In Proceedings of the 9th Annual Conference of the Cognitive Science Society (pp. 177-186), 1987. Hillsdale, NJ: Erlbaum.

Using Fast Weights to Deblur Old Memories

Geoffrey E. Hinton and David C. Plaut

Computer Science Department
Carnegie-Mellon University

Abstract

Connectionist models usually have a single weight on each connection. Some interesting new properties emerge if each connection has two weights: A slowly changing, plastic weight which stores long-term knowledge and a fast-changing, elastic weight which stores temporary knowledge and spontaneously decays towards zero. If a network learns a set of associations and then these associations are "blurred" by subsequent learning, *all* the original associations can be "deblurred" by rehearsing on just a few of them. The rehearsal allows the fast weights to take on values that temporarily cancel out the changes in the slow weights caused by the subsequent learning.

1. Introduction

Most connectionist models have assumed that each connection has a single weight which is adjusted during the course of learning. Despite the emerging biological evidence that changes in synaptic efficacy at a single synapse occur at many different time-scales (Kupferman, 1979; Hartzell, 1981), there have been relatively few attempts to investigate the computational advantages of giving each connection several different weights that change at different speeds. Even for phenomena like short-term memory where fast-changing weights might seem appropriate, connectionist models have typically used the activation levels of units rather than the weights to store temporary memories (Little and Shaw, 1975; Touretzky and Hinton, 1985). We know very little about the range of potential uses of fast weights. How do they alter the way networks behave, and what extra computational properties do they provide?

In this paper we assume that each connection has both a fast, elastic weight and a slow, plastic weight. The slow weights are like the weights normally used in connectionist networks–they change slowly and they hold all the long-term knowledge of the network. The fast weights change more rapidly and they continually regress towards zero so that their magnitude is determined solely by their recent past. The effective weight on the connection is the sum of these two.

At any instant, we can think of the system's knowledge as consisting of the slow weights with a temporary overlay of fast weights. The overlay gives a temporary context–a temporary associative memory that allows networks to do more flexible information processing. Many ways of using this temporary memory have been suggested.

1. It can be used for rapid temporary learning. When presented with a new association the network can store it in one trial, provided the storage only needs to be temporary.
2. It can be used for creating temporary bindings between features. Recent work by Von der Malsburg (1981) and Feldman (1982) has shown that fast-changing weights can be used to dynamically bind together a number of different properties into a coherent whole, or to discover approximate homomorphisms between two structured domains.
3. It can be used to allow truly recursive processing. During execution of a procedure, the

values of local variables and the stage reached in the procedure (the program counter) can be stored in the fast weights. This allows the procedure to call a subprocedure whose execution involves different patterns of activity in the very same units as the calling procedure. When the subprocedure has finished using the units, the state of the calling procedure can be reconstructed from the temporary associative memory. In this way the state does not need to be stored in the *activity* of the units, so the very same units can be used for running the subprocedure. A working simulation of this kind is described briefly in McClelland & Kawamoto (1986).

4. It can be used to implement a learning method called "shortest descent" which is a way of minimizing the amount of interference caused by new learning. Shortest descent will be described in a separate paper and is not discussed further here.

In this paper we describe a novel use for a temporary memory stored in the fast weights: it can be used for cancelling out the interference in a set of old associations caused by more recent learning. Consider a network which has slowly and painfully learned a set of associations in its slow weights. If this network is then taught a new set of associations without any more rehearsal of the old associations, there is normally some degradation of the old associations. We show that by using fast weights it is possible to quickly restore a whole set of old associations by rehearsing on just a subset of them. The fast weights cancel out the changes in the slow weights that have occurred since the old associations were learned, so the combination of the current slow weights and the fast weights approximates the earlier slow weights. The fast weights therefore create a context in which the old associations are present again. When the fast weights decay away, the new associations return.

