

Kruschke, J. K., and Movellan, J. R. (1991).
Benefits of gain: speeded learning and minimal
hidden layers in back-propagation networks.
*IEEE Transactions on Systems, Man,
and Cybernetics*, 21, 273-280.

Benefits of Gain: Speeded Learning and Minimal Hidden Layers in Back-Propagation Networks

John K. Kruschke and Javier R. Movellan

Abstract—The gain of a node in a connectionist network is a multiplicative constant that amplifies or attenuates the net input to the node. The objective of the work is to explore the benefits of adaptive gains in back propagation networks. First it is shown that gradient descent with respect to gain greatly increases learning speed by amplifying those directions in weight space that are successfully chosen by gradient descent on weights. Adaptive gains also allow normalization of weight vectors without loss of computational capacity, and we suggest a simple modification of the learning rule that automatically achieves weight normalization. Finally, a method for creating small hidden layers by making hidden node gains compete according to similarities between nodes, with the goal of improved generalization performance, is described. Simulations show that this competition method is more effective than the special case of gain decay.

I. THE GAIN PARAMETER IN BACK PROPAGATION

The back propagation learning algorithm [1]–[5] has become a very popular method for training connectionist networks. Two of the appealing properties of back propagation are its tolerable learning speed and its ability to generalize to novel inputs. Unfortunately back propagation is sometimes too slow, and

Manuscript received December 10, 1988; revised June 30, 1990.

J. K. Kruschke was with the Department of Psychology, University of California, at Berkeley, Berkeley, CA. He is now with the Department of Psychology, Indiana University, Bloomington, IN 47405.

J. R. Movellan was with the Department of Psychology, University of California at Berkeley, Berkeley, CA. He is now with the Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA 15213.

IEEE Log Number 9039978.

generalization is not always good. In this article we introduce a new parameter, gain, into back propagation networks and show that it can yield benefits for learning speed and generalization.

Consider a multilayer feed-forward network, as in standard back propagation. Let a_i^s be the activation of the i th node of layer s , and let $\mathbf{a}^s = [a_1^s \cdots a_n^s]^T$ be the column vector of activation values in layer s . The input layer is layer 0. Let w_{ij}^s be the weight on the connection from the j th node in layer $s-1$ to the i th node in layer s , and let $\mathbf{w}_i^s = [w_{i1}^s \cdots w_{in}^s]^T$ be the column vector of weights from layer $s-1$ to the i th node of layer s . The net input to the i th node of layer s is defined as $\text{net}_i^s = \langle \mathbf{w}_i^s, \mathbf{a}^{s-1} \rangle = \sum_k w_{ik}^s a_k^{s-1}$, and let $\mathbf{net}^s = [\text{net}_1^s \cdots \text{net}_n^s]^T$ be the column vector of net input values in layer s . The activation of a node is given by a function of its net input,

$$a_i^s = f(g_i^s \text{net}_i^s), \quad (1)$$

where f is any function with a bounded derivative, and g_i^s is a real number called the gain of the node.

Suppose that for a particular input pattern, \mathbf{a}^0 , the desired output is the teacher pattern $\mathbf{t} = [t_1 \cdots t_n]^T$, and the actual output is \mathbf{a}^L , where L denotes the output layer. Define an error function on that pattern, $E = (1/2) \sum_i (t_i - a_i^L)^2$. The overall error on the training set is simply the sum, across patterns, of the pattern error E . We then perform gradient descent on E with respect to w_{ij}^s . The chain rule yields

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^s} &= \frac{\partial E}{\partial \mathbf{net}^{s+1}} \cdot \frac{\partial \mathbf{net}^{s+1}}{\partial a_i^s} \cdot \frac{\partial a_i^s}{\partial \text{net}_i^s} \cdot \frac{\partial \text{net}_i^s}{\partial w_{ij}^s} \\ &= [-\delta_1^{s+1} \cdots -\delta_n^{s+1}] \cdot \begin{bmatrix} w_{1i}^{s+1} \\ \vdots \\ w_{ni}^{s+1} \end{bmatrix} \cdot f'(g_i^s \text{net}_i^s) g_i^s a_j^{s-1}, \quad (2) \end{aligned}$$

where $\delta_i^s = -\partial E / \partial \text{net}_i^s$. In particular, the first three factors of (2) indicate that

$$\delta_i^s = \left(\sum_k \delta_k^{s+1} w_{ki}^{s+1} \right) f'(g_i^s \text{net}_i^s) g_i^s. \quad (3)$$

The recursive formula (3) for δ_i^s is the same as in standard back propagation [1], [2] except for the appearance of the gain parameter. Combining (2) and (3) yields the learning rule for weights:

$$\Delta w_{ij}^s = \epsilon_w \delta_i^s a_j^{s-1}, \quad (4)$$

where ϵ_w is a small positive constant called the “step size” of gradient descent with respect to weights.

