A Nonlinear Error Function on Which Per-Sample BP is Faster than Batch BP*

Thomas H. Hildebrandt
Electrical Engineering and Computer Science Department
Room 304 Packard Laboratory
19 Memorial Drive West
Lehigh University
Bethlehem, PA 18015-3084
thildebr@athos.eecs.lehigh.edu

March 23, 1993

Abstract

We present a nonlinear error function with respect to which a broad class of batch Backpropagation (BP) algorithms can require arbitrarily many steps to converge. In contrast, the Best-Guess variant of Per-Sample BP converges in two steps, regardless of the initial state of the network. The error function results from the application of certain training vectors to a two-layer network with two inputs. Our results firmly establish that there exist neural network learning problems on which Per-Sample BP is faster, and lays the groundwork for a theoretical exposition of the limitations inherent in batch BP algorithms.

1 INTRODUCTION

As an implementation of the maximum gradient descent (MGD) optimization algorithm, Batch Backpropagation (Batch-BP) requires that weights in the network be updated only after all training examples have been presented. At the other extreme, per-sample backpropagation allows the weights to be changed after the presentation of each training example. Rumelhart and McClelland claim that the two algorithms will behave equivalently, "provided the learning rate ... is sufficiently small."[1, p.324] However, as we show here, the two algorithms can differ significantly in terms of their convergence rates, even though they approach the same theoretical minimum.

*submitted to IEEE T. on Neural Networks.

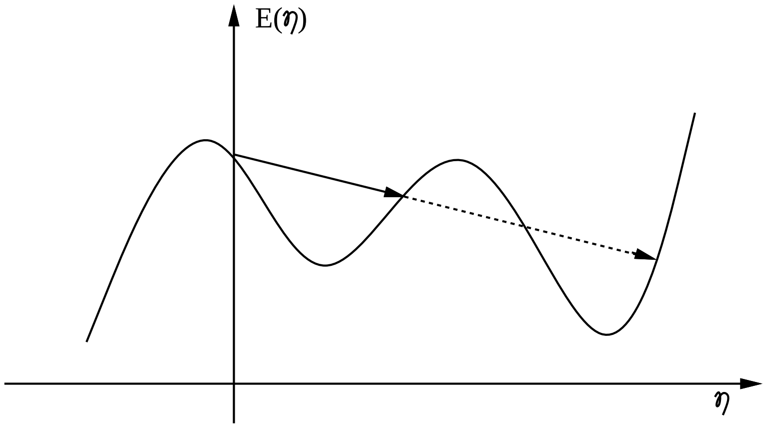

Figure 1: An example of tunnelling. The move represented by the solid arrow is legal; the move represented by the dashed arrow is not.

For this comparison, we choose Batch Backpropagation with Line Search (Batch-BPLS) which includes a line search in order to find the true minimum in the direction of the maximum gradient. In fact, our result is independent of the batch algorithm used as long as it

(a) computes the current step with respect to the gradient of the total error function,

(b) makes steps which lie along a straight line,

(c) cannot accept a new state at which the energy is higher than the current state, and

(d) is not permitted to "tunnel" (i.e. it may not pass through a region where the energy is higher than the current state).

The first three assumptions are incorporated in Batch-BPLS by definition. The last one forces the line search to find the local minimum rather than the global minimum in the direction of the gradient (see Figure 1). Several different batch minimization algorithms are discussed in [2, 3, 4].

For comparison, we define a Best-Guess version of the per-sample BP algorithm (BGBP) which analytically locates the expected position of the minimum in the direction of the current per-sample gradient. As with the batch version, the per-sample BP algorithm with line search can more precisely locate the minimum (in the direction of the gradient with respect to the current sample) in the presence of nonlinearity. On the other hand, a line search represents a considerable computational burden if multiple probes are made for each training example. In contrast, the computational burden of the best-guess algorithm is fixed. With respect to the nonlinear error function presented here, the best-guess and line-search versions of the per-sample algorithm behave identically.