2. A deblurring analogy

There is an analogy between the use of fast weights to recover unretrained associations and a technique called "deblurring" which is sometimes used for cleaning up blurred images. Suppose that you are in your office and you want to take some photographs of what is on your computer screen. You set up a tripod in front of the screen and then you carefully focus the camera and take one photograph. Unfortunately, before you can take any more, your office mate moves the camera to a tripod in front of his screen and refocuses it. You now move the camera back to your tripod and it seems as if you must refocus, but there is an interesting alternative. You take another photograph of the same screen without refocussing and you compare it with your first photograph. The first photo is the "desired output" of the process that maps from screens to photos, and the difference between it and the "actual output" achieved with the out-of-focus camera can be used to estimate a deblurring operator which can be applied to the actual output to convert it to the desired output. This same deblurring operator can then be applied to any image taken with the out-of-focus camera. Focussing the camera is analogous to learning the slow weights that are required to map from input vectors (screens) to output vectors (photos). Estimating the deblurring operator is analogous to learning the fast weights that are required to compensate for the noise that has been added to the slow weights since the original learning (see figure 1).

The advantage of using fast weights rather than slow ones for deblurring is that it does not permanently interfere with the new associations. As soon as the fast weights have decayed back to zero, the new knowledge is restored.

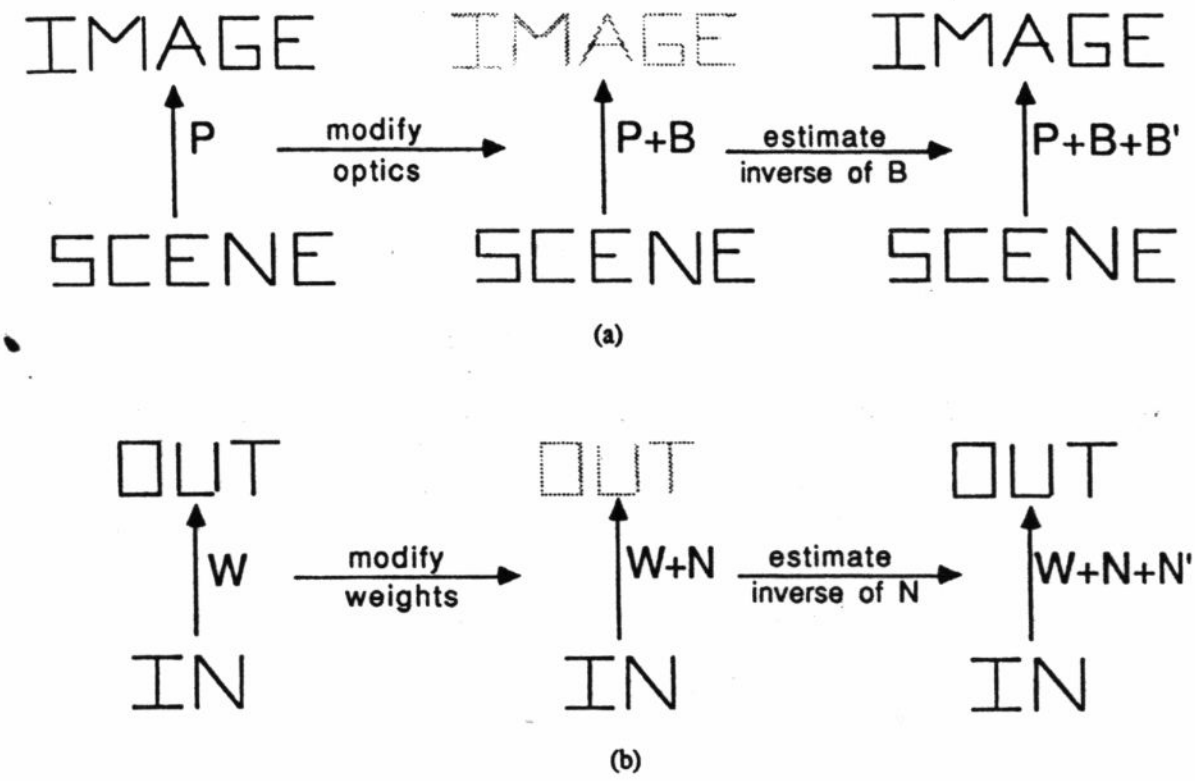

Figure 1: An illustration of the deblurring analogy between (a) images and (b) networks. The laws of projection (P) determine the mapping from the scene to the image. After applying a blurring function (B), applying the inverse of B to the blurred image restores the image. Similarly, the weights (W) define a mapping from input to output. After noise (N) is added, determining and applying the inverse of N allows the network to produce the original output.

