Back Propagation
with
Discrete Weights and Activations

Guy Smith
William H. Wilson
June 1989

Discipline of Computer Science,
Flinders University of South Australia,
Bedford Park 5042
AUSTRALIA

© G.M. Smith & W.H. Wilson, 1989
The Flinders University of South Australia

Back Propagation with Discrete Weights and Activations

Guy Smith
William H. Wilson
June 1989

School of Mathematical Sciences
1 Introduction

In this report a variation of Rumelhart, Hinton and William's Error Back Propagation algorithm\(^1\) [3] [4] is described. This variation generates discrete values for weights and activations. The performance of the variation in learning a variety of mappings is compared with the performance of the standard algorithm. The mappings learnt and the techniques for measuring performance are described.

This variation is motivated by the desire to build Neural Nets which are cheap to implement in hardware. The variation described here generates discrete (not integral) weights and activations. Even though the weights and activations are not integral, the activation flow in the completed net could be implemented using integer arithmetic with a scaling factor. No attempt is made to remove real arithmetic while the net is still learning.

There is little or no mention in the literature of variations of standard Back Propagation which generate discrete weights and activations. Hopfield Nets [2] generate discrete weights and activations, but have quite a different network structure and learning algorithm.

2 Description of Algorithm

2.1 Standard Back Propagation

This section describes the standard Back Propagation algorithm. Readers familiar with it may wish to skip this section. Those wanting more detail are referred to [3].

The Standard Back Propagation algorithm assumes that an acyclic directed graph (the neural net) and a set of data vector / target vector pairs (the environment mapping) exist. All the data vectors should be of the same length, which should be the same as the number of data nodes in the net\(^2\). Likewise all the result vectors should be the same length, which should be the same as the number of result nodes in the net.

The algorithm describes how activation flows through the net, and how to modify the net's attributes to learn any given mapping. Each node of the net has the (real) attributes activation, stimulus and error. Each arc has the (real) attributes weight and change. The mapping defined by the net is completely determined by the weight attributes of the arcs.

The net interacts cyclically with the environment. Each cycle, the environment presents the net with a data vector; the net returns a result vector and

---

\(^1\)Hereafter called standard Back Propagation

\(^2\)Data nodes are those nodes with no input arcs. Result nodes are those with no output arcs. There are always some such nodes because the net is a finite acyclic directed graph.
finally, if the net is learning, the environment presents the net with an error vector (the difference between the target vector and the result vector).

Within the net, only the arc attributes are preserved between cycles; the attributes of nodes are cleared between cycles. At the beginning of a cycle, the activation attribute of each data node is set to the value of an element of the data vector. The activation of a non-data node is defined as follows: the dot product of the vector of input weights and the vector of input activations is the stimulus of the node. The squashing function (see §2.2.1) is applied to stimulus, giving activation. The error attribute of the result nodes is set from elements of the error vector received from the environment. The error of a non-result node is the derivative of its squashing function multiplied by the dot product of its output arcs' weights and output nodes' activations.

An epoch consists of many cycles, during which a selection of the data vectors are presented to the net. At the beginning of each epoch, the change attribute of the arcs is zeroed. During the cycles of the epoch, the change attribute stores the desirable direction of change of the weight attribute of the arc. At the end of the epoch, each arc increments (or decrements, if change ≤ 0) weight by an amount proportional to change.

The change attribute is accumulated thus: each cycle, it is incremented by the product of the error of the node after it and the activation of the node before it.

There is strong mathematical support for this algorithm. Consider the dot product of the error vector with itself as a scalar function measuring of the error of the result vector for a particular data vector. The error attribute is the derivative of the error with respect to the activation of that node. The increment to change is the derivative of the error with respect to the weight attribute of an arc. Thus, the change attribute at the end of an epoch is the derivative of the sum of the errors for the data vectors presented in that epoch. Thus, the Standard Back Propagation algorithm is merely a gradient descent method of minimising the squared-error of the net.

2.2 Changes to Standard and Discrete Algorithm

Later in this report, I compare the results of simulations of standard Back Propagation and the modified algorithm. In this section, I describe some details of these simulations.

---

3 Thus the only information kept between epochs is the weight attribute, which determines the net's mapping.
2.2.1 Activation Range

Standard Back Propagation uses the squashing function \( f_{std}(x) = \frac{1}{1 + e^{-x}} \). This gives \( f_{std}(x) \) the open interval \((0.0, 1.0)\). In this report I use the squashing function \( f_{sym}(x) = 2f_{std}(x) - 1 \). This gives the squashing function, and hence the node activations, the interval \((-1.0, 1.0)\). I find this improves the performance of Back Propagation considerably — I suspect much less effort is expended learning biases.

\[
\begin{array}{c|c|c|c}
 f_{std}(x) & f_{sym}(x) \\
 1 & 1 \\
 z & z \\
 -1 & -1 \\
\end{array}
\]

2.2.2 Phases of an Epoch

In the simulations, each epoch contained three phases: learning, weight update and testing.