After defining these algorithms, we construct a nonlinear error function which is easily analyzed. At the same time, we show that this function results from a very simple neural circuit by the application of carefully chosen training vectors. We proceed to show that the parameters of this function can easily be chosen so that arbitrarily many steps are required by the batch algorithm to reach the minimum. In contrast, we show that the per-sample algorithm finds the minimum in two steps, regardless of the initial state of the network. We thus establish that situations in which Per-Sample BP converges faster are likely to arise in real problems.

2 DEFINITIONS

2.1 Neural Network Model

A neural network accepts as input the vector $\mathbf{x} = [x_1, x_2, \ldots, x_N]^T$ and produces the output scalar $y(\mathbf{x})$ as an approximation to the target output function $t(\mathbf{x})$. In the standard model[1], the network is composed of simple, independent computational elements, each of which computes its output o_j by taking the weighted sum of its inputs o_i and applying some bounded, monotonic "squashing" function $f(\cdot)$ to the result[1]:

$$o_j = f(v_j) \ , \quad v_j = \sum_i w_{ji} o_i \ . \tag{1}$$

The weights w_{ji} contained in each neuron constitute the adjustable parameters of the approximation. In this correspondence, we consider a single neuron, so we may write the input as the vector

[1]Thoughout the text, the notation $\sum_i \cdot$ is used to denote "the sum over all i of ...". The limits of the summation should be clear from context.

$\mathbf{x}$ and the weight set as the vector $\mathbf{w}$. Thus, for the sole output neuron (1) becomes

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$$

in vector notation.

2.2 Batch Backpropagation

Backpropagation implements the Maximum Gradient Descent (MGD) algorithm to minimize some cost function $E(\mathbf{w})$. The usual cost function is the sum over all training vectors of the error between the target and actual output functions with respect to each training vector $\mathbf{x}_p$.

$$E = \sum_p E_p \ ,$$

$$E_p = \frac{1}{2}[t(\mathbf{x}_p) - y(\mathbf{x}_p)]^2 \ . \tag{2}$$

Thus, Backpropagation uses Maximum Gradient Descent to find the Minimum Mean Squared Error solution.

During each iteration, the Batch Backpropagation algorithm seeks to reduce the error by changing the weights in the direction of the gradient at the current value of weight set.

$$\Delta\mathbf{w} \propto -\left.\frac{\partial E}{\partial \mathbf{w}}\right|_{\mathbf{w}(t)} \ . \tag{3}$$

Due to the linearity of partial differentiation, we can write

$$\Delta\mathbf{w} = -\eta \sum_p \left.\frac{\partial E_p}{\partial \mathbf{w}}\right|_{\mathbf{w}(t)} \ , \tag{4}$$

where η determines the step size. This can be rewritten as

$$\Delta\mathbf{w} = \sum_p -\eta \Delta_p \mathbf{w} \ , \tag{5}$$

where

$$\Delta_p\mathbf{w} = \left.\frac{\partial E_p}{\partial \mathbf{w}}\right|_{\mathbf{w}(t)} \ . \tag{6}$$

This partial derivative is easily evaluated by applying the chain rule:

$$\frac{\partial E_p}{\partial \mathbf{w}} = -[t(\mathbf{x}_p) - y(\mathbf{x}_p)]f'(\mathbf{w}^T\mathbf{x}_p)\mathbf{x}_p \ . \tag{7}$$

Since in the batch version of BP this partial derivative must be evaluated with respect to each training example in order to determine the direction of the total gradient, the calculation of one step corresponds to a training "epoch". After the old weight set is replaced by the new weight set $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}$, the process is repeated until the error has been reduced to a tolerable value.

For more details on the general case including the adjustment of weight belonging to hidden layer neurons, consult Rumelhart and McClelland[1, pp.325–327].

2.3 Backpropagation with Line Search

The procedure defined above leaves unspecified the length η of the step to be taken each time $\mathbf{w}$ is updated. Once the direction of the update has been determined, we can assume that the local quadratic approximation is valid, and calculate where the minimum should lie in that direction. That is the idea behind Best-Guess Backpropagation. In the presence of nonlinearities, however, it is likely that the position of the true minimum in the direction of the gradient will lie nearer or farther from the current weight set $\mathbf{w}(t)$.

