\mathbf{w}}_t \times \mathbf{vol}_{t+1}) - D(\mathbf{u}, \mathbf{w}_{t+1} \times \mathbf{vol}_{t+1}) & \quad (3) \\ \geq a \cdot L(y_t, \hat{y}_t) - b \cdot L(y_t, y_t^*) & \quad (4) \end{aligned}$$

where the distance function D is applied to the total weights of the blocks in $\mathcal{B}_{t+1}$, y_t is the desired output, $\hat{y}_t$ is the prediction of the learning algorithm, and y_t^* is the output of the optimal depth two neural network. In order to replace $D(\mathbf{u}, \tilde{\mathbf{w}}_t \times \mathbf{vol}_{t+1})$ by $D(\mathbf{u}, \mathbf{w}_t \times \mathbf{vol}_t)$ in (3) we assume that there is a function f_D such that

$$\begin{aligned} D(\mathbf{u}, \mathbf{w}_t \times \mathbf{vol}_t) - D(\mathbf{u}, \tilde{\mathbf{w}}_t \times \mathbf{vol}_{t+1}) & \\ \geq f_D(\mathbf{u}, \mathbf{vol}_t) - f_D(\mathbf{u}, \mathbf{vol}_{t+1}). & \quad (5) \end{aligned}$$

This additional inequality is required to capture the effect on the distance function when blocks are split. Observe that inequality (5) is the only new part in the analysis of the modified algorithms. The other parts can be taken from the analysis of the learning algorithms for single neurons. Combining (3) and (5) we get

$$\begin{aligned} D(\mathbf{u}, \mathbf{w}_t \times \mathbf{vol}_t) &- D(\mathbf{u}, \mathbf{w}_{t+1} \times \mathbf{vol}_{t+1}) \\ &\geq f_D(\mathbf{u}, \mathbf{vol}_t) - f_D(\mathbf{u}, \mathbf{vol}_{t+1}) \\ &\quad + a \cdot L(y_t, \hat{y}_t) - b \cdot L(y_t, y_t^*). \end{aligned}$$

Summing over all trials we get

$$\begin{aligned} D(\mathbf{u}, \mathbf{w}_1 \times \mathbf{vol}_1) &- D(\mathbf{u}, \mathbf{w}_{T+1} \times \mathbf{vol}_{T+1}) \\ &\geq f_D(\mathbf{u}, \mathbf{vol}_1) - f_D(\mathbf{u}, \mathbf{vol}_{T+1}) \\ &\quad + a \cdot L_A(\mathcal{S}) - b \cdot L_{\mathrm{opt}}(\mathcal{S}) \qquad (6) \end{aligned}$$

where $L_A(\mathcal{S})$ is the loss of the learning algorithm on the trial sequence $\mathcal{S}$ and $L_{\mathrm{opt}}(\mathcal{S})$ is the loss of the optimal depth two neural network. Solving for $L_A(\mathcal{S})$ gives a mistake bound for the algorithm.

Note again that an essential step in the analysis of our on-line algorithms for depth two neural networks is the introduction of the function f_D which allows us a very general and elegant treatment of the splitting of blocks. For specific algorithms with specific distance functions D the only remaining step is to find such a function f_D which satisfies inequality (5) and gives a good loss bound by inequality (6).

[3] The weights are not changed but only duplicated when a block is split.

[4] One function might be represented in more then one block. Then the weight of the function has to be distributed equally among all representations of this function.

3.1 Results

Definition 1 *Let $\mathcal{N}(N,d,U,\phi)$ be the class of neural networks for input patterns $\mathbf{x} \in \mathbf{R}^N$ of the following type: the hidden nodes compute a linear threshold function of at most d of the inputs, the output node computes the transfer function ϕ of the weighted sum of the outcomes at the hidden nodes, and the L_1-norm[5] of the weights from the hidden nodes to the output node is bounded by U.*

If ϕ is a threshold function such that $\phi(z) = 1$ for $z \geq 1$ and $\phi(z) = 0$ for $z < 1$, then the class of neural networks $\mathcal{N}(N,d,U,\phi,\delta)$ with separation $0 < \delta \leq 1$ contains all neural networks of the above type which in addition satisfy $|\mathbf{u} \cdot \mathbf{y} - 1| \geq \delta$ for any binary vector $\mathbf{y}$ where $\mathbf{u}$ is the vector of the weights from the hidden nodes to the output node.

For any loss function L and any sequence of examples $\mathcal{S}$ let $L_{\rm opt}(\mathcal{S},N,d,U,\phi)$ denote the minimal loss on $\mathcal{S}$ among all neural networks in $\mathcal{N}(N,d,U,\phi)$.

The loss of an on-line algorithm A on a sequence of examples $\mathcal{S}$ is denoted by $L_A(\mathcal{S})$.