Gradient descent on error *with respect to the gains* can also be computed. Using the chain rule as previously, it is easy to compute that

$$\frac{\partial E}{\partial g_i^s} = \left(\sum_k \delta_k^{s+1} w_{ki}^{s+1} \right) f'(g_i^s \text{net}_i^s) \text{net}_i^s. \quad (5)$$

Then

$$\Delta g_i^s = \epsilon_g \delta_i^s \text{net}_i^s / g_i^s, \quad (6)$$

where ϵ_g is the step size of the gains. The learning rule for gains (6) is easily incorporated into standard back propagation programs. In particular, all the quantities that appear in (6) are locally available to the affected gain g_i^s .

An equivalent method was first introduced by Movellan [6] and independently proposed by Tawel [7]. Other authors (e.g.,

TABLE I
RATIOS OF AVERAGE LEARNING TIMES (IN SWEEPS) FOR BP RELATIVE TO BPG^a

Number Hidden Nodes	Criterion = 0.20				
	Weight Step Size, ϵ_w				
	0.25	0.5	1.0	2.0	4.0
2	1.9	1.7	2.1(1/1)	1.3(1/2)	2.3(8/11)
4	1.3	1.4	1.8	1.8	5.3
8	1.5	1.4	1.5	1.3	1.4
16	1.5	1.4	1.3	1.3	1.2
32	1.6	1.4	1.3	1.3	1.2
Number Hidden Nodes	Criterion = 0.10				
	Weight Step Size, ϵ_w				
	0.25	0.5	1.0	2.0	4.0
2	3.0	2.4	2.4(1/1)	1.4(1/3)	3.5(8/11)
4	1.9	1.8	2.0	1.9	4.9
8	1.9	1.5	1.7	1.4	1.3
16	2.0	1.6	1.4	1.4	1.2
32	2.2	1.7	1.5	1.3	1.2
Number Hidden Nodes	Criterion = 0.05				
	Weight Step Size, ϵ_w				
	0.25	0.5	1.0	2.0	4.0
2	5.7	5.1	4.3(1/1)	1.6(1/3)	3.5(8/11)
4	4.2	3.7	2.9	2.2	4.7
8	4.1	2.6	2.2	1.5	1.4
16	4.2	2.8	1.7	1.5	1.3
32	4.6	3.3	1.8	1.4	1.3

^aWhen applied to the 4-4 encoder problem. Numbers in parentheses indicate the number of times (out of 15) that BP or BPG failed to reach criterion in 1000 sweeps.

[8]–[10]) have also used a gain parameter, with different purposes and different architectures. These are discussed later.

Throughout this article, we will refer to back propagation with adaptive gains as BPG, and to standard back propagation, for which $g_i^s = 1$ for all i, s and $\epsilon_g = 0.0$, as BP.

II. SPEEDED LEARNING FROM GRADIENT DESCENT WITH GAIN

Learning time in BPG is remarkably faster than standard BP. In this section we discuss some criteria for comparing learning times of different algorithms, and then report our simulation procedure and results.

Comparing Learning Speeds

Comparing the speeds of variants of back propagation can be tricky. One must control all of the following variables: updating procedure (every trial vs. periodic), solution criteria, initialization procedure, learning rates, number of hidden nodes, and error function. In our comparisons of BP and BPG, we performed gradient descent on the sum of squared errors, weight and gain changes were accumulated after each pattern presentation but only executed at the end of a full sweep (or "epoch") through the training set, and learning time was measured as the number of sweeps until $|t_i - a_i^L| < \epsilon_{crit}$, for a pre-set critical value ϵ_{crit} , across all patterns and output nodes i .

Since the initial values of the weights may affect convergence, it is common practice to run several simulations with different starting weights and to combine the results with appropriate statistics. The choice of these statistics is not straightforward. Arithmetic averages are excessively influenced by the occasions when the network fails to reach criterion in the allotted time, but robust statistics, like the median, do not reflect those occasions at all. In our simulations we characterized speed with two statistics: the arithmetic average of the number of sweeps when the solution was achieved in less than 1000 sweeps; and,

the number of simulations when the system needed more than 1000 sweeps.