3. The learning procedure

We used the back propagation learning procedure (Rumelhart *et al.*, 1986a; 1986b) to investigate the properties of networks that learn with both fast and slow weights. We summarize the procedure below–the full mathematical details can be found in the above references.

The procedure operates on layered, feed-forward networks of deterministic, neuron-like units. These networks have a layer of input units at the bottom, any number of intermediate layers of hidden units, and a layer of output units at the top. Connections are allowed only from lower to higher layers.

The aim of the learning procedure is to find a set of weights on the connections such that, when the network is presented with each of a set of input vectors, the output vector produced by the network is sufficiently close to the corresponding desired output vector. The error produced by the network is defined to be the squared difference between the actual and desired output vectors summed over all input-output cases. The learning procedure minimizes this error by performing gradient descent in weight

space. This requires adjusting each weight in the network in proportion to the partial derivative of the error with respect to that weight. These error derivatives are calculated in two passes.

The forward pass determines the output (or *state*) of each unit in the network. An input vector is presented to the network by setting the states of the input units. Layers are then processed sequentially, working from the bottom upward, with the states of units within a layer set in parallel. The input to a unit is the scalar product of the states of units in lower layers from which it receives connections, and the weights on these connections. The output of a unit is a real-valued smooth non-linear function of its input. The forward pass ends when the states of the output units are set.

The backward pass starts with the output units at the top of the network and works downward through each successive layer, "back-propagating" error derivatives to each weight in the network. For each layer, computing the error derivatives of the weights on incoming connections involves first computing the error derivatives with respect to the outputs, and then with respect to the inputs, of the units in that layer. The simple form of the unit input-output functions makes these computations straightforward. The backward pass is complete when the derivative of the error with respect to each weight in the network has been determined.

The simplest version of gradient descent is to decrement each weight in proportion, ε, to its error derivative. A more complicated version that usually converges faster is to add to the current weight change a proportion, α, of the previous weight change. This is analogous to introducing *momentum* to movement in weight space.

Since the effective weight on a connection is the sum of the fast and slow weights, these weights experience the same error derivative. Hence they behave differently only because they are modified using different weight change parameters ε and α, and because the fast weights decay towards zero. This is achieved by reducing the magnitude of each fast weight by some fraction, h, after each weight change.

4. A simulation of the deblurring effect

To demonstrate the deblurring effect, we used a simple task and a simple network. The task was to associate random binary 10-bit input vectors with random binary 10-bit output vectors. We selected 100 input vectors at random (without replacement) from the set of all 2^{10} binary vectors of length 10, and for each input vector we chose a random output vector. The network we used had three layers: 10 input units, 100 hidden units, and 10 output units. Each input unit was connected to all the hidden units, and each hidden unit was connected to all the output units. The hidden and output units also had variable biases that were modified during the learning.

We trained the network by repeatedly sweeping through the whole set of 100 associations and changing the weights after each sweep. After prolonged training (1300 sweeps with $\varepsilon = .02$ and $\alpha = .9$) the network knew the associations perfectly, and all the knowledge was in the slow weights. The fast weights were very close to zero because the errors were very small towards the end of the training, so the tiny error derivatives were dominated by the tendency of the fast weights to decay towards zero by 1% after each weight update.

Once the 100 associations were learned, we trained the network on 5 new random associations without further rehearsal on the original 100. Again, we continued the training until the new knowledge was in the slow weights (400 sweeps). We then retrained the network on only a subset of the original

associations, and compared the improvement in performance on this retrained subset with the (incidental) improvement in performance on the rest of the associations. The main result, shown in figure 2, was that in the early stages of retraining the improvements in the associations that were not retrained were very nearly as good as in the associations that were explicitly retrained. This rather surprising result held even if only 10% of the old associations were retrained.

