Benefits of Gain:
Speeded learning and minimal hidden layers
in back-propagation networks.*

John K. Kruschke†
Department of Psychology
Indiana University
Bloomington, IN 47405
kruschke@ucs.indiana.edu

Javier R. Movellan
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213
jm2z+@andrew.cmu.edu

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 this article is to explore the benefits of adaptive gains in back propagation networks. First we show 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. Adpative 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, we describe 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. Simulations show that this competition method is more effective than the special case of gain decay.

* In press: *IEEE Transactions on Systems, Man and Cybernetics.* Scheduled to appear January 1991 (vol. 21, no. 1). Originally submitted November 1988.

† The order of authors is alphabetical. Requests for reprints may be sent to either author. Most of the work reported here was carried out when both authors were at the Department of Psychology, University of California at Berkeley.

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 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 multi-layer 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 $\mathbf{a}^s = [a_1^s \ldots a_n^s]^T$ 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 $\mathbf{w}_i^s = [w_{i1}^s \ldots w_{in}^s]^T$ 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 $\mathrm{net}_i^s = \langle \mathbf{w}_i^s, \mathbf{a}^{s-1} \rangle = \sum_k w_{ik}^s a_k^{s-1}$, and let $\mathbf{net}^s = [\mathrm{net}_1^s \ldots \mathrm{net}_n^s]^T$ 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,

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

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 $\mathbf{t} = [t_1 \ldots t_n]^T$, and the actual output is $\mathbf{a}^L$, where L denotes the output layer. Define an error function on that pattern, $E = \frac{1}{2}\sum_i (t_i - a_i^L)^2$. 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:

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

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

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

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

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

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 above, it is easy to compute that

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

Then

$$\Delta g_i^s = \epsilon_g \delta_i^s \mathrm{net}_i^s / g_i^s, \tag{6}$$

where ϵ_g is the step size of the gains. The learning rule for gains (Eqn. 6) is easily incorporated into standard back propagation programs. In particular, all the quantities that appear in Equation 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., [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 $g_i^s = 1\ \forall i, s$ 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 $|t_i - a_i^L| < ecrit$, for a pre-set criterial value $ecrit$, 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 straight-forward. 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.
- Criterial value ($ecrit$) for solution: 0.20, 0.10, 0.05.

There was a single hidden layer, in which all nodes used the logistic activation function $a = 1/(1+\exp(-g\,\mathrm{net}))$. 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 the standard fan-in correction formula (Plaut et al., [10]).

In BPG, gains were modified according to Equation 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.

Results.

Tables 1 and 2 show the learning times of BP relative to BPG for the 4-4 encoder problem and the XOR problem. Average results are sumarized in Table 3. 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 2 hidden nodes were used. In that case BPG could not solve the problem 13% of the time, as opposed to BP which did not solve it 11% of the time. Table 3 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.

criterion=0.20

number hidden nodes	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

criterion=0.10

number hidden nodes	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

criterion=0.05

number hidden nodes	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

Table 1. Ratios of average learning times (in sweeps) for BP relative to BPG, when 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.

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] QuickProp, 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 descent. Adaptive gains also provide a variety of other benefits that we will discuss in later sections.

The 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.

criterion=0.20

		weight step size, ϵ_w				
		0.25	0.5	1.0	2.0	4.0
number hidden nodes	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

criterion=0.10

		weight step size, ϵ_w				
		0.25	0.5	1.0	2.0	4.0
number hidden nodes	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

criterion=0.05

		weight step size, ϵ_w				
		0.25	0.5	1.0	2.0	4.0
number hidden nodes	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

Table 2. Ratios of average learning times (in sweeps) for BP relative to BPG, when applied to the XOR encoder problem. Numbers in parentheses indicate the number of times, out of 15, that BP/BPG failed to reach criterion within 1000 sweeps.