The idea behind Backpropagation with Line Search (BPLS) is to find the minimum in the direction of the current estimate of the gradient using a line search method such as Newton's Method. For the present purpose, we merely require that the line search find a point that lies within the same basin (assumption (d) above).

2.4 Per-Sample Best-Guess BP

Here, we define the Per-Sample BP algorithm, and derive the value of η required to take a best-guess step. The best guess is the expected location of the minimum derived analytically. This analysis assumes that the linear equation for the gradient evaluated at $\mathbf{w}(t)$ is valid globally.

In contrast to the Batch algorithm, which computes a step with respect to the total error E, Per-Sample BP computes a step with respect to the error E_p associated with each training pattern $\mathbf{x}_p$. Thus, a per-sample step becomes

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta_p \mathbf{w} \ , \tag{8}$$

where $\Delta_p \mathbf{w}$ is given by (6).

We can evaluate the gradient with respect to a given training sample p using the chain rule:

$$\frac{\partial E_p}{\partial \mathbf{w}} = -[t(\mathbf{x}_p) - f(\mathbf{w}^T \mathbf{x}_p)] f'(\mathbf{w}^T \mathbf{x}_p) \mathbf{x}_p \ . \tag{9}$$

Substituting this into (6), we obtain

$$\Delta_p \mathbf{w} = \eta [t(\mathbf{x}_p) - f(\mathbf{w}(t)^T \mathbf{x}_p)] f'(\mathbf{w}(t)^T \mathbf{x}_p) \mathbf{x}_p \ . \tag{10}$$

Using the substitution

$$e(t) = [t(\mathbf{x}_p) - f(\mathbf{w}(t)^T \mathbf{x}_p)] \ , \tag{11}$$

we can write any weight set $\mathbf{w}$ along this line as

$$\mathbf{w} = \mathbf{w}(t) + \Delta_p \mathbf{w} = \mathbf{w}(t) + \eta e(t) f'(\mathbf{w}(t)^T \mathbf{x}_p) \mathbf{x}_p \ . \tag{12}$$

Thus, we can write an equation for the error along this line as a function of η:

$$E_p(\eta) = \frac{1}{2} [t(\mathbf{x}_p) - f((\mathbf{w}(t) + \eta e(t) f'(\mathbf{w}(t)^T \mathbf{x}_p) \mathbf{x}_p)^T \mathbf{x}_p)]^2 \ . \tag{13}$$

To find the minimum, we take the partial derivative of this with respect to η, and set the result to zero:

$$\begin{aligned} \frac{\partial E_p}{\partial \eta} &= -[t(\mathbf{x}_p) - f((\mathbf{w}(t) + \eta e(t) f'(\mathbf{w}(t)^T \mathbf{x}_p) \mathbf{x}_p)^T \mathbf{x}_p)] \\ &\quad \cdot f'((\mathbf{w}(t) + \eta e(t) f'(\mathbf{w}(t)^T \mathbf{x}_p) \mathbf{x}_p)^T \mathbf{x}_p) e(t) f'(\mathbf{w}(t)^T \mathbf{x}_p) \mathbf{x}_p)^T \mathbf{x}_p = 0 \ . \end{aligned} \tag{14}$$

Due to the zero on the right-hand side, we may eliminate the minus sign and the factors generated by the chain rule expansion so that only the bracketed portion of the expression remains:

$$[t(\mathbf{x}_p) - f((\mathbf{w}(t) + \eta e(t) f'(\mathbf{w}(t)^T \mathbf{x}_p) \mathbf{x}_p)^T \mathbf{x}_p)] = 0 \ . \tag{15}$$