Combining the results for different step sizes is also tricky. One possibility is to compare learning times for optimal step-sizes. However, it is often preferable to learn moderately fast over a wide range of step sizes than to learn exceptionally fast over a very limited range. In this article we report learning times for a variety of parameter combinations. We also report averages of the different combinations to give an estimate of the relative efficiencies of the algorithms.

Procedure

We compared BP and BPG on two standard benchmark problems, the exclusive-or (XOR) and the 4-4 encoder problem [1]. We tested learning times of BP and BPG on all combinations of the following independent variables.

- Number of hidden nodes: 2, 4, 8, 16, 32.
- Step size ϵ_w for weight change: 0.25, 0.5, 1, 2, 4.
- Critical value (ϵ_{crit}) for solution: 0.20, 0.10, 0.05.

There was a single hidden layer, in which all nodes used the logistic activation function $a = 1/(1 + \exp(-g \text{ net}))$. In each condition 15 different initializations for weights were used, with weights assigned according to a uniform random distribution from -1 to $+1$, such that the fan-in weight vector (including bias) of each node was normalized to Euclidean length 1. The step sizes for weights in the output layer were scaled according to the standard fan-in correction formula (Plaut *et al.*, [10]).

In BPG, gains were modified according to (6) with $\epsilon_g = 0.20$. In the XOR problem, gain was restricted such that $0.75 < g < 3.50$, so that gains would be neither too small, making the node ineffective, nor too large, producing excessive change. Momentum (see [1]) was set to 0.9 for both weights and gains. To assure that our program was correct, its performance, with $\epsilon_g = 0.0$ and $g = 1$, was compared to McClelland and Rumelhart's [11] back-propagation software. The programs behaved identically to the fifth decimal place.

TABLE II
RATIOS OF AVERAGE LEARNING TIMES (IN SWEEPS) FOR BP RELATIVE TO BPG^a

Number Hidden Nodes	Criterion = 0.20 Weight Step Size, ϵ_w				
	0.25	0.5	1.0	2.0	4.0
2	1.2(7/3)	1.4(0/2)	3.2(0/1)	1.8(0/2)	1.2
4	1.2(2/2)	2.0	3.8	2.7	1.4
8	1.6(2/2)	1.5	2.3	3.1	1.6
16	1.4(4/1)	1.3(1/0)	1.8	3.0	3.0
32	1.3(8/2)	1.4	1.4	1.9	3.6

Number Hidden Nodes	Criterion = 0.10 Weight Step Size, ϵ_w				
	0.25	0.5	1.0	2.0	4.0
2	1.3(7/3)	1.5(0/2)	3.3(0/1)	1.8(0/2)	1.3(0/1)
4	1.3(4/2)	2.1	3.9	2.8	1.4
8	1.7(3/2)	1.6	2.5	3.1	1.6
16	1.5(6/1)	1.5(1/0)	2.0	3.1	3.1
32	1.2(11/3)	1.6(2/0)	1.6	2.2	3.7

Number Hidden Nodes	Criterion = 0.05 Weight Step Size, ϵ_w				
	0.25	0.5	1.0	2.0	4.0
2	1.5(8/3)	1.9(0/2)	3.5(0/1)	1.8(0/2)	1.3(0/1)
4	1.5(8/2)	2.5(1/0)	4.5	3.0	1.5
8	1.8(10/2)	2.0(2/0)	3.2	3.5	1.7
16	1.4(4/3)	1.8(4/0)	2.6	3.8	3.4
32	1.3(15/7)	1.6(9/0)	2.3	3.0	4.2

^aWhen applied to the XOR problem. Numbers in parentheses indicate the number of times, out of 15, that BP/BPG failed to reach criterion within 1000 sweeps.

TABLE III
RATIOS OF LEARNING TIMES FOR BP AND BPG^a

Criterion	Encoder	XOR	Average
0.05	2.9	2.4	2.7
0.1	1.9	2.1	2.0
0.2	1.7	2.0	1.8
Average	2.2	2.2	2.2

^aSummarized from Tables II and III.

Results

Tables I and II show the learning times of BP relative to BPG for the 4-4 encoder problem and the XOR problem. Average results are summarized in Table III. For all trials when criterial error was reached, BPG was faster than BP. The learning time ratios vary from 1.1 to 5.7, with an average of 2.2. In terms of the number of times that criterial error was not reached in 1000 sweeps, BPG also performed better than BP except when two hidden nodes were used. In that case BPG could not solve the problem 13% of the time, as opposed to BP that did not solve it 11% of the time. Table III shows the average speed ratio as a function of the learning criteria and the type of problem. The average learning time ratio is 1.83 for a criterion of 0.2, and 2.68 for a criterion of 0.05. In a previous study [6] it was shown that for smaller criteria, BPG is more than 6 times faster than BP.