Theorem 2 *The following results hold for an arbitrary sequence $\mathcal{S}$ of examples $(\mathbf{x}_t, y_t)$ where the input patterns $\mathbf{x}_t$ are from $\mathbf{Z}_P^N$ and the desired output y_t lies in the range of the considered transfer function ϕ.*

(a) For the logistic transfer function $\phi(z) = \frac{1}{1+e^{-z}}$ and the entropic loss there is an on-line learning algorithm A such that

$$L_A(\mathcal{S}) \;\leq\; \frac{4}{3} \cdot L_{\rm opt}(\mathcal{S},N,d,U,\phi) + 3 \cdot d^2 \cdot U^2 \cdot \ln(16dNP).$$

The run time in trial t is $O\left(\binom{N}{d} t^{d+1}\right)$.

(b) For the identity function $\phi(z) = z$ and the square loss there is an on-line learning algorithm A such that

$$L_A(\mathcal{S}) \;\leq\; \frac{3}{2} \cdot L_{\rm opt}(\mathcal{S},N,d,U,\phi) + 12 \cdot d^2 \cdot U^2 \cdot \ln(16dNP).$$

The run time in trial t is $O\left(\binom{N}{d} t^{d+1}\right)$.

(c) For the threshold function $\phi(z) = 1$ for $z \geq 1$ and $\phi(z) = 0$ for $z < 1$, for separation $0 < \delta \leq 1$ and with the discrete loss, there is an on-line learning algorithm A such that

$$L_A(\mathcal{S}) \;\leq\; \frac{4U}{\delta} \cdot L_{\rm opt}(\mathcal{S},N,d,U,\phi,\delta) + \frac{64d^2U}{\delta^2} \cdot \ln(16dNP).$$

The run time in trial t is $O\left(\binom{N}{d} m_t^{d+1}\right)$ where m_t is the number of mistakes made up to trial t.

Recall that the VC dimension of a class is always a lower bound on the discrete loss obtainable by any on-line algorithm [Lit88, Ang90]. The class of k-term monotone DNF with at most d literals per term has VC dimension $kd(\ln_2 N - \ln_2 \ln_2 N)$, when $k = N$ and $d = \ln N$ (by Lemma 6 of [Lit88]). Hence, any on-line algorithm for learning this class of formulas must have discrete loss $\Omega(kd \ln_2 N)$. The class of neural networks considered here contains this class of DNF formulas and the discrete loss bound of our on-line algorithm is $O(kd^2 \ln_2 N)$ (Apply part (c) of Theorem 2 with $\delta = 1$). Thus, for this class of DNF formulas, our loss bound is reasonably good. We believe there are cases where the other bounds of Theorem 2 are more or less tight as well.

3.2 The transformation of the learning algorithm for the logistic transfer function and the entropic loss

In this section we give a quite detailed description of the transformation from the learning algorithm for a single neuron to a learning algorithm for depth two neural networks where the transfer function is the logistic function and the performance of the algorithm is measured by the entropic loss. The analysis of the modified algorithm is given in Section 3.3. The transformation of the algorithms for other transfer and loss functions is very similar and is only sketched in Sections 3.4 and 3.5.

We start with an algorithm which learns single neurons with the logistic transfer function $\phi(z) = \frac{1}{1+e^{-z}}$ and where the loss of the algorithm is measured by the entropic loss function. In [HKW96] an algorithm $A_{\log}^{(1)}$ (a version of $EG^{\pm}$) for learning such a neuron was developed (see Figure 2). In each trial each weight is updated by a positive factor. Since such multiplicative updates do not change the sign of a weight, two weights have to be maintained for each input, one representing a possibly positive value of the weight, the other representing a possibly negative value of the weight. The total loss of this algorithm is compared with the total loss of the optimal neuron where the weights are restricted to have L_1-norm at most U. Although the original algorithm was more general it is sufficient for our purposes to consider only inputs from $[0,1]$. The loss bound obtained

[5] The L_1-norm of a weight vector $\mathbf{u} \in \mathbf{R}^k$ is $||\mathbf{u}||_1 = \sum_{i=1}^k |u_i|$.

Parameters:

The number of inputs k and an upper bound U such that $||\mathbf{u}||_1 \leq U$.

Notations:

The algorithm maintains normalized weights $w^+_{t,i}, w^-_{t,i} > 0$, $i = 1, \ldots, k$, with $\sum_{i=1}^k \left[w^+_{t,i} + w^-_{t,i}\right] = 1$. Intentionally the scaled difference $U \cdot (w^+_{t,i} - w^-_{t,i})$ approximates the optimal weight u_i.

Initialization:

Set $w^+_{1,i} = w^-_{1,i} = \frac{1}{2k}$ for $i = 1, \ldots, k$ and set the learning rate $\eta = \frac{4}{U^2}$.

Prediction: In each trial $t \geq 1$:

Receive input pattern $\mathbf{x}_t \in [0,1]^k$.

Predict with $\hat{y}_t = \phi\left(U \cdot \sum_{i=1}^k (w^+_{t,i} - w^-_{t,i}) \cdot x_{t,i}\right)$, where ϕ is the logistic function.

Update:

Receive the feedback $y_t \in (0,1)$.