In order to solve for η, we replace the squashing function $f(\cdot)$ with a first-order Taylor Series expansion

$$f(v) = f(v(t)) + f'(v(t))\Delta v \ , \tag{16}$$

where $v(t) = \mathbf{w}(t)^T\mathbf{x}_p$ and $\Delta v = \eta e(t) f'(\mathbf{w}(t)^T\mathbf{x}_p)\mathbf{x}_p^T\mathbf{x}_p$. Combining this with (15), we obtain

$$t(\mathbf{x}_p) - f(\mathbf{w}(t)^T\mathbf{x}_p) = \eta e(t)[f'(\mathbf{w}(t)^T\mathbf{x}_p)]^2\|\mathbf{x}_p\|^2 \ . \tag{17}$$

Recognizing the left side to be equal to $e(t)$, this reduces to

$$\eta = \frac{1}{[f'(\mathbf{w}(t)^T\mathbf{x}_p)]^2\|\mathbf{x}_p\|^2} \ . \tag{18}$$

Having given both the direction and step size for the BGBP algorithm, one cycle of the algorithm with respect to training vector $\mathbf{x}_p$ has been completely defined. It is usual to take a step with respect to each training example in turn until every sample has been seen. This completes one training epoch.

3 A BATCH-PERVERSE FUNCTION

In the previous section, we defined Batch-BPLS and derived the equation for a Per-Sample Best-Guess step. In this section, we use those definitions to demonstrate an error function on which the Per-Sample version converges immediately, while the Batch version can be forced to require an arbitrarily large number of steps to converge.

The function which we wish to consider is constructed as follows. As usual,

$$E = \sum_p E_p$$

where here, $p \in 1, 2$. The training vectors are

$$\mathbf{x}_1 = [0 \ a_1]^T, \quad \mathbf{x}_2 = [a_2 \ a_3]^T$$

with target values of $t(\mathbf{x}_p) = 0$ for $p = 1, 2$. The structure of the network is given by

$$o_1 = [w_1 x_1]^2, \quad o_2 = w_2 x_2 \ ,$$

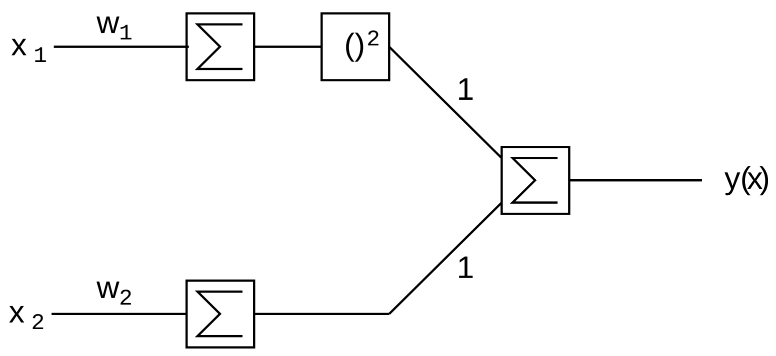

Figure 2: The neural circuit used.

$$y(\mathbf{x}) = 1([1\ 1][o_1\ o_2]^T)\ ,$$

where $1(\cdot)$ — the squashing function attached to the output neuron — is the identity function. A representation of the corresponding neural circuit is shown in Figure 2. Evaluating the error functions, we find that

$$E_1 = \frac{1}{2}[0 - a_1 w_2]^2, \quad E_2 = \frac{1}{2}[0 - a_3 w_2 - (a_2 w_1)^2]^2\ .$$

The total error function, being the sum of these, is given by

$$E = \frac{1}{2}\left([a_1 w_2]^2 + [a_3 w_2 + (a_2 w_1)^2]^2\right)\ . \tag{19}$$

A plot (Figure 3) of the $E = 1/2$ contour of this function with parameters $a_1 = 0.25$, $a_2 = 1$, and $a_3 = -2$ shows it to be a canyon whose deepest point is at the origin, and which curves upward (in the w_2 direction) as w_1 is increased. From its appearance, it is obviously a non-linear error function.

3.1 The Batch Solution

In this section, we analyze the behavior of the batch algorithm on the given function. A simulation of the training algorithm which merely counts the number of steps required for convergence to a specified tolerance does not provide us with any intuition as to how the count will change when various parameters are altered. Hence, we will establish a lower bound on the number of steps required for convergence of Batch BPLS.