During the learning phase, a subset of the legal data vectors were presented to the net, the error values back propagated through the net and the weight derivatives summed. The subset presented varies from epoch to epoch. In the mappings learnt, the data vector / target vector pairs break up naturally into several categories. The subset presented consists of one data vector from each category, each vector being chosen at random from its category.

During the weight update phase, all the weights were modified according to their summed derivatives, which were then set to zero.

During the testing phase, data vectors were presented to the net until all legal vectors had been presented, or until an incorrect result vector was generated by the net. If all result vectors were correct, then the net had learnt the mapping, and the simulation ceased.

I judged a result vector to be correct if all its elements were correct. An element whose target was 1.0 was correct if it was greater than 0.2. Likewise an element whose target was -1.0 was correct if it was less than -0.2. This is the 40-20-40 criterion advocated in [1]. No mapping in this report had a target vector with elements not equal to ±1, so this criterion sufficed.

2.2.3 Net Structure

The nets simulated in this report had one hidden layer. For the Xor mappings they contained 4 nodes, for the Stereo3 mapping4 36 nodes, for the Stereo5

---

4See §4.2 for a description of the Stereo mappings
mapping 100 nodes. In each case these were more nodes than were necessary to learn the mapping. My main interest was to compare standard Back Propagation and the discrete variant, not optimise their performance.

The result layer of the net had arcs from both the hidden layer and the data layer.

To allow nodes in the hidden and result layers to generate a bias, an input with a constant value of 1.0 was appended to the input vector of every node.

### 2.3 Variation giving Discrete Weights and Activations

The objective of the algorithm was to generate a net that needed only discrete (rather than real) arithmetic for activation flow. There was no attempt to eliminate real arithmetic while the net is learning. Figure 1 shows the modifications needed to the code for the algorithm.

The variation from standard Back Propagation was simple. Two additional parameters were needed: `steps_weight` and `steps_actvn`. These parameters controlled the 'graininess' of the discrete values. The value of `steps_weight` was the number of legal arc weight values per integer. Thus if `steps_weight = 2` then \[-1.5, -1, -0.5, 0, 0.5, 1, 1.5, \ldots\] were the allowed values for weights. A value of zero for `steps_weight` meant continuous values were to be used for weights. `Steps_actvn` similarly restricted the legal values for node activations.

The summed inputs to a node were not restricted to discrete values explicitly, but implicitly by being the sum of products of discrete values.

To get these discrete weight and activation values, the weight (or activation) value was calculated in the normal way, giving a real value. Where the real value fell exactly on a discrete value, no rounding was necessary. However, the real value usually fell between two legal discrete values. It was then rounded to one of these discrete values, chosen thus:

- **During learning**, the real value may have been rounded to either of the adjacent discrete values, with the nearest one being most likely. The chance of rounding to one of the adjacent discrete values was proportional to the real value's distance from the other adjacent discrete value. For example if `steps_weight = 4` and a weight value of 0.3 was calculated using normal Back Propagation, there was a 80% chance it was rounded to 0.25 and a 20% percent chance it was rounded to 0.5. This allowed small weight derivatives to have a chance of altering their weight.

- **During testing**, the real value was rounded to the nearest of the discrete values. This removed any randomness from testing the net.

These changes introduce only one major problem to Back Propagation: zero values of the activation derivatives. These occur only when the activation has the extreme values -1 or 1. This can not occur in standard Back Propagation
Figure 1: Pseudocode Illustrating Modification to Algorithm

STANDARD ALGORITHM:

proc CalcActvn( node )
    ...
    Actvn := f( stimulus );
    ...

proc ModifyWeight( arc )
    ...
    new_weight := old_weight + rate * summed_derivs
    ...

MODIFIED ALGORITHM

func discrete( x ,steps ) returns real
    if testing_phase then
        ran := 0.5
    else
        ran := random() (* 0<ran<1 *)
    end if
    discrete_x := x * steps
    new_discrete_x := floor( discrete_x + ran )
    return new_discrete_x / steps (* real division *)
end func

proc CalcActvn( node )
    ...
    Actvn := discrete( f( stimulus ) ,steps_actvn );
    ...

proc ModifyWeight( arc )
    ...
    new_weight_tmp := old_weight + rate * summed_derivs
    new_weight := discrete( new_weight_tmp ,steps_weight )
    ...
because the squashing function asymptotically approaches but never reaches \pm 1.0 However, it can occur in the modified Back Propagation.