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. Eqns. 2–6):

$$\Delta \mathbf{w}_i^s = -\epsilon_w \frac{\partial E}{\partial (g_i^s \mathrm{net}_i^s)} g_i^s \, \mathbf{a}^{s-1} \tag{7}$$

criterion	*Encoder*	*XOR*	*average*
.05	2.9	2.4	2.7
.1	1.9	2.1	2.0
.2	1.7	2.0	1.8
Average	2.2	2.2	2.2

Table 3. Ratios of learning times for BP and BPG, summarized from Tables 2 and 3.

$$\Delta g_i^s = -\epsilon_g \frac{\partial E}{\partial (g_i^s \mathrm{net}_i^s)} \langle \mathbf{w}_i^s, \mathbf{a}^{s-1} \rangle \tag{8}$$

Solving equation (7) for $\mathbf{a}^{s-1}$ and substituting into equation (8) yields:

$$\Delta g_i^s = \frac{\epsilon_g}{\epsilon_w} \frac{1}{g_i^s} \|\mathbf{w}_i^s\| \; \|\Delta \mathbf{w}_i^s\| \cos\theta \tag{9}$$

where θ is the angle between $\mathbf{w}_i^s$ and $\Delta \mathbf{w}_i^s$. Note also that

$$\Delta \|\mathbf{w}_i^s\|^2 = \|\Delta \mathbf{w}_i^s\|^2 + 2 \|\mathbf{w}_i^s\| \; \|\Delta \mathbf{w}_i^s\| \cos\theta. \tag{10}$$

Case I: $\cos\theta \geq 0$. From Equation 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 Equation 10, $\Delta \|\mathbf{w}_i^s\|^2 > 0$. In this case, then, the signs of $\Delta |g_i^s|$ and $\Delta \|\mathbf{w}_i^s\|$ agree: They both are positive.

Case II: $\cos\theta < 0$. From Equation 9, $\Delta |g_i^s| < 0$, by argument analogous to that used in Case I. From Equation 10, we can see that $\Delta \|\mathbf{w}_i^s\|^2 < \|\Delta \mathbf{w}_i^s\|^2$. For arbitrarily small[1] step sizes, $\|\Delta \mathbf{w}_i^s\|^2 \to 0$, so that $\Delta \|\mathbf{w}_i^s\|^2 < 0$. Again the signs of $\Delta |g_i^s|$ and $\Delta \|\mathbf{w}_i^s\|$ agree: They both are negative. Q.E.D.

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 Equations 3 and 4,

[1] For non-infinitesimal changes, suppose that $\|\Delta \mathbf{w}_i^s\| = p \, \|\mathbf{w}_i^s\|$, 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 $\Delta \mathbf{w}_i^s$ 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 ephemerally as to be negligible.

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:

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

where Δw_{ij}^s is the modified weight change. Notice that the normalized rule of Equation 11 is nearly the same as the standard weight update rule of Equation 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 Equation 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 $a_i = f(g_i\,\mathrm{net}_i) = f(0)$ 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 Equation 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 criterial 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

---

[2] There 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.

by building a network with a large number of hidden nodes and dynamically reducing them as learning progresses. Our previous observations regarding the gain parameter suggest that a judicious manipulation of gains could accomplish this goal.

Generalized cost function for gain competition.

We can use the gain parameter to make the hidden nodes compete for the right to participate in the representation and in learning. The idea is to start with a large hidden layer of candidate nodes, and to impose a cost on nodes that are similar. In general, the cost of similar nodes can be expressed as

$$C = \sum_{i,j} h(g_i)\, h(g_j)\, s_{ij} \tag{12}$$

where h is a nondecreasing function of gain, and s_{ij} is some measure of the similarity of nodes i and j. The sum is taken over all nodes in the hidden layer. Thus if two nodes are similar and both have large gains, they make a large contribution to the cost. We can perform gradient descent on the cost C with respect to gain in order to decrease the gains of redundant nodes. (One might also perform gradient descent on C with respect to connection weights.) We desire the derivative to be locally computable in a network architecture compatible with back propagation. Thus, the gain of a node is adjusted simultaneously by gradient descent on error and by gradient descent on the cost C (Eqn. 12).

The special case of gain decay. One special case of the cost function defines h to be the identity function, and defines s_{ij} to be 1 if $i = j$ and 0 otherwise. Then the cost function reduces to $C = \sum_i g_i^2$, and gradient descent on C with respect to g_i yields $\Delta g_i \propto -g_i$. That is simply gain decay, analogous to weight decay, which has often been used in applications of back propagation (e.g., [18,27]).