For all $i = 1, \ldots, k$ set

$$v_i^+ = w^+_{t,i} \cdot \exp\{\eta \cdot (y - \hat{y}) \cdot x_{t,i}\},\ \ v_i^- = w^-_{t,i} \cdot \exp\{-\eta \cdot (y - \hat{y}) \cdot x_{t,i}\}$$

and

$$w^+_{t+1,i} = \frac{v_i^+}{\sum_{j=1}^k (v_j^+ + v_j^-)},\ \ w^-_{t+1,i} = \frac{v_i^-}{\sum_{j=1}^k (v_j^+ + v_j^-)}.$$

Figure 2: Algorithm $A^{(1)}_{\log}$ for learning single neurons with the logistic transfer function and the entropic loss function.

[HKW96] for algorithm $A^{(1)}_{\log}$ is

$$L_{A^{(1)}_{\log}}(\mathcal{S}) \leq \frac{4}{3} \cdot L_{\mathbf{u}}(\mathcal{S}) + \frac{U^2}{3} \cdot \ln(2k)$$

for any sequence $\mathcal{S}$ of examples where $L_{\mathbf{u}}(\mathcal{S})$ denotes the loss of the optimal neuron with weight vector $\mathbf{u}$ for which $||\mathbf{u}||_1 \leq U$.

Using the technique sketched in Section 2 the transformation of algorithm $A^{(1)}_{\log}$ into a learning algorithm $A^{(2)}_{\log}$ for depth two neural networks is quite straight forward (see Figure 3). The inputs to the original algorithm $A^{(1)}_{\log}$ are now provided by the hidden nodes, or more precisely by blocks of hidden nodes. Each block contains weight vectors of hidden nodes which have calculated the same values in all previous trials. Such a block is given by d out of N input coordinates (i.e. a projection) and a polyhedron of the corresponding weight space $\mathbf{R}^{d+1}$. The following variation of a lemma by Maass and Turán shows that all linear threshold functions (over the discrete domain) can be represented by a finite number of points in this weight space.

Lemma 3 *[MT92] Let $h : \mathbf{Z}_P^d \to \{0,1\}$ be a linear threshold function. Then there are weights $\mathbf{c} = (c_0, \ldots c_d) \in \mathbf{Z}_C^{d+1}$, where $C = (8Pd)^{3d}$, such that for all $\mathbf{y} \in \mathbf{Z}_P^d$:*

$$h(\mathbf{y}) = \begin{cases} 1 & \text{if } c_0 + \sum_{i=1}^d c_i \cdot y_i \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

Since each block represents a set of functions, its weight has to be multiplied by the number of functions in the block when a prediction has to be made. Since the exact number of functions is hard to calculate it is approximated by the volume of the block. The loss bound we can prove for algorithm $A^{(2)}_{\log}$ is given in Theorem 2(a).

3.3 Analysis of algorithm $A^{(2)}_{\log}$

For the analysis of algorithm $A^{(2)}_{\log}$ we rely on the original analysis of $A^{(1)}_{\log}$. At first observe that for a given weight-vector $\mathbf{u} \in \mathbf{R}^k$ with $||\mathbf{u}||_1 \leq U$ there is a normalized expansion into two vectors $\mathbf{u}^+$ and $\mathbf{u}^-$ in $[0,1]^k$ such that $\mathbf{u} = U \cdot (\mathbf{u}^+ - \mathbf{u}^-)$ and $||\mathbf{u}^+||_1 + ||\mathbf{u}^-||_1 = 1$. In the proof of the loss bound for $A^{(1)}_{\log}$ the distance between an expansion of the optimal weight vector $\mathbf{u}$ and the weight vectors $\mathbf{w}_t^+, \mathbf{w}_t^- \in (0,1)^k$ of the algorithm is measured by $D((\mathbf{u}^+,\mathbf{u}^-),(\mathbf{w}^+,\mathbf{w}^-)) = \sum_{i=1}^k \left[u_i^+ \ln \frac{u_i^+}{w_i^+} + u_i^- \ln \frac{u_i^-}{w_i^-}\right]$ with the convention that $0 \ln 0 = 0$. Note that the $2N$ weights of algorithm $A^{(1)}_{\log}$ are always normalized. The loss bound obtained [HKW96] for algorithm $A^{(1)}_{\log}$ is

$$L_{A^{(1)}_{\log}}(\mathcal{S}) \leq \frac{4}{3} \cdot L_{\mathbf{u}}(\mathcal{S}) + \frac{U^2}{3} \cdot \left[D(\mathbf{u}, \mathbf{w}_1) - D(\mathbf{u}, \mathbf{w}_{T+1})\right].$$

Note that this bound is equivalent to equation (2) with $b = 4/U^2$ and $a = 3/U^2$.