III. DISCUSSION

Speeded Learning by Other Methods

There have been many other back propagation speed-up methods described in the literature, e.g., Fahlman's [12] Quick-Prop, or Jacob's [13] learning-rate adjustment heuristics. We do not claim that BPG is faster or incompatible with these methods. An advantage of adaptive gain is that it is easy to introduce into back propagation programs, and that it accelerates learning without the need to invoke principles other than gradient de-

scent. Adaptive gains also provide a variety of other benefits that we will discuss in later sections.

Relation of Gain Change and Weight Change

In this section we explore the relationship between gain and weight change. We then analyze why BPG speeds up learning relative to BP.

Lemma 1: Suppose weights are changed by gradient descent on the error E , and gains are also changed by gradient descent on E . Then, for a given node, the magnitude of its gain increases if and only if the length of its weight vector increases.

Proof: Gradient descent on the error E with respect to weights and gains can be expressed as follows (cf. (2)-(6):

$$\Delta w_i^s = \epsilon_w \frac{\partial E}{\partial (g_i^s \text{net}_i^s)} g_i^s a^{s-1} \quad (7)$$

$$\Delta g_i^s = \epsilon_g \frac{\partial E}{\partial (g_i^s \text{net}_i^s)} \langle w_i^s, a^{s-1} \rangle \quad (8)$$

Solving (7) for a^{s-1} and substituting into (8) yields

$$\Delta g_i^s = \frac{\epsilon_g}{\epsilon_w} \frac{1}{g_i^s} \|w_i^s\| \|\Delta w_i^s\| \cos \theta \quad (9)$$

where θ is the angle between w_i^s and Δw_i^s . Note also that

$$\Delta \|w_i^s\|^2 = \|\Delta w_i^s\|^2 + 2\|w_i^s\| \|\Delta w_i^s\| \cos \theta. \quad (10)$$

Case 1: $\cos \theta \geq 0$. From (9), $\Delta g_i^s \cdot g_i^s > 0$. Therefore, the signs of g_i^s and Δg_i^s are the same, so $\Delta |g_i^s| > 0$. From (10), $\Delta \|w_i^s\|^2 > 0$. In this case, then, the signs of $\Delta |g_i^s|$ and $\Delta \|w_i^s\|$ agree: They both are positive.

Case II: $\cos \theta < 0$. From (9), $\Delta|g_i^s| < 0$, by argument analogous to that used in Case I. From Equation 10, we can see that $\Delta\|w_i^s\|^2 < \|\Delta w_i^s\|^2$. For arbitrarily small¹ step sizes, $\|\Delta w_i^s\|^2 \rightarrow 0$, so that $\Delta\|w_i^s\|^2 < 0$. Again the signs of $\Delta|g_i^s|$ and $\Delta\|w_i^s\|^2$ agree: They both are negative.

Adaptive gain has a catalytic effect in the learning process by modifying the magnitude, not the direction, of the weight change. Empirical observations show that in the early phases of learning, when a successful direction has not yet been found, weight vectors change direction from trial to trial and both gain and weight-vector length shrink. As successful directions are found, weights and gains grow larger. From (3) and (4), gain can be seen as modulating the stepsize ϵ_w of the weight change, amplifying learning in nodes that find successful directions in weight space. This may help to explain why BPG does not necessarily provide a big advantage with very small networks: Larger networks provide a larger pool of candidate directions, so that the benefits of gain are more likely to be utilized.

Gain and Weight Normalization

By including the gain parameter in back propagation, one can also normalize weight vectors without loss of computational power. One advantage of normalization is that it maintains the weight values within bounds [14], [15]. Normalization can also be used to facilitate comparison between nodes [16]. Weight vectors can be factored into length and direction, with direction indicated by the normalized weight vector, and length indicated by the gain of the node. By performing gradient descent on gain, and by implication from Lemma 1, the effects of back propagation on the lengths of the weight vectors will be reflected by the gains, even with the weight vectors themselves normalized.