Another special case defines $h(g_i) = g_i/(1+g_i)$, with s_{ij} again equal to 1 if $i = j$ and 0 otherwise, so that $C = \sum_i g_i^2/(1+g_i)^2$. This is also a form of gain decay, in which the decay rate increases to a maximum for some small value of gain, and then decreases as gain gets larger. It is analogous to a cost function used by Rumelhart [19,20,28].

Those special cases of the cost function are particularly simple and easy to implement in a back propagation network. Unfortunately, gain decay alone does not take full advantage of the potential captured by the generalized cost function. We suggest now, and provide empirical evidence later, that gain decay excises slow-learning nodes rather than redundant nodes.

Specific cost function for gain competition. The measure of similarity used in the special cases of gain decay is really a degenerate case. We seek instead some measure of similarity which rises continuously from zero. Moreover, the measure must allow gradient descent on the cost of gains to be locally computable in a network compatible with back propagation. One such function is as follows:

$$s_{ij} = \frac{\left(\sum_p \mathrm{net}_{ip}\mathrm{net}_{jp}\right)^2}{\left(\sum_p \mathrm{net}_{ip}^2\right)\left(\sum_p \mathrm{net}_{jp}^2\right)} \tag{13}$$

where the sum is taken over all patterns, p, in a training sweep. The notation net_{ip} denotes the net input to node i when pattern p is input. The similarity in Equation 13 is a measure of the squared correlation of the net input values of nodes i and j across patterns. (It is not the usual definition of correlation because we have not subtracted the *mean* net input values.) It is always between zero and one, varies continuously, and it has a particularly simple derivative with respect to gain: zero. If we choose the identity function for $h(g)$ in the generalized cost function C, then gradient descent on C with respect to gain yields the competition rule:

$$\Delta g_i = -\gamma \sum_j g_j s_{ij} \tag{14}$$

where $\gamma \geq 0$ is a constant of proportionality, called the *competition rate*. A variant of Equation 14 was first introduced by Kruschke [16], who used a slightly different definition for s_{ij}.

Local computation. The value of Δg_i can be computed locally in a back propagation network if lateral connections between hidden layer nodes are allowed, that do not propagate activation or errors, but gain values. The strength on the lateral connection from node i to node j is s_{ij}, which can be computed with information available locally at the two nodes. Then $\sum_j g_j s_{ij}$ is easily computed by summing across the lateral connections. [3]

Simulation procedure.

The encoder problem [1] is an especially appropriate test for the gain competition scheme. In the encoder problem, a network with N input nodes and N output nodes is trained to map the N canonical basis vectors to themselves; i.e., $(1, 0, \ldots) \mapsto (1, 0, \ldots)$, $(0, 1, \ldots) \mapsto (0, 1, \ldots)$, etc. "Good" solutions to the encoder problem use $\log_2 N$ hidden nodes that together act as a binary encoder of the input. In that case, the hidden nodes are completely uncorrelated with each other. That is exactly the sort of situation that the gain competition scheme is supposed to encourage.

We used a network with 8 input nodes, 16 hidden nodes and 8 output nodes. To initialize the network, the weights on each node were chosen at random and then the weight vector of each node, including its bias weight, was normalized to Euclidean length one. The gain of each node was also initially set to one. Standard back propagation was performed; i.e., gradient descent on error with respect to weights, with a learning rate of 0.20 and a momentum of 0.85. Gradient descent on error *with respect to gains* was also performed on both the hidden and output layers, with the learning rate set to 0.20, and zero momentum. The learning was run until the maximum absolute error for every output node was less than 0.15.

[3] One can also easily derive $\partial C / \partial w_{uv}$, but computing the change in weight is not any simple variant of the feed-forward network for back propagation. One goal for future research is to construct a function s_{ij} which yields gradient descent on C with respect to weights that is locally computable.