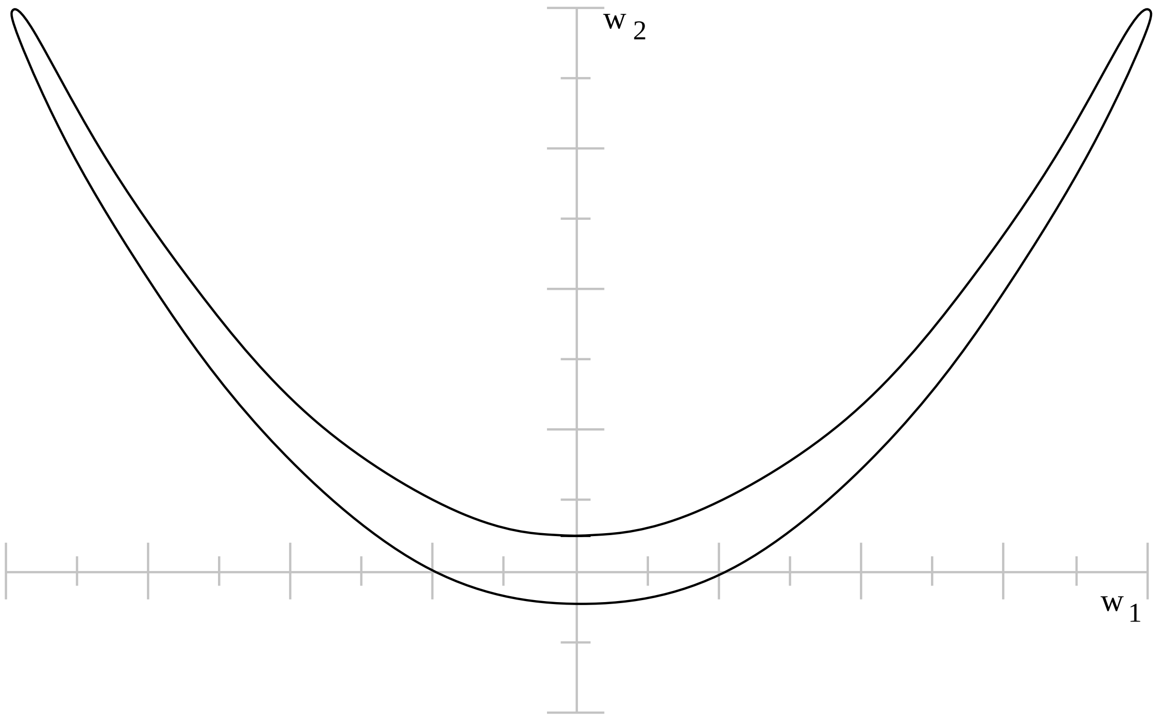

Figure 3: A plot of the $E = 1/2$ iso-error contour of the function with $a_1 = 0.25$, $a_2 = 1$, and $a_3 = -2$.

We demonstrate the lower bound by recalling that the algorithm cannot accept a step $\Delta\mathbf{w}$ which will increase the total error $E(\mathbf{w})$ at any point between $\mathbf{w}(t)$ and $\mathbf{w}(t+1)$. This assumption is true by definition for BPLS with respect to the error surface being considered, as long as a_1 is sufficiently small. In terms of more general error functions and algorithms, however, it requires that BPLS choose a new value for $\mathbf{w}$ which lies in the same basin as defined by $E(t) = E(\mathbf{w}(t))$ (i.e. "tunneling" is not allowed). In the most general interpretation, BPLS does not obey this restriction. However, it is uncertain that a better minimum exists in a basin other than the current one, and the current gradient provides no information as to where such a basin might be, or whether it exists at all. Therefore, resorting to random probes along the current gradient line will most often prove fruitless, and are better omitted from the algorithm. Consequently, the usual behavior of the algorithm will be to remain in the current basin, justifying assumption (d) above.