Several approaches might be used to normalize the weight vectors. One possibility is to let the weights change via back propagation and then to normalize the weights directly, dividing them by the length of the weight vector. Such an approach was taken by Kruschke [16]. Another possibility, used by Oja [15] in the context of a Hebbian rule, is to generate a Taylor series approximation to the normalization process. Movellan [17] derived a Taylor series approximation to normalization for back propagation:

$$\Delta w_{ij}^s = \epsilon_w \delta_i^s (a_j^{s-1} - \text{net}_i^s w_{ij}^s), \quad (11)$$

where Δw_{ij}^s is the modified weight change. Notice that the normalized rule of (11) is nearly the same as the standard weight update rule of (4) but with a corrected activation from the layer below. An advantage of the Taylor approximation over "brute-force" normalization is that the length of the weight vector need not be computed. We have applied the normalization rule of (11) in conjunction with BPG, with good results [17].

IV. MINIMAL HIDDEN LAYERS FROM GAIN COMPETITION

We have shown that the gain parameter is useful for speeding up learning and for allowing weight vector normalization. We will now show that it can also be used to create hidden layer bottlenecks; i.e., hidden layers with relatively few participating

nodes. Small hidden layers have been shown to improve the generalization performance of back-propagation networks in some applications.

Gain Indicates Participation in Representation and Learning

We say that a given hidden node *participates in representing* the input if and only if its activation changes for some change of input. The gain of a node indicates how much the node participates in representing the input. When the gain of a node is zero, its activation is constant: If $g_i = 0$, then $a_i = f(g_i \text{net}_i) = f(0)$ for all net_i , and as the magnitude of the gain grows, the variance of activation values from $f(0)$ increases for monotonic f .

The gain of a node also indicates the extent to which the node participates in learning. When the gain of a node is zero, the error propagated to its weights is also zero, and hence its weights cannot learn. When the magnitude of the gain grows, the learning by weights also increases. That can be seen explicitly from (3), in which gain acts as a continuous modulator of the magnitude of error propagated to the weights.

The gain parameter is not just a passive indicator like the pointer of a speedometer. If one tries to go faster by moving the pointer of a speedometer to a higher value, one only succeeds in breaking the speedometer. On the contrary, if we adjust the gain of a node, we actually change its activation variance and its weight learning rate. It is the causal efficacy of the gain parameter that makes it an attractive device for introducing constraints into back propagation.

Improved Generalization from Small Hidden Layers

There is empirical evidence that generalization to novel input patterns is improved by using hidden layers with a small number of nodes (e.g. [18]–[20]). In these cases, generalization from the training set to novel inputs was better when the number of hidden nodes was relatively small.² The reason for improved generalization is intuitively clear: A small hidden layer forces the input patterns to be mapped through a low-dimensional space, enforcing proximities between hidden-layer representations that were not necessarily present in the input-pattern representations. Only the differences between patterns that are most important for decreasing error will be preserved as large distances between hidden layer patterns. Differences between input patterns that are not preserved in the hidden layer representation are thereby generalized over completely. A more detailed discussion can be found in Kruschke [16], [22], [23]. Theoretical and empirical results regarding learnability, generalization, and network size can be found in, e.g., [24]–[26].

Given the desirability of small hidden layers, why not just build a network with an arbitrarily small number of nodes? First, for most learning applications it is not known what the minimal number of hidden nodes is. If one uses too few nodes, then the network has insufficient computational power to learn the training set to criterion accuracy, regardless of learning algorithm used. Second, back-propagation learning is slower, and more likely to encounter local minima or extensive plateaus on the error surface, when small hidden layers are used. For example, the minimal number of hidden nodes required to learn the XOR is two, but when only two nodes are built into the network, extended learning times are often encountered. In general, back-propagation learning is faster when larger hidden layers are used [10]. We try to satisfy these two competing constraints on hidden layer size by building a network with a

¹For non-infinitesimal changes, suppose that $\|\Delta w_i^s\| = p\|w_i^s\|$, for some $0 < p < 1$. Then we must have $\cos \theta < -p/2$ for the conclusion to hold. For small p , that condition is virtually the same as $\cos \theta < 0$. In other words, for non-infinitesimal changes, the conclusion of the lemma can be violated when the change vector Δw_i^s is at an angle just slightly obtuse to the weight vector ($-p/2 \leq \cos \theta < 0$) or when it happens to be huge relative to the weight vector ($p > 1$). Empirically, such conditions occur so ephemeral as to be negligible.

²There are also cases for which reducing the number of hidden nodes did not improve generalization (e.g., [21]). It is beyond the scope of this article to analyze in detail which particular applications will benefit from hidden layer bottlenecks. The premise is that at least some applications do benefit from bottlenecks.