Now we consider algorithm $A^{(2)}_{\log}$. Recall that the weights of the optimal neural network can be translated into optimal weights $u(B)$ of the blocks $B \in \mathcal{B}$. Then the distance between the optimal weights and the weights assigned by the algorithm to the blocks in $\mathcal{B}$ is given by $D(\mathbf{u}, w, \mathcal{B}) = \sum_{B\in\mathcal{B}} \left[u^+(B)\ln\frac{u^+(B)}{w^+(B)\cdot\mathrm{vol}(B)} + u^-(B)\ln\frac{u^-(B)}{w^-(B)\cdot\mathrm{vol}(B)}\right]$. To obtain a loss bound for algorithm $A^{(2)}_{\log}$ we have to find a function f such that $D(\mathbf{u}, w, \mathcal{B}) - D(\mathbf{u}, \tilde{w}, \tilde{\mathcal{B}}) \geq f(\mathbf{u}, \mathcal{B}) - f(\mathbf{u}, \tilde{\mathcal{B}})$ (see inequality (5)), where $\tilde{\mathcal{B}}$ results from $\mathcal{B}$ by splitting some of the blocks in $\mathcal{B}$. Let B_0 and B_1 denote the blocks obtained by splitting a block $B \in \mathcal{B}$. Since these blocks inherit the weight of B we have $\tilde{w}(B_0) = \tilde{w}(B_1) = w(B)$. Furthermore $u(B) = u(B_0) + u(B_1)$. Thus

$$
\begin{aligned}
&D(\mathbf{u}, w, \mathcal{B}) - D(\mathbf{u}, \tilde{w}, \tilde{\mathcal{B}}) \\
&\quad= \sum_{B\in\mathcal{B}} \Bigg[u^+(B)\ln\frac{u^+(B)}{w^+(B)\cdot\mathrm{vol}(B)} \\
&\qquad +u^-(B)\ln\frac{u^-(B)}{w^-(B)\cdot\mathrm{vol}(B)} \\
&\qquad -u^+(B_0)\ln\frac{u^+(B_0)}{\tilde{w}^+(B_0)\cdot\mathrm{vol}(B_0)} \\
&\qquad -u^+(B_1)\ln\frac{u^+(B_1)}{\tilde{w}^+(B_1)\cdot\mathrm{vol}(B_1)} \\
&\qquad -u^-(B_0)\ln\frac{u^-(B_0)}{\tilde{w}^-(B_0)\cdot\mathrm{vol}(B_0)} \\
&\qquad -u^-(B_1)\ln\frac{u^-(B_1)}{\tilde{w}^-(B_1)\cdot\mathrm{vol}(B_1)}\Bigg] \\
&\quad\geq \sum_{B\in\mathcal{B}} [u^+(B_0)\ln\mathrm{vol}(B_0) + u^+(B_1)\ln\mathrm{vol}(B_1) \\
&\qquad +u^-(B_0)\ln\mathrm{vol}(B_0) + u^-(B_1)\ln\mathrm{vol}(B_1) \\
&\qquad -u^+(B)\ln\mathrm{vol}(B) - u^-(B)\ln\mathrm{vol}(B)]
\end{aligned}
$$

since $(a+b)\ln(a+b) \geq a\ln a + b\ln b$ for all $a, b \geq 0$. Therefore we can choose $f(\mathbf{u}, \mathcal{B}) = -\sum_{B\in\mathcal{B}}(u^+(B) + u^-(B))\ln(\mathrm{vol}(B))$ and the loss bound of algorithm $A^{(2)}_{\log}$, as expressed in equation (6), with $b = 4/U^2$ and $a = 3/U^2$, becomes

$$
\begin{aligned}
&L_{A^{(2)}_{\log}}(\mathcal{S}) \\
&\quad\leq \frac{4}{3}\cdot L_{\mathrm{opt}}(\mathcal{S}, N, d, U) \\
&\qquad +\frac{U^2}{3}\cdot[D(\mathbf{u}, \mathbf{w}_1, \mathcal{B}_1) - D(\mathbf{u}, \mathbf{w}_{T+1}, \mathcal{B}_{T+1})] \quad (7)\\
&\qquad +\frac{U^2}{3}\cdot[f(\mathbf{u}, \mathcal{B}_{T+1}) - f(\mathbf{u}, \mathcal{B}_1)]. \quad (8)
\end{aligned}
$$

Since $D(\mathbf{u}, \mathbf{w}, \mathcal{B}) \geq 0$ for all sets of blocks and weights maintained by the algorithm, (7) can be bounded by

$$
\begin{aligned}
&D(\mathbf{u}, \mathbf{w}_1, \mathcal{B}_1) - D(\mathbf{u}, \mathbf{w}_{T+1}, \mathcal{B}_{T+1}) \\
&\quad\leq D(\mathbf{u}, \mathbf{w}_1, \mathcal{B}_1) \\
&\quad= \sum_{B\in\mathcal{B}_1}[u^+(B)\ln u^+(B) + u^-(B)\ln u^-(B)] \\
&\qquad + \sum_{B\in\mathcal{B}_1}[u^+(B) + u^-(B)]\cdot\ln\left(2\binom{N}{d}\right) \\
&\quad\leq \ln\left(2\binom{N}{d}\right)
\end{aligned}
$$

(recall that the expanded weights $\mathbf{u}^+, \mathbf{u}^-$ are normalized).

To bound (8) observe that $f(\mathbf{u}, \mathcal{B}_{T+1})$ is maximal when each linear threshold function with non-zero weight is contained in a block as small as possible. The following lemma lower bounds the volume of such a block.