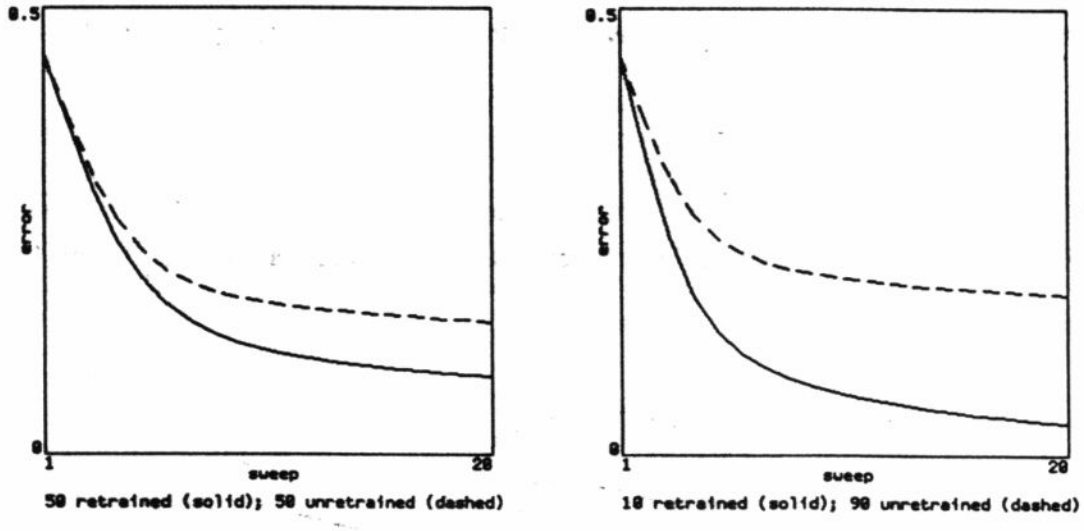

Figure 2: Performance on retrained and unretrained subsets of 100 input-output cases after (1) learning all 100 cases; and (2) interfering with the weights by learning 5 unrelated cases. The average error of an output unit is shown for the first 20 sweeps through the retrained subset.

The reason for this effect is that the knowledge about each association is distributed over many different connections, and when the network is retrained on *some* of the associations *all* the weights are pushed back towards the values that they used to have after the associations were first learned. So there is improvement in the unretrained associations even though they are only randomly related to the retrained ones. Of course, if the retrained and unretrained associations share some regularities the transfer will be even better.

4.1. A geometric explanation of the transfer effect

Consider the very simple network shown in figure 3. There are two input units which are directly connected to a single, linear output unit. The network is trained on two different associations each of which maps a two component input vector into a one component output vector. For each of the two input vectors, there will be many different combinations of the weights w_1 and w_2 that give the desired output vector, and since the output unit is linear, these combinations will form straight lines in weight space as shown is figure 4. In general, the only combination of weights that will satisfy both associations lies at the intersection of the two lines. So in this simple network, a gradient descent learning procedure will converge on the intersection point.