Gain competition was simultaneously performed on the hidden layer. We wanted to impose as much competition as possible without extinguishing learning (error reduction), so we dynamically modulated the competition rate as learning progressed, in the following simple way. If learning was proceeding well, with the square root of the mean squared error (RMSE) falling rapidly, then competition was increased. If learning was slow, competition decreased. Specifically, the competition rate γ of Equation 14 was a constant multiple of an exponential average of the change in RMSE over previous trials, divided by the RMSE of the present trial:

$$\gamma(\tau) = -\kappa \left(\sum_{t=0}^{\tau} \omega^t \, \Delta\mathrm{RMSE}(\tau - t) \right) \Big/ \mathrm{RMSE}(\tau) \tag{15}$$

where $\gamma(\tau)$ is the competition rate on trial τ, where κ is a non-negative constant of proportionality called the *competition rate modulator*, where the sum is taken over all previous trials t, and where ω is a positive constant governing the weighting of the exponential average. In our simulations we set $\omega = 0.70$, and tested the competition rate modulator κ at several values between zero and two. Note that if the network is in fact learning, then ΔRMSE should be negative, and so the comptetition rate γ should be positive. However, gradient descent algorithms like back-propagation can sometimes "overshoot" a local descent and cause ΔRMSE to be greater than zero, consequently causing γ to be less than zero. To deal with that possibility, the value of the competition rate γ was actually set to the maximum of zero and the value computed in Equation 15. Gain competition was not actually executed until the fifth sweep, so that the value of γ would reflect the smooth behavior of RMSE rather than the initial turbulence often seen in back propagation. Finally, gain was restricted to be non-negative so that competition could never "explode," driving some gains to large negative values and others to large positive values.

It was found that gain decay contributed almost nothing to the excision of hidden nodes (these results are reported later), and added significantly to learning time. We therefore did not include the diagonal terms of Equation 12 in the computation of gain cost, C. Thus the change in gain due to competition was

$$\Delta g_i = -\gamma \sum_{j \neq i} g_j s_{ij} \tag{16}$$

(cf. Eqn. 14). Every factor in the right hand side of Equation 16 is a non-negative number. Hence competition could only decrease the gains, never increase them.

Results.

The results showed a large decrease in the number of participating hidden nodes as a function of the competition rate modulator, κ. Results are summarized in Table 4. Fifteen trials were run at each value of κ, each trial starting with a different set of random weights. (The same fifteen initializations were used for each of the values of κ.)

κ	0.0	0.5	1.0	2.0
mean	16.0	5.33	3.93	3.20
range	$16-16$	$4-6$	$3-5$	$3-4$
SD	0	0.70	0.57	0.40

Table 4. The number of participating hidden nodes (nodes with non-zero gain) after learning with competition, for different values of κ.

When there was no gain competition ($\kappa = 0.0$), all 16 of the hidden nodes built into the network ended up participating in the representation. In fact, all 16 nodes had gains greater than 1 on every trial. From these results it is clear that back propagation alone does nothing to excise redundant nodes, but instead utilizes all the nodes available to it.

When only a small amount of competition was used ($\kappa = 0.5$), the number of participating hidden nodes dropped to an average of only 5.33. All the other hidden nodes had gains of zero (to more than six decimal places). The number of participating hidden nodes did not vary much from trial to trial, as can be seen from Table 4.

As the gain competition was increased to $\kappa = 2.0$, usually (in 12 of 15 trials) only three hidden nodes remained. That is exactly the "good" solution, using $\log_2 8$ hidden nodes, that the competition scheme was meant to encourage.

The benefits of gain competition do not come without some cost. As might be expected, learning time increased as the competition rate modulator κ increased. Table 5 shows that for $\kappa = 2.0$, learning time is almost three times that for $\kappa = 0.0$. It is important to notice that learning time is a *negatively* accelerating function of κ. Thus gain competition does not cause an *explosive* growth in learning time.

For comparison, an 8-3-8 network was tested on the encoder problem with fifteen random weight initializations, using the same procedure but without competition. The network reached criterion in an average of 328 sweeps, with a range of 195 to 455 sweeps, and a standard deviation of 66.8 sweeps. Thus adding competition takes less than twice as long as back propagation with gain. Since back propagation with gain was shown to be about twice as fast as standard back propagation, the competition scheme ends up taking no longer than standard back propagation.

The results reported so far have described the *retirement* of nodes from participation. On the other hand, if the competition rate modulator κ is set to much larger values, some interesting behavior occurs. The high competition initially suppresses all but one or two nodes. The network will reduce the error as best it can, but as the rate of error reduction slows, the competition rate γ gets smaller (by Eqn. 15). This allows new nodes to be *recruited* from the pool of retired nodes. One or more previously suppressed nodes grows to have positive gain, as a result of gradient descent on error with respect to gain. The newly recruited node is dissimilar to the previously participating nodes (otherwise competition

κ	0.0	0.5	1.0	2.0
mean	218	403	497	600
range	105 – 303	203 – 722	274 – 769	353 – 824
SD	55.3	129.9	162.3	145.0

Table 5. Learning time in sweeps for various amounts of competition.

would have continued to suppress it), and error can be reduced further (otherwise it would not have been recruited).