To solve this problem, the function which calculated the activation derivative replaced an extreme value of activation with the average of the extreme value and the adjacent legal value. For example, if \texttt{steps\_actvn} = 2, \( f'(\text{-1}) \) was replaced by \( f'(\text{-0.75}) \). This modification improved the performance of the algorithm considerably for small values of \texttt{steps\_actvn}.

3 Measuring Technique

In measuring the performance of an algorithm on a mapping, I used the median learning time over many simulations. I preferred the median to the mean for two reasons, both related to the usual distribution of epochs required for learning.

Usually, the distribution has one peak. It drops off quickly to the left (fewer epochs), often with no trials requiring an order of magnitude fewer epochs than the mode (see Figure 2). However, it trails off slowly to the right with many trials requiring twice the mode. It often extends to an order of magnitude greater than the mode, and may extend to infinity.

The value of the mean can be dominated by extreme values of the tail, and becomes meaningless if some trials take infinite epochs (i.e. do not converge on a solution). Even the use of an arbitrary maximum number of epochs for any one trial does not fully learn this mapping, since the choice of this maximum influences the mean.

The value of the median, however, is unaffected by the shape of the tail. Once a trial has exceeded the median, it can be cut off without affecting the value of the median. In practice, one has to estimate when a trial has exceeded
the median, and this estimation may influence the final value.

Hence, using the median of the trials rather than the mean increases the usefulness of the measure, and reduces the computational expense of simulating the net. The measurements quoted in this report are the median of at least 40 trials, using a cutoff value of two times the current median for the Xor mappings, and three times for the Stereo mappings.

4 Mappings to Learn

4.1 Xor Mappings

The simplest mapping learnt was the Xor1 mapping. The data vector in the has two elements \(<D1, D2>\) and the target vector is \(<D1 \neq D2>\).

The second mapping was the Xor4 mapping. It resembles the Xor1 mapping, but has some noise added to its input. The data vector is split into two fields: the Bit1 and Bit2 fields. The Bit1 and Bit2 fields have four nodes each, and represent either a 0 or 1 input to the XOR function. They have either three or four nodes on (representing 1), or three or four nodes off (representing 0). The target vector is \(<(\text{Bit1} \neq \text{Bit2})>\). For example, the data vector \(<1,0,1,1,0,0,0,0>\) represents the inputs \((1,0)\) and has a target vector of \(<1>\).

There are 100 legal data vectors for the Xor4 mapping. Each field has 5 ways of representing a 0 (or 1) input — no permutations or any one node permuted. This gives \(25 = 5^2\) ways to represent each one of the four inputs to the XOR function.

Both the Xor mappings present four data vectors to the net for learning each epoch — representing each of the \((0,0), (0,1), (1,0), (1,1)\) inputs to the XOR function.

4.2 Stereo Mappings

The object of this mapping is to measure the stereo offset between fields in the data vector. The data vector contains two fields, each of \(N\) nodes. The first field contains any combination of 0's and 1's (except all 0's or all 1's). The second field contains the same pattern as the first, but rotated by \(M\) nodes where \(0 \leq M < N\). The target vector contains \(N\) elements, which are all 0 except for the \(M^\text{th}\) which is 1.

The field width \(N\) must be chosen so that each data vector unambiguously defines a rotation \(M\). For example, the data vector with the fields "0101" "1010" might be rotated by either 1 or 3 elements. If \(N\) is a prime number, all data vectors do unambiguously define a rotation.

In each epoch \(N\) data vectors are presented to the net for learning, one for
each different offset.

In this report I use $N = 3$ and $N = 5$, which have 18 and 150 legal data vectors respectively. In general, the number of data vectors is $N(2^N - 2)$.

For this mapping, I allow 4 hidden nodes for each combination of nodes from the two fields, i.e. $4N^2$ nodes.

5 Comparative runs

Standard Back Propagation is represented by values of zero for the parameters steps_weight and steps_actvn.

5.1 Fine Grained Weights and Activations

I tested the algorithm with parameters designed to determine at what graininess the number of epochs required to learn a mapping was affected. The parameter steps_weight took the values 0, 1000, 100 and 10. Likewise the parameter steps_actvn took the values 0, 1000, 100, 10. This gave 16 combinations for each of the Xor1, Xor4, Stereo3 and Stereo5 mappings. None of these mappings showed significant variation in epochs required over these parameters (see Figure 3).

5.2 Coarse Grained Weights and Activations

I also tested the algorithm with very grainy parameters: varying steps_weight and steps_actvn from 10 to 2. The behaviour of the algorithm as these parameters varied was similar for the different mappings, but was markedly different in the size of the effect. For all mappings except Xor1, the algorithm’s performance degraded slowly at first, but increasingly quickly, as either parameter was reduced from 10 towards 1. The more complex the mapping, the faster this degradation seemed to be (see Figure 4).