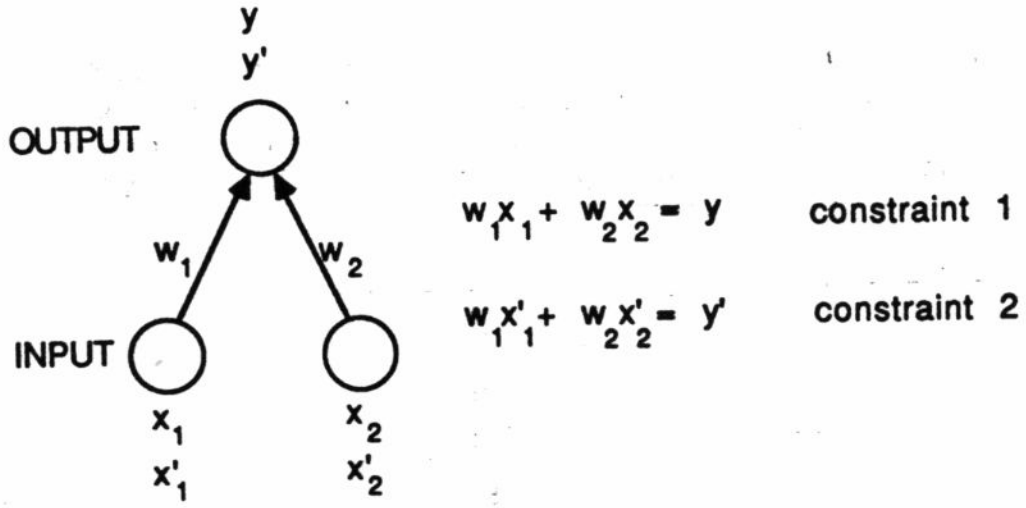

Figure 3: A simple network that learns to associate (x_1, x_2) with y and (x_1', x_2') with y'.

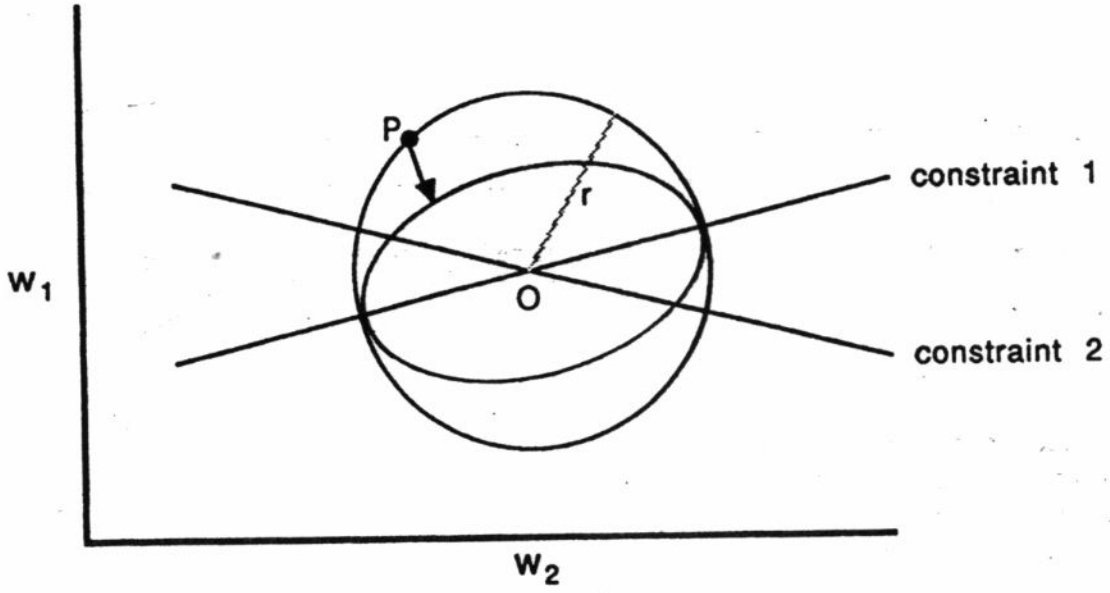

Figure 4: A plot of weight space for the simple network in figure 3.

Now, suppose we add to a network that has learned the associations a noise vector of length r that is selected at random from a distribution that is uniformly distributed over all possible orientations. The new combination of weights will be uniformly distributed over the circle shown in figure 4. If we now retrain an infinitessimal amount on association 1, we will move perpendicularly towards line 1. If we start from a point such as P that lies on one of the larger arcs of the circle, the movement towards line 1 will have a component *towards* line 2, whereas if we start from a point on one of the smaller arcs, the movement will have a component *away from* line 2. Thus, when we retrain a small amount on association 1, we are more likely than not to improve the performance on association 2, even though the associations are only randomly related to each other.