Lemma 4 *Let Δ be a polyhedron bounded by hyperplanes*

$$\left\{\mathbf{c} : (1, \mathbf{y}_\tau)\cdot\mathbf{c} = \frac{1}{2}\right\}$$

where all $\mathbf{y}_\tau \in \mathbf{Z}^d_P$. If Δ contains a point with integer coordinates then vol$(\Delta) \geq ((d+1)\cdot P)^{-(d+1)}$.

Proof. The distance of any integer point $\mathbf{c} \in \mathbf{Z}^{d+1}_C$ to any of the hyperplanes is at least

$$\frac{|(1, \mathbf{y}_\tau)\cdot\mathbf{c} - \frac{1}{2}|}{||(1, \mathbf{y}_\tau)||_2} \geq \frac{1}{2\cdot\sqrt{1 + dP^2}}$$

since $(1, \mathbf{y}_\tau)\cdot\mathbf{c}$ is integer. Thus the sphere of radius $\frac{1}{2\cdot\sqrt{1+dP^2}}$ around the integer point in Δ lies completely inside Δ. □

Thus we get $f(\mathbf{u}, \mathcal{B}_{T+1}) \leq (d+1)\ln[(d+1)P]$ and $-f(\mathbf{u}, \mathcal{B}_1) \leq (d+1)\ln(2C+1)$. Finally we obtain the following loss bound for algorithm $A^{(2)}_{\log}$,

$$
\begin{aligned}
L_{A^{(2)}_{\log}}(\mathcal{S}) \quad&\leq \frac{4}{3}\cdot L_{\mathrm{opt}}(\mathcal{S}, N, d, U) \\
&\quad +\frac{(d+1)U^2}{3}\cdot\ln\left[(d+1)NP(2C+1)\right].
\end{aligned}
$$

To obtain a bound on the run time of algorithm $A^{(2)}_{\log}$ we have to count the number of blocks which algorithm $A^{(2)}_{\log}$ has to maintain in trial t. The following lemma bounds the number of polyhedra in which the weight space $\mathbf{R}^{d+1}$ is split by t hyperplanes.

Lemma 5 ([Ede87, Sei95]) *The number of polyhedra obtained by dissecting $\mathbf{R}^d$ with t hyperplanes is bounded by $\binom{t}{\leq d} \leq t^d$. If the dissection with $t-1$ hyperplanes is given then the dissection with the t-th hyperplane can be constructed in $O(t^{d-1})$ time. The volumes of these polyhedra can be calculated in $O(t^d)$ time.*

Thus there are at most $\binom{N}{d}t^{d+1}$ blocks in trial t, and the dissection with the new hyperplane corresponding to the new input pattern $\mathbf{x}_t$ and the calculation of the new volumes takes $O\left(\binom{N}{d}t^{d+1}\right)$ time. This bounds the run time of algorithm $A^{(2)}_{\log}$ for trial t.

3.4 An algorithm for the linear transfer function and the square loss

For a single neuron with the linear transfer function and the square loss an on-line learning algorithm $A^{(1)}_{\text{lin}}$ was presented in [KW94]. As the algorithm for the logistic transfer function it maintains pairs of weights $w_i = (w_i^+, w_i^-)$ and the analysis makes use of the same distance function $D(\mathbf{u}, \mathbf{w})$. The loss of algorithm $A^{(1)}_{\text{lin}}$ is

$$L_{A^{(1)}_{\text{lin}}}(\mathcal{S}) \leq \frac{3}{2} \cdot L_{\mathbf{u}}(\mathcal{S}) + \frac{3U^2}{2} \cdot (D(\mathbf{u}, \mathbf{w}_1) - D(\mathbf{u}, \mathbf{w}_{T+1}))$$

where again U is an upper bound on the L_1-norm of the weight vector and $\mathbf{u}$ is the optimal weight vector with L_1-norm at most U. This algorithm can be transformed into an algorithm $A^{(2)}_{\text{lin}}$ for learning depth two neural networks analogously as $A^{(1)}_{\log}$ was transformed into $A^{(2)}_{\log}$. The loss bound that can be obtained for $A^{(2)}_{\text{lin}}$ is then

$$\begin{aligned} L_{A^{(2)}_{\text{lin}}}(\mathcal{S}) &\leq \frac{3}{2} \cdot L_{\text{opt}}(\mathcal{S}, N, d, U) \\ &\quad + \frac{3(d+1)U^2}{2} \cdot \ln\left[(d+1)NP(2C+1)\right]. \end{aligned}$$

Finally the run time of algorithm $A^{(2)}_{\text{lin}}$ is bounded in the same way as the run time of algorithm $A^{(2)}_{\log}$.