Due to the curvature of the walls of the canyon described by our function, then, the size of the largest step that can be taken from a given starting point is limited. The narrower the canyon, the smaller this maximum step size. Thus, by choosing the parameters of the function (i.e. the components of the training vectors) appropriately, we can force the batch algorithm to consume arbitrarily many steps to reach the minimum.

To simplify the analysis, we choose a_1 to be negligible in comparison to a_2 and a_3. This means that the parabolic canyon becomes the dominant feature, and that the bottom of the canyon slopes very little as one proceeds away from the origin in the direction of $[0\ w_2]^T$. Now, let us derive the iso-error curve which passes through the point $E_0 = E(\mathbf{w}(0))$ on the resulting error surface. Any step that the algorithm takes must lie entirely inside this curve. The curve is found by intersecting $E(\mathbf{w})$ with the plane $E = E_0$. Thus, for $a_1 = 0$ we have

$$E = E_0 = \frac{1}{2}[a_3 w_2 + (a_2 w_1)^2]^2 \ . \tag{20}$$

whence

$$(a_2 w_1)^2 + a_3 w_2 = \pm\sqrt{2E_0} \ ,$$

or

$$w_2 = -\frac{a_2^2}{a_3}w_1^2 \pm \frac{\sqrt{2E_0}}{a_3} \ . \tag{21}$$

We recognize this to be the equation for two identical parabolae, separated in the w_2 direction by a distance of $\sqrt{2E_0}/a_3$.

Now, noting that the initial weight lies on this iso-error curve, the largest step the algorithm can take (completely ignoring the direction of the gradient) must lie along the line which passes from a point on the outer curve to another point on the outer curve and is just tangent to the inner curve. Let the point of tangency on the inner curve be at $[\phi_1 \ \phi_2]$. Immediately, we have

$$\phi_2 = -\frac{a_2^2}{a_3}\phi_1^2 - \frac{\sqrt{2E_0}}{a_3} \ . \tag{22}$$

We know that the slope of the tangent line is

$$\frac{dw_2}{dw_1} = -2\frac{a_2^2}{a_3}\phi_1 \ , \tag{23}$$

whence

$$\begin{aligned} w_2 &= \phi_2 + \frac{dw_2}{dw_1}(w_1 - \phi_1) \\ &= \phi_2 - 2\frac{a_2^2}{a_3}\phi_1(w_1 - \phi_1) \ . \end{aligned} \tag{24}$$

Equating the right side of this equation with that of (21) and substituting (22), we can find the points w_1 at which this line intersects the outer curve.

$$\begin{aligned} -\frac{a_2^2}{a_3}\phi_1^2 - \frac{\sqrt{2E_0}}{a_3} - 2\frac{a_2^2}{a_3}(\phi_1 w_1 - \phi_1^2) &= \frac{\sqrt{2E_0}}{a_3} - \frac{a_2^2}{a_3}w_1^2 \\ \frac{a_2^2}{a_3}(w_1^2 - 2\phi_1 w_1 + \phi_1^2) &= \frac{2\sqrt{2E_0}}{a_2} \\ (w_1 - \phi_1)^2 &= \frac{2\sqrt{2E_0}}{a_2^2} \\ w_1 &= \phi_1 \pm \frac{\sqrt{2\sqrt{2E_0}}}{a_2} \ . \end{aligned} \tag{25}$$

From this, we see that the maximum step size in the w_1 direction is given by

$$\Delta_{\max} w_1 = 2\frac{\sqrt{2\sqrt{2E_0}}}{a_2} \ . \tag{26}$$

Note that this step size is constant with respect to the initial weight set $\mathbf{w}(0)$, since parabolae have constant curvature. We can thus bound from below the number of steps the algorithm will require to reach the minimum, by dividing the initial w_1 by this maximum step size:

$$\text{Steps}_{\text{min}} = \frac{w_1}{\Delta_{\max} w_1} = \frac{w_1 a_2}{2\sqrt{2\sqrt{2E_0}}} \ . \tag{27}$$

From this equation it becomes clear that, regardless of the initial values of w_1 and w_2, we can force the algorithm to take arbitrarily many steps to reach the origin by adjusting the value of a_2 to be as large as desired.