The expected positive transfer between retraining on one association and performance on the other can only be a result of starting the retraining from a point in weight space that has some special

relationship to the solution point, O. We initially thought that the starting point for the retraining needed to be *near* the solution point, but the geometrical argument we have just given is independent of the magnitude, r, of the noise vector. The crucial property of the starting point is that it is selected at random from a distribution of points that are all the same distance from O, and selecting starting points in this way induces a bias towards starting points that cause positive transfer between the two associations that define the point O.

The positive transfer effect obviously disappears if the two associations are orthogonal, and so we might expect the effect to be rather small when the network is large, because randomly related high-dimensional vectors tend to be almost orthogonal. In fact, it is important to distinguish between two qualitatively different cases. If the activity levels of the input units have a mean of zero, most pairs of high-dimensional vectors will be approximately orthogonal and so the effect is small and the expected transfer from one retrained association to one unretrained one gets smaller as the dimensionality increases. If, however, the activity levels of the input units range between 0 and 1 (as in the simulation described above) random high-dimensional vectors are not close to orthogonal and we show in the next section that the expected transfer from one retrained association to one unretrained one is independent of the dimensionality of the vectors.

5. A mathematical analysis

It is hard to analyze the expected size of the transfer effect for networks with multiple layers of non-linear units or for tasks with complex regularities among the input-output associations. However, an analysis of the transfer effect in simple networks learning random associations may provide a useful guide to what can be expected in more complex cases. We therefore derive the magnitude of the expected transfer from one retrained association, A, to one unretrained association, B, in a network that has no hidden units and only one linear output unit.[1] We assume that the network has previously learned to produce the correct activity level in the output unit for two input vectors, **a** and **b**, and that it is now retraining on association A after independent gaussian noise has been added to each weight. The noise added to the i^{th} weight is g_i and is chosen from a distribution with mean 0 and standard deviation σ. We also assume that the retraining involves changing the weights by a fixed, infinitesimal amount in the direction of steepest descent in the error function $\frac{1}{2}(y_A - d_A)^2$ where y_A is the actual output of the output unit and d_A is the desired output that was achieved before the noise was added.

A good measure of the magnitude of the transfer effect is the the ratio of two improvements: the improvement in association B caused by retraining on A and the improvement in association B that would have been caused by retraining directly on B. If the retraining involves changing the weight vector by a fixed amount in the direction of steepest descent, this ratio is simply the cosine of the angle between the direction of steepest descent for association A and the direction of steepest descent for association B.[2]

If the original learning was perfect, all of the error in the output would be caused by the noise added

---

[1]For layered, feed-forward networks with no hidden units, each output unit has its own separate set of weights. So a network with many output units can be viewed as composed of many separate networks each of which has a single output unit.

[2]If the retraining involves multiplying the gradient by a coefficient, ε, to determine the weight change, the actual ratio may differ because the magnitudes of the gradients may differ for A and B. But this will not affect the *expected* ratio, so the analysis we give for the *expected* transfer is still valid.

to the weights, so

$$\frac{\partial E_A}{\partial y_A} = y_A - d_A = \sum_i a_i g_i$$

where a_i is the activity level of the i^{th} input unit in association A. So, for a weight, w_i, the derivatives of the error E_A for association A and E_B for association B are given by

$$\frac{\partial E_A}{\partial w_i} = \frac{\partial y_A}{\partial w_i} \cdot \frac{\partial E_A}{\partial y_A} \cdot a_i \cdot \sum_i a_i g_i, \qquad \frac{\partial E_B}{\partial w_i} = \frac{\partial y_B}{\partial w_i} \cdot \frac{\partial E_B}{\partial y_B} \cdot b_i \cdot \sum_i b_i g_i$$