As suggested earlier, gain decay alone should not specifically excise *redundant* nodes, but only those that learn slowly. Here we present empirical support for that claim. Gain decay alone ($\Delta g_i = -\gamma g_i$) was applied using the same procedure as for competition without decay. Table 6 shows the results for the same fifteen initializations as used in Tables 4 and 5, with κ equal to 0.5 and 1.0. In this context, κ is the "decay" rate modulator, rather than a "competition" rate modulator.[4] All the hidden nodes had gains greater than zero after learning; that is, none were excised at all. Gain decay had two significant effects: It increased the learning time; and, it reduced the mean hidden node gain. Clearly this form of gain decay is counter-productive, and so we have good evidence that competition is an improvement over decay alone.

Other minimization techniques.

Several researchers have introduced other hidden layer minimization techniques, e.g. [16, 19, 20, 22, 23, 28–34]. No direct comparisons have been made of the relative performance of the alternative methods, in terms of learning speed and generalization improvement for particular applications. Each method introduces its own bias upon the type of representation learned by the network, and hence upon the generalization made by the network. For example, the gain competition method described here biases the network toward hidden layer representations that have uncorrelated node activations. That bias might be appropriate for some applications, and not for others. Fortunately, the generalized cost function for gains (Eqn. 12) allows other definitions of similarity that might be more appropriate for specific applications.

In some applications it might be known in advance that specific kinds of bias are desired, and they might have nothing to do with minimization of hidden layers. For example, in visual pattern recognition, the system should generalize over spatial translations, and so the constraint of translational symmetry can be built in from the start [1]. Hidden layer minimization techniques, on the other hand, are usually intended for applications in which the appropriate domain-specific constraints are not known in advance.

[4] Not shown are results for $\kappa = 2.0$, which led to unstable oscillations even for very small learning step sizes.

κ	0.5	1.0
mean hidden nodes	16	16
mean hidden gain	0.343	0.234
mean learning time	346	430

Table 6. Effects of gain decay alone.

V. General Discussion.

Gain and temperature.

Both Boltzmann-type networks [18,35] and back-propagation networks use sigmoidal activation functions. The "temperature" parameter used in Boltzmann machines can be thought of as the reciprocal of gain [6,7]. In view of this apparent similarity it is important to point out the following differences.

Boltzmann machines are stochastic and activation values are binary. In these networks, temperature connotes stochasticity of the activation values. On the other hand, BPG is a deterministic network and therefore, gain (or its reciprocal, temperature) cannot connote randomness. But BPG activation functions may be interpreted as mean field approximations to binary stochastic nodes (e.g., [36–38]). The stochastic interpretation of temperature in Boltzmann machines corresponds to discriminating power in deterministic systems.

In Boltzmann and Mean Field networks, all nodes have the same temperature and are subject to a global "cooling" or "annealing" schedule (but cf. [39]). In Boltzmann machines, sufficiently slow cooling rates guarantee that the activation values for a given fixed set of weights will settle in a global energy minimum [40]. No corresponding claim is being made for BPG; there is no externally imposed annealing schedule for gains, and changing the gains by gradient descent does nothing to avoid local minima on the error surface.

Gain and natural networks.

There is evidence suggesting that the nervous system has mechanisms that modulate the neural response function in a manner similar to gain. Servan-Schreiber, Printz & Cohen [41] proposed that a gain parameter helps explain the effect of biogenic monoamines, a type of neurotransmiter associated with neural responsiveness modulation. They used the gain parameter in back-propagation networks to simulate phenomena associated with catecholamine manipulations. There is also evidence that these amines play an important role in the modulation of learning [42].

Summary and Conclusion.

We have introduced a gain parameter into the standard back propagation algorithm and shown it to be beneficial for speeding up learning and creating hidden layer bottlenecks for the goal of improved generalization. It was shown that the gain of a node is a handle on its participation in representing the input, and on the node's ability to learn. Simulations confirmed that adaptive gains significantly accelerated standard back propagation. Other simulations showed that gain competition very effectively created hidden layer bottlenecks in the case of the encoder problem.