The picture that emerges is that, for small a_1, large a_2, and large a_3, batch updates can require arbitrarily many steps to converge, under the restriction that no step is allowed to increase the total error anywhere along its length.

3.2 The Per-Sample Solution

The value of $\mathbf{w}$ which drives E_1 to zero is $w_2 = 0$, independent of the value of w_1. This is exactly the step found by the best-guess per-sample algorithm, where we have the gradient

$$\left.\frac{\partial E_1}{\partial \mathbf{w}}\right|_{\mathbf{w}(0)} = [[0 \ a_1^2]\mathbf{w}_0 \ 0]^T$$

The step is given by

$$\Delta_1 \mathbf{w} = -\eta[[0 \ a_1^2]\mathbf{w}_0 \ 0]^T \ ,$$

where, noting that the derivative of $f(x)$ is

$$\frac{\partial}{\partial x} 1(x) = 1 \ ,$$

we find that

$$\eta = \frac{1}{||\mathbf{x_1}||^2} = \frac{1}{a_1^2} \ .$$

This makes

$$\Delta_1 \mathbf{w} = [-w_1 \ 0]^T \ ,$$

which sets the first component of $\mathbf{w}(1)$ to zero. Continuing in this way, we find that the second component of $\mathbf{w}(2)$ is likewise set to zero. Thus, the per-sample algorithm finds the exact minimum of the function in two steps, regardless of the initial values of w_1 and w_2.

4 DISCUSSION

The function we chose is special in the sense that once E_1 is minimized, updates with respect to other training samples $\mathbf{x}_p$ do not disturb this minimum. In other words, the partial of E_1 with respect to $\mathbf{w}$ is perpendicular to the partial with respect to $\mathbf{w}$ of all other E_p when these partials are evaluated at the point $\mathbf{w}_1$ which minimizes E_1. In our example, it is possible to disrupt this property by choosing different training examples, for instance, by making the second component of $\mathbf{x}_1$ be nonzero. In the more general case, the per-sample algorithm may exhibit considerable "cross-stitching" in its search for the minimum, especially if the training vectors are nearly collinear as perceived by the neuron being updated. On the other hand, per-sample updates are able to accept moves which increase the total error, and our example clearly shows that this can represent a considerable advantage when nonlinear error functions are being considered.

4.1 Computational Burden

Our result would be of little consequence if the per-sample algorithm were computationally much more expensive than batch algorithms. In fact, the computational burdens for an epoch of each are comparable, as shown in Table 1.

If each neuron has multiple ($\gg 1$) inputs, the vector operations dominate. Taking one vector scalar product plus one vector addition to be equal in computational burden to a vector inner product, we see that an epoch of the per-sample algorithm requires the equivalent of $3p + \delta$ vector inner products, where δ represents the cost of the scalar operations. For Batch-BPLS, we can summarize the computational burden as approximately $2(p + q)$ vector inner products per epoch. Note that the computations required by the line search can be considerable, depending on the

Table 1: Computational requirements of the Per-Sample and Batch algorithms.

Operation	Per-Sample	Batch
Vector Inner Products	$2p$	$p+q$
Vector Scalar Products	p	$p+q$
Vector Additions	p	$p+q$
Evaluations of $f(\cdot)$	p	$p+q$
Evaluations of $f'(\cdot)$	p	p
Scalar Multiplications	$3p$	$p+q$
Scalar Divisions	p	q
Scalar Differences	p	$p+2q$

p is the number of training samples.
q is the number of iterations during line search.

strictness of the convergence criterion. Still, the per-epoch computational burden of Per-Sample BGBP is comparable to that of Batch-BPLS.