3.5 An algorithm for the thresholded transfer function and the discrete loss

In this section we consider thresholded neurons where $\phi(z) = 1$ for $z \geq 1$ and $\phi(z) = 0$ for $z < 1$. The output of such a neuron is either 0 or 1 and the performance is measured by the discrete loss. For learning such neurons algorithm WINNOW was developed [Lit88, Lit91, AW95]. Its performance again depends on an upper bound on the L_1-norm of the weight vector, but also on the separation parameter $0 < \delta \leq 1$. To achieve separation δ the weight vector $\mathbf{u}$ must be chosen such that $|\mathbf{u} \cdot \mathbf{x} - 1| \geq \delta$ for any input pattern $\mathbf{x}$. The loss of WINNOW is bounded by

$$\begin{aligned} L_{\text{WINNOW}}(\mathcal{S}) &\leq \frac{4U}{\delta} \cdot L_{\mathbf{u}}(\mathcal{S}) \\ &\quad + \frac{8}{\delta^2}\left[D(\mathbf{u}, \mathbf{w}_1) - D(\mathbf{u}, \mathbf{w}_{T+1})\right] \\ &\leq \frac{4U}{\delta} \cdot L_{\mathbf{u}}(\mathcal{S}) + \frac{8U}{\delta^2} \cdot \ln(2k) \end{aligned}$$

where $\mathbf{u}$ is the optimal weight vector with L_1-norm at most U and separation δ, and where D is the unnormalized entropic distance $D(\mathbf{u}, \mathbf{w}) = \sum_{i=1}^k \left[w_i^+ + w_i^- - u_i^+ - u_i^- + u_i^+ \ln \frac{u_i^+}{w_i^+} + u_i^- \ln \frac{u_i^-}{w_i^-}\right]$. (Here the weight vectors $(\mathbf{u}^+, \mathbf{u}^-)$ and $(\mathbf{w}^+, \mathbf{w}^-)$ are non-negative but not normalized.) This allows us to still use the same function f as in Section 3.3 and we get for the transformed algorithm WINNOW$^{(2)}$ for learning depth two neural networks that

$$\begin{aligned} & L_{\text{WINNOW}^{(2)}}(\mathcal{S}) \\ &\leq \frac{4U}{\delta} \cdot L_{\text{opt}}(\mathcal{S}, N, d, U, \delta) \\ &\quad + \frac{8(d+1)U}{\delta^2} \cdot \ln\left[(d+1)NP(2C+1)\right]. \end{aligned}$$

Since WINNOW updates its weights only when it makes a wrong prediction, the time complexity of WINNOW$^{(2)}$ for trial t is $O\left(\binom{N}{d}m_t^{d+1}\right)$ where m_t is the number of mistakes made by WINNOW$^{(2)}$ until trial t.

4 Transformation for the batch model

At first we state our results.

Definition 6 *For any loss function L and any distribution $\mathcal{D}$ on the space of examples let $L_{\text{opt}}(\mathcal{D}, N, d, U, \phi)$ denote the minimal expected loss with respect to $\mathcal{D}$ among all neural networks in $\mathcal{N}(N, d, U, \phi)$.*

The expected loss with respect to distribution $\mathcal{D}$ of a batch learning algorithm A, when given m training examples, is denoted by $L_A(\mathcal{D}, m)$.

Theorem 7

(a) For the logistic transfer function $\phi(z) = \frac{1}{1+e^{-z}}$ and the entropic loss there is a batch learning algorithm A such that

$$\begin{aligned} L_A(\mathcal{D}, m) &\leq \frac{4}{3} \cdot L_{\text{opt}}(\mathcal{D}, N, d, U, \phi) \\ &\quad + \frac{(d+1) \cdot U^2 \cdot \ln(Nm)}{m}. \end{aligned}$$

(b) For the identity function $\phi(z) = z$ and the square loss there is a batch learning algorithm A such that

$$\begin{aligned} L_A(\mathcal{D}, m) &\leq \frac{3}{2} \cdot L_{\text{opt}}(\mathcal{D}, N, d, U, \phi) \\ &\quad + \frac{3 \cdot (d+1) \cdot U^2 \cdot \ln(Nm)}{2m}. \end{aligned}$$

(c) For the threshold function $\phi(z) = 1$ for $z \geq 1$ and $\phi(z) = 0$ for $z < 1$, for separation $0 < \delta \leq 1$

and with the discrete loss, there is a batch learning algorithm A such that

$$L_A(\mathcal{D},m) \quad \leq \quad \frac{4U}{\delta}\cdot L_{\mathrm{opt}}(\mathcal{D},N,d,U,\phi,\delta) + \frac{8\cdot(d+1)\cdot U\cdot\ln(Nm)}{\delta^2 m}.$$

The run time of all algorithms is $O\left(\binom{N}{d}m^{d+2}\right)$.

To obtain these results we have to construct an algorithm which takes m input/output pairs $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m)$ and predicts the output for an $m+1$-st input pattern $\mathbf{x}_{m+1}$. To construct such an algorithm we will use a conversion technique from a special type of on-line algorithms into batch algorithms.

At first observe that we could use an on-line algorithm to predict, one after the other, the outputs $y_1,\ldots,y_m$, giving the desired output to the algorithm after each prediction. Finally, the on-line algorithm could predict y_{m+1}. Since all the input patterns $\mathbf{x}_1,\ldots,\mathbf{x}_{m+1}$ can be given to the on-line algorithm in advance, such an on-line algorithm is called a *Lookahead Prediction algorithm* [HW95]. The formal conversion of such an algorithm into a batch algorithm is a little bit more complicated and is sketched below.