Not all applications will benefit equally from the techniques we have introduced. BPG speeds up learning when there are nodes which are learning successfully. When very few hidden nodes are included, there might be no successful nodes, and hence BPG might not significantly improve learning. The competition scheme is designed to favor solutions with uncorrelated hidden nodes. Our simulations showed that it works very well for the encoder problem, but if a "good" solution to some other particular problem uses highly correlated nodes, then the competition scheme might be counter-productive. The general cost function for gains allows other definitions of similarity that might solve this difficulty.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, & R. J. Williams, "Learning internal representations by back-propagating errors." In: D. E. Rumelhart & J. L. McClelland (eds.), *Parallel Distributed Processing, v.1.* Cambridge, MA: MIT Press, 1986. Ch.8.

[2] D. E. Rumelhart, G. E. Hinton, & R. J. Williams, "Learning representations by back-propagating errors." *Nature*, **323**, 533-536. 1986.

[3] Y. LeCun. "A learning scheme for asymmetric threshold networks." In: *Proceedings of Cognitiva 85*, pp.599-604, Paris, France. 1985.

[4] D. B. Parker. *Learning-Logic*, Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University. 1982.

[5] P. J. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences.* PhD dissertation, Harvard University. 1974.

[6] J. R. Movellan. "Self-Regulated Temperature in back propagation networks." Paper presented at The Ninth Annual Berkeley-Stanford Conference, 1987, Berkeley CA.

[7] R. Tawel. "Does the neuron 'learn' like the synapse?" In: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems, I.* San Mateo, CA: Morgan Kaufmann, 1989., pp.169-176.

[8] J.J. Hopfield. "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences USA*, 81, 3888-3092, May 1984.

[9] S. Nowlan. "Gain variation in recurrent error propagation networks." Technical Report CRG-TR-88-1, Dept. of Comp. Sci., U. of Toronto, June 1988.

[10] D. Plaut, S. Nowlan, & G. Hinton. "Experiments on learning by back propagation." Carnegie-Mellon University Computer Science Department Technical Report, CMU-CS-86-126, 1986.

[11] J. L. McClelland, & D. E. Rumelhart. *Explorations in Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1989.

[12] S. E. Fahlman, "Faster-learning variations on back-propagation: An empirical study." In: D. Touretzky, G. Hinton, & T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, pp.38-51. San Mateo, CA: Morgan Kaufmann, 1989. See also: S. E. Fahlman, "An empirical study of learning speed in back-propagation networks." Tech. Rep. CMU-CS-88-162, Comp. Sci. Dept., Carnegie-Mellon U., June 1988.

[13] R. A. Jacobs. "Increased rates of convergence through learning rate adaptation." *Neural Networks*, **1**, 295-307, 1988.

[14] R. S. Sutton and A. G. Barto. "Toward a modern theory of adaptive networks: expectation and prediction." *Psychological Review*, **88**, 135-170, 1981.

[15] E. Oja. "A Simplified Neuron Model as A Principal Component Analyzer." *Journal of Mathematical Biology*, 15, 267- 273, 1982.

[16] J. K. Kruschke. "Creating local and distributed bottlenecks in hidden layers of back propagation networks." In: D. Touretzky, G. Hinton, & T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, pp.120-126. San Mateo, CA: Morgan Kaufmann, 1989.

[17] J. R. Movellan. *Computational Aspects of Contingency Detection: New Options from Connectionism.* PhD dissertation. Department of Psychology, University of California at Berkeley, 1989.

[18] G. Hinton. "Learning translation invariant recognition in a massively parallel network." In: G. Goos and J. Hartmanis (eds.), *Lecture Notes in Computer Science, Volume I: Parallel Architectures and Languages Europe.* New York: Springer-Verlag, 1987.

[19] D. E. Rumelhart. "Learning and generalization." Plenary Address, IEEE International Conference on Neural Networks, San Diego, 1988.

[20] A. S. Weigend, B. A. Huberman, & D. E. Rumelhart. "Predicting the future: A connectionist approach." Technical Report Stanford-PDP-90-01, Dept. of Psychology, Stanford University, 1990. Also in *International J. Neural Systems*, **1**(3).