4.2 Extension to Other Nonlinear Functions

In this paper, we chose to examine a parabolic nonlinearity, due to its nice analytical properties: It is easy to derive the equation for the points of intersection between a line and a parabola. Furthermore, it has constant curvature, so one can estimate the number of steps required simply by dividing the initial w_1 by the computed maximum step size. However, the basic idea of the proof can easily be extended to other nonlinear functions. As long as the function has nonzero curvature over a finite domain and the initial and optimal points lie at opposite ends of this interval, one can choose the separation between the bounding iso-error curves (through an appropriate choice of training vectors) so that arbitrarily many steps are required by a batch algorithm obeying the anti-tunnelling restriction. The property which needs to be preserved in order for the per-sample algorithm to converge immediately is that the direction of the current step be orthogonal to all preceding steps.

4.3 Implications for Neural Network Training

Based on our analysis, we may make the following observations: Initial weights which are close to the solution help considerably. Training vectors with very large components are clearly troublesome, and training vectors which are nearly collinear give rise to intense "cross-stitching". This suggests that transforming the problem so that the origin lies in the midst of the training examples can have beneficial results.

Violation of the "anti-tunnelling" restriction is essential to more rapid convergence, and also to finding the global minimum regardless of the initial weights. For example, minimization based on Simulated Annealing can tunnel, and finds the global minimum with probability 1, provided that the initial temperature is high enough and the annealing schedule is gradual enough.

Overall, due to the presence of "cross-stitching" in both versions of the basic BP algorithm, neither converges very rapidly. Methods such as Conjugate Gradient (CG) — which accumulate information on the Hessian — are generally likely to converge more rapidly when "cross-stitching" is the dominant behavior. On the other hand, CG becomes more batch-like as it proceeds, and will become limited in regard to its step size if it cannot accept an increase in the total error. The behavior of CG can be improved if the algorithm can accurately determine when to abandon all or part of its current estimate. This observation has been corroborated experimentally by Møller[5].

5 CONCLUSION

We have shown that for a certain neural network there exist sets of training vectors and initial weights which will allow the per-sample version of backpropagation to converge much faster than a batch training algorithm. From our analysis, we observe that

1. It is nice to have initial weights which are close to the solution.

2. If the problem can be transformed so that the origin lies in the midst of the training examples, then convergence can be improved.

3. If the network contains a nonlinearity with nonzero curvature over finite interval, then there exist sets of training vectors on which its convergence can be arbitrarily bad.

4. Algorithms which can accept an increase in the total error can converge faster in the presence of nonlinearities.

Observation 3 has very strong implications on the implementation of artificial neural networks. It suggests that piecewise linear squashing functions are superior for training purposes. Experimental data in support of this observation have been presented by Lo and Bavarian[6]. Ongoing research addresses this observation, specifically examining the characteristics of the sigmoid function.

References

[1] Rumelhart, David E. and J.L. McClelland, **Parallel Distributed Processing**, MIT Press (1986).

[2] Kramer, Alan H. and Alberto Sangiovanni-Vincentelli, "Efficient parallel learning algorithms for neural networks", in **Advances in Neural Information Processing Systems 1**, Morgan-Kaufmann, San Mateo, pp.40–48 (1989).

[3] Rohwer, Richard, "Time trials on second-order and variable-learning-rate algorithms", in **Advances in Neural Information Processing Systems 3**, Morgan-Kaufmann, San Mateo, pp.977–983 (1991).

[4] Saarinen, S., R. Bramley and George Cybenko, "The numerical solution of neural network training problems", CSRD Report 1089, Center for Supercomputing Research and Development, University of Illinios, Urbana, IL 61801-2932 (1991).

[5] Møller, Martin Fodslett, "Supervised learning on large redundant training sets", in **Proceedings of Neural Networks for Signal Processing 2**, IEEE, pp.79–92, (1992).

[6] Lo, Zhen-Ping and Behnam Bavarian, "Comparison of a neural network and a piecewise linear classifier", *Pattern Recognition Letters*, 12(11), pp.649–655 (1991).

[7] Corwin, Lawrence J. and Robert H. Szczarba, **Calculus in Vector Spaces**, Marcel Dekker (1979).

[8] Kuan, Chung-Ming and Kurt Hornik, "Convergence of learning algorithms with constant learning rates", *IEEE T. on Neural Networks*, vol. 2, pp. 484–489, Sep. 1991.