A Lookahead Prediction algorithms A receives $m+1$ input patterns and then successively produces an output for each of the $m+1$ patterns. Let $L_A(\mathcal{S})$ denote the loss of algorithm A where $\mathcal{S}=\langle(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_{m+1},y_{m+1})\rangle$ is the sequence of examples. Then the corresponding batch algorithm receives m random examples (pairs of input patterns with desired outputs) which represent the hypothesis of the batch algorithm. To predict the output for a new input pattern the Lookahead Prediction algorithm is used to successively predict the outputs of the first τ stored examples, where τ is chosen at random in $\{0,\ldots,m\}$. Finally it is used to produce an output for the new pattern. A simple calculation shows that the expected loss of the batch algorithm is at most $\mathbf{E}_{\mathcal{S}}L_A(\mathcal{S})/(m+1)$, where the expectation is with respect to the $m+1$ draws that produced $\mathcal{S}$ and the random choice of τ.

To obtain such an Lookahead Prediction algorithm for depth two neural networks we note that the $m+1$ input patterns dissect the weight space into at most $\binom{N}{d}(m+1)^{d+1}$ polyhedra, see the discussion in Section 2. Recall that these polyhedra represent all linear threshold functions on the $m+1$ input patterns. Thus the Lookahead Prediction algorithm has to keep only one hidden node for each polyhedron and can use the on-line algorithms for single neurons to make its predictions and update the weights from the hidden nodes to the output node. For example we get for the the logistic transfer function and the entropic loss that

$$\begin{aligned} L_A(\mathcal{S}) &\leq \frac{4}{3}\cdot L_{\mathbf{u}}(\mathcal{S})+\frac{U^2}{3}\cdot\ln\left(\binom{N}{d}(m+1)^{d+1}\right) \\ &\leq \frac{4}{3}\cdot L_{\mathrm{opt}}(\mathcal{S})+(d+1)\cdot U^2\cdot\ln(Nm) \end{aligned}$$

where U is an upper bound on the L_1-norm of the weights from the hidden nodes to the output node and $L_{\mathrm{opt}}(\mathcal{S})$ is the loss of the neural network that best fits the sequence $\mathcal{S}$. Since $\mathbf{E}_{\mathcal{S}}L_{\mathrm{opt}}(\mathcal{S}) \leq (m+1)\cdot L_{\mathrm{opt}}(\mathcal{D},N,d,U,\phi)$ the conversion argument gives Theorem 7(a). The other parts of the theorem follow analogously. Finally, the run times of the algorithms are $O\left(\binom{N}{d}m^{d+2}\right)$ since the Lookahead Prediction algorithm performs at most $m+1$ trials and each trial has run time at most $O\left(\binom{N}{d}m^{d+1}\right)$.

5 Conclusion

We presented a technique for transforming learning algorithms for single neurons to learning algorithms for depth two neural networks in both the batch and the on-line model. The main idea is to consider a dual space in which the weight vectors of the hidden nodes are points and the examples are hyperplanes that partition this space into polyhedra. Linear threshold functions belonging to the same polyhedron classify the examples in the same manner and therefore the learning algorithms have to maintain only a single weight for each of the polyhedra. This is a quite sophisticated application of the "virtual weights" technique of Maass and Warmuth [MW95]. For example we had to show that the distance function methodology of the amortized analysis for single neurons can be adapted to handle volume approximations instead of exact counting. Our technique can handle continuous as well as discrete transfer functions at the output node and it is generalizable to any increasing differentiable transfer function and its *matching loss* [HKW96]. The identity function and the square loss plus the logistic function and the entropic loss are just commonly used special cases.

Recall that we require that the hidden nodes of the class of neural networks we are learning have constant fan-in d. The loss bounds are polynomial in this fan-in, however the time bounds are exponential in d. This may not be surprising for otherwise one of our algorithms would lead to a polynomial-time noise tolerant on-line algorithm for learning DNF formulas.

A reasonable next step would be to generalize our algorithms to allow more general transfer functions at the hidden nodes than step functions: interesting candidates are piecewise linear transfer functions or the logistic function.

Acknowledgment

Peter Auer gratefully acknowledges support by the Fonds zur Förderung der wissenschaftlichen Forschung, Austria, through grant J01028-MAT.

References

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[Ang90] D. Angluin. Negative Results for Equivalence Queries. *Machine Learning*, 2:121–150, 1990.

[AW95] P. Auer and M. K. Warmuth. Tracking the best disjunction. In *Proc. of the 36th Symposium on the Foundations of Comp. Sci.* IEEE Computer Society Press, Los Alamitos, CA, 1995.

[BGM+96] Bshouty, Goldman, Mathias, Suri, and Tamaki. Noise-tolerant distribution-free learning of general geometric concepts. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing.* 1996.

[CBLW95] 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 Transactions on Neural Networks*, 1995. To appear. An extended abstract appeared in COLT '93.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[Hau92] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78-150, September 1992.