5.3 Imposing a Maximum Weight

I was curious about the effect of imposing a maximum magnitude on the size of the weights, thus limiting them to a finite set of values. I devised another parameter, max_weight, to represent the maximum legal magnitude of a weight. On testing the algorithm varying this parameter I again found, as the parameters moved from fine grained to coarse grained, the performance would degrade slowly at first but increasingly quickly. All three parameters appear to interact: the less

5Insufficient trials with Stereo5 were run to include the results, but those that were run agree with this statement.
Figure 3: Fine Grained Activations and Weights

The area of each box in the figures is proportional to the median number of epochs required to learn the mapping. The areas have been normalised in each figure so that a $1cm^2$ box represents the highest number of epochs. Since the axis grids have $1cm$ spacing, some boxes will touch (but not overlap). Where the largest number of epochs in a figure is a round number (e.g., 2000), this means the net usually failed to learn the mapping with the parameters of the largest box(es).
Figure 4: Coarse Grained Activations and Weights

Xor4
steps-actvn 20000 epochs = 1cm²

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

steps-weight

Stereo3
steps-actvn 1403 epochs = 1cm²

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

steps-weight
Figure 5: Effect of \textit{steps\_weight} and \textit{max\_weight} on Xor4

\begin{align*}
\text{steps\_actvn} &= 10 \\
\text{max-weight} &\quad 2000 \text{ epochs} = 1cm^2
\end{align*}

\begin{align*}
\text{steps\_actvn} &= 7 \\
\text{max-weight} &\quad 2000 \text{ epochs} = 1cm^2
\end{align*}

\begin{align*}
\text{steps\_actvn} &= 4 \\
\text{max-weight} &\quad 2000 \text{ epochs} = 1cm^2
\end{align*}
Figure 6: Effect of $steps\_weight$ and $max\_weight$ on Stereo3

$steps\_actvn = 10$

$max\_weight \quad 116 \text{ epochs} = 1cm^2$

$steps\_actvn = 7$

$max\_weight \quad 142 \text{ epochs} = 1cm^2$

$steps\_actvn = 4$

$max\_weight \quad 1351 \text{ epochs} = 1cm^2$
grainy the value of two of the parameters, the grainier the third parameter could be before the performance dropped quickly. This is seen clearly in Figures 5 and 4, but is less clear in Figure 6. Note that for large (e.g., 9, 12) values of max_weight there may be no weights trying to grow larger than max_weight, and thus the performance figures will be the same as for max_weight = 0.

5.4 Xor1 Mapping is Anomalous

The behaviour of the algorithm with very grainy parameters differed for the Xor1 mapping in comparison to the other mappings. Specifically, the performance often actually improved as the parameters were made grainier (see Figures 7 and 8).

6 Conclusions

A variation of standard Back Propagation which restricts weights and activations to discrete (not necessarily integral) values has been tested, and found to work. Restricting weights and activations to as few as 10 evenly spaced values between integers does not affect the learning rate of the algorithm. Restricting weights and activations to even grainier discrete values results in accelerating degradation of performance of the learning algorithm. Restricting weights to a maximum magnitude of 12 does not seriously affect the performance of the algorithm, but further restrictions do result in accelerating degradation of performance.

This algorithm was tested over four mappings, and for very grainy trials the well-known XOR mapping produced results differing qualitatively to those from
Figure 7: Effect of steps_weight and max_weight on Xor1

<table>
<thead>
<tr>
<th>max-weight</th>
<th>steps_actvn = 1 (105 epochs = 1cm²)</th>
<th>max-weight</th>
<th>steps_actvn = 4 (2000 epochs = 1cm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

steps_weight

steps_actvn = 7 (2000 epochs = 1cm²)

max-weight

steps-weight
Figure 8: Coarse Grained parameters with Xor1

steps-actvn  123 epochs = 1\text{cm}^2

\begin{figure}
\centering
\begin{tabular}{ccccccccccc}
1 & & & & \cdot & & \cdot & & \cdot & & \cdot \\
2 & & & & \cdot & & \cdot & & \cdot & & \cdot \\
3 & & & & \cdot & & \cdot & & \cdot & & \cdot \\
4 & & & & \cdot & & \cdot & & \cdot & & \cdot \\
5 & & & & \cdot & & \cdot & & \cdot & & \cdot \\
8 & & & & \cdot & & \cdot & & \cdot & & \cdot \\
10 & & & & \cdot & & \cdot & & \cdot & & \cdot \\
\end{tabular}
\end{figure}

the other mappings.

References