[21] M. A. Franzini, "Learning to recognize spoken words: A study in connectionist speech recognition." In: D. Touretzky, G. Hinton, & T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, pp.407-416. San Mateo, CA: Morgan Kaufmann, 1989.

[22] J. K. Kruschke. "Improving generalization in back-propagation networks with distributed bottlenecks." In: *IJCNN International Joint Conference on Neural Networks*, New York: IEEE, 1989., v.I, pp.443-447.

[23] J. K. Kruschke. "Distributed bottlenecks for improving generalization in back-propagation networks." *International J. of Neural Networks Res. and Appl.*, **1**, 187-193, 1989.

[24] E. B. Baum, & D. Haussler. "What size net gives valid generalization?" *Neural Computation*, **1**, 151-160, 1989.

[25] J. Denker, D. Schwartz, B. Wittner, S. Solla, J. Hopfield, R. Howard, & L. Jackel. "Automatic learning, rule extraction, and generalization." *Complex Systems*, **1**, 877-922, 1987.

[26] M. A. Gluck, M. Pavel, & V. Henkle. "Constraints on adaptive networks for modeling human generalization." In: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems, I.* San Mateo, CA: Morgan Kaufmann, 1989., pp.2-10.

[27] G. Hinton. "Connectionist learning procedures." *Artificial Intelligence*, **40**, 185-234, 1989.

[28] S. J. Hanson & L. Y. Pratt "Comparing biases for minimal network construction with back-propagation." In: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems, I.* San Mateo, CA: Morgan Kaufmann, 1989., pp. 177-185.

[29] T. Ash "Dynamic node creation in back-propagation networks (abstract)." In: *IJCNN International Joint Conference on Neural Networks*, New York: IEEE, 1989., p.II-623. Also Technical Report 8901, Institute for Cognitive Science, U. C. San Diego.

[30] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units." In: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems, I.* San Mateo, CA: Morgan Kaufmann, 1989., pp. 519-526.

[31] M. C. Mozer & P. Smolensky "Skeletonization: A technique for trimming the fat from a network via relevance assessment." In: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems, I.* San Mateo, CA: Morgan Kaufmann, 1989., pp. 107-115.

[32] D. Psaltis & M. Neifeld "The emergence of generalization in networks with constrained representations." In: *Proceedings of the IEEE International Conference on Neural Networks*, San Diego CA, 24-27 July 1988. Piscataway NJ: IEEE Service Center, v.I, pp.371-381.

[33] P. A. Sandon & L. M. Uhr "A local interaction heuristic for adaptive networks." In: *Proceedings of the IEEE International Conference on Neural Networks*, San Diego CA, 24-27 July 1988. Piscataway NJ: IEEE Service Center, v.I, pp.317-324.

[34] J. Sietsma & R.J.F. Dow, "Neural Net Pruning - Why and How." In: *Proceedings of the IEEE International Conference on Neural Networks*, San Diego CA, 24-27 July 1988. Piscataway NJ: IEEE Service Center, v.1, pp.325-333.

[35] G.E. Hinton & T.J. Sejnowski, "Learning and Relearning in Boltzmann Machines," in D.Rumelhart and J. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Volume 1*, Cambridge MA, MIT Press, 1986.

[36] G. Hinton. "Deterministic Boltzmann learning performs steepest descent in weight-space." *Neural Computation*, **1**, 143-150, 1989.

[37] C. Peterson, & J. R. Anderson. "A mean field theory learning algorithm for neural networks." *Complex Systems*, **1**, 995-1019, 1987.

[38] C. Peterson, & E. Hartman. "Explorations of the mean field theory learning algorithm." *Neural Networks*, **2**, 475-494, 1989.

[39] J. Leinbach. "Automatic local annealing." In: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems, I.* San Mateo, CA: Morgan Kaufmann, 1989., pp.602-609.

[40] S. Geman and D. Geman. "Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, 721-741, 1984.

[41] C. Servan-Schreiber, H. Printz, & J. D. Cohen. "A network model of neuromodulatory effects: Gain, signal-to-noise ratio, and behavior." *Science*, **249**, 892-895, 1990.

[42] T. Kamatsu & J. D. Pettigrew. "Depletion of brain cathecolamines: Failure of ocular dominance shift after neurocular occlusion in kittens." *Science*, **194**, 206-208, 1976.