[HKW96] D. P. Helmbold, J. Kivinen, and M. K. Warmuth. Worst-case loss bounds for sigmoided linear neurons. In *Proc. 1996 Neural Information Processing Conference*, 1996. To appear.

[HLW94] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting {0,1} functions on randomly drawn points. *Information and Computation*, 115(2):284–293, 1994.

[HW95] D. Helmbold and M. K. Warmuth. On weak learning. *Journal of Computer and System Sciences*, 50(3):551–573, June 1995.

[Jum90] G. Jumarie. *Relative information.* Springer-Verlag, 1990.

[KK92] J. N. Kapur and H. K Kesavan. *Entropy Optimization Principles with Applications.* Academic Press, Inc., 1992.

[Koi94] P. Koiran. Efficient learning of continuous neural networks. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 348–355. ACM Press, New York, NY, 1994.

[KSS92] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 341-352. ACM Press, New York, NY, 1992.

[KW94] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, University of California, Santa Cruz, Computer Research Laboratory, June 1994. An extended abstract to appeared in the STOC 95, pp. 209-218.

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

[Lit91] N. Littlestone. Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 147–156, San Mateo, CA, 1991. Morgan Kaufmann.

[Maa93] W. Maass. Bounds for the computational power and learning complexity of analog neural nets. In *Proc. 25th Annu. ACM Sympos. Theory Comput.*, pages 335–344. ACM Press, New York, NY, 1993.

[MT92] W. Maass and G. Turán. How fast can a threshold gate learn? In *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects.* MIT Press, 1992. Previous versions appeared in FOCS89 and FOCS90.

[MW95] Wolfgang Maass and Manfred K. Warmuth. Efficient learning with virtual threshold gates. In *Proc. 12th International Conference on Machine Learning*, pages 378–386. Morgan Kaufmann, 1995.

[Sei95] R. Seidel. The volume of a polyhedron. Private communication, 1995.

Parameters:

The number of inputs N, the fan-in of the hidden nodes d, the discretization parameter P, and an upper bound U on the L_1-norm of the weights from the hidden nodes to the output node.

Notations:

Let $\mathcal{B}_t$ denote the set of blocks maintained by the algorithm in trial t, and let $w_t^+(B)$ and $w_t^-(B)$ denote the weight pair of a block B in trial t.
Let $h_B(\mathbf{x})$ be the value for input pattern $\mathbf{x}_t$ calculated by the functions represented in B. (Since blocks are split when necessary this value is always well defined.)
The dissection of the blocks in $\mathcal{B}$ by a hyperplane corresponding to an input pattern $\mathbf{x}$ is denoted by $\mathcal{B} \times \mathbf{x}$.
When a block is split then the weight of the original block is assigned to both resulting parts.

Initialization:

At the beginning $\mathcal{B}_1$ contains one block for each of the $\binom{N}{d}$ projections $p : \mathbf{Z}_P^N \to \mathbf{Z}_P^d$. The polyhedron associated with each block is $[-C - \frac{1}{2}, C + \frac{1}{2}]^{d+1}$ with $C = (8Pd)^{3d}$.
The corresponding weights are set to $w_1^+(B) = w_1^-(B) = \frac{1}{2\binom{N}{d}(2C+1)^{d+1}}$.
The learning rate is set to $\eta = \frac{4}{U^2}$.

Prediction: In each trial $t \geq 1$:

Receive input pattern $\mathbf{x}_t \in \mathbf{Z}_P^N$.
Set $\mathcal{B}_{t+1} = \mathcal{B}_t \times \mathbf{x}_t$ and let $\tilde{w}_t$ denote the weights of the blocks in $\mathcal{B}_{t+1}$.
Let ϕ denote the logistic function and predict with

$$\hat{y}_t = \phi\left(U \cdot \sum_{B \in \mathcal{B}_{t+1}} \left(\tilde{w}_t^+(B) - \tilde{w}_t^-(B)\right) \cdot \mathrm{vol}(B) \cdot h_B(\mathbf{x}_t) \right).$$

Update:

Receive the feedback $y_t \in (0,1)$.
For all $B \in \mathcal{B}_{t+1}$ set

$$v^+(B) = w_t^+(B) \cdot \exp\{\eta \cdot (y - \hat{y}) \cdot h_B(\mathbf{x}_t)\},\ v^-(B) = w_t^-(B) \cdot \exp\{-\eta \cdot (y - \hat{y}) \cdot h_B(\mathbf{x}_t)\}$$

and

$$w_{t+1}^+(B) = \frac{v^+(B)}{\sum_{B' \in \mathcal{B}_{t+1}} [v^+(B') + v^-(B')] \cdot \mathrm{vol}(B')},\ w_{t+1}^-(B) = \frac{v^-(B)}{\sum_{B' \in \mathcal{B}_{t+1}} [v^+(B') + v^-(B)] \cdot \mathrm{vol}(B')}.$$

Figure 3: Algorithm $A_{\log}^{(2)}$ for learning depth two neural networks with the logistic transfer function and the entropic loss function.