Hence, the cosine of the angle, θ, between the directions of steepest descent for the two associations is given by

$$\cos(\theta) = \frac{\sum_i a_i b_i \cdot \sum_i a_i g_i \cdot \sum_i b_i g_i}{\left[\sum_i a_i^2 \left(\sum_i a_i g_i\right)^2 \cdot \sum_i b_i^2 \left(\sum_i b_i g_i\right)^2\right]^{1/2}}$$

$$= \frac{\sum_i a_i b_i}{\left(\sum_i a_i^2 \cdot \sum_i b_i^2\right)^{1/2}} \cdot \frac{\sum_i a_i g_i \cdot \sum_i b_i g_i}{\left|\sum_i a_i g_i \cdot \sum_i b_i g_i\right|} \qquad (1)$$

5.1. The zero-one case

The first part of equation 1 is simply the cosine of the angle between the two input vectors, and so it is independent of the weights. If the components of **a** and **b** are all 0 or 1 it can be written as

$$\frac{n_{ab}}{(n_a n_b)^{1/2}}$$

where n_a is the number of components that have value 1 in the vector **a**, and n_{ab} is the number that have value 1 in both **a** and **b**.

The second part of equation 1 depends on the weights. It always has a value of 1 or −1, depending on whether E_A and E_B have the same or opposite signs. Its numerator can be written as

$$\left(\sum_{i \in S_{ab}} g_i + \sum_{i \in S_{a\bar{b}}} g_i\right)\left(\sum_{i \in S_{ab}} g_i + \sum_{i \in S_{\bar{a}b}} g_i\right)$$

where S_{ab} is the set of input units that have value 1 in both **a** and **b**, $S_{a\bar{b}}$ is the set that have value 1 in **a** and 0 in **b**, and $S_{\bar{a}b}$ is the set that have value 0 in **a** and 1 in **b**. Since these sets are disjoint and the expected value of the products of independent zero-mean gaussians is 0, all the cross-products in the numerator of equation 1 vanish when we take expected values except for the term

$$\left\langle \sum_{i \in S_{ab}} g_i \cdot \sum_{i \in S_{ab}} g_i \right\rangle = \left\langle \sum_{i \in S_{ab}} g_i^2 \right\rangle = n_{ab}\sigma^2$$

So the expected value of equation 1 can be written as

$$\frac{n_{ab}\sigma^2}{\left\langle \left| \sum_i a_i g_i \sum_i b_i g_i \right| \right\rangle} = \frac{n_{ab}\sigma^2}{n_a^{1/2}\sigma\, n_b^{1/2}\sigma} = \frac{n_{ab}}{n_a^{1/2} n_b^{1/2}}$$

and so the expected value of $cos(\theta)$ is given by

$$\langle cos(\theta) \rangle = \frac{n_{ab}^2}{n_a n_b} \tag{2}$$

If each component of a and b has value 1 with probability 0.5 and value 0 otherwise, the expected value of $cos(\theta)$ is 0.25. So if we add random noise to the weights of a simple network that has learned 100 associations and then we retrain on 50 of them using very small weight changes, the ratio of the expected improvements on an unretrained and a retrained association at the start of the retraining will be

$$\frac{50 \times .25}{1 + 49 \times .25} = .943$$

This agrees well with simulations we have done using networks with no hidden units.

5.2. The 1, 0, -1 case

When the components of the input vectors can also take on values of −1, a modification of the derivation used above leads to

$$\langle cos(\theta) \rangle = \frac{(n_{agree} - n_{disagree})^2}{n_a n_b} \tag{3}$$

where n_{agree} is the number of input units for which $a_i b_i = 1$ and $n_{disagree}$ is the number of input units for which $a_i b_i = -1$.

For a pair of random vectors in which each component has a probability, p, of being a 1 and the same probability of being a −1, the expected value of the numerator of equation 3 is just the square of the expected length of a random walk in which each step has a probability of $2p^2$ of being to the left, $2p^2$ of being to the right, and $1-4p^2$ of being zero. The expected value of the numerator is therefore $4p^2n$, where n is the dimensionality of the input vector. Since the denominator has an expected value of order n^2, the expected transfer effect is inversely proportional to n.

The decrease in the expected transfer between a single pair of vectors is balanced by the fact that larger networks can hold more associations. If the number of associations learned is proportional to the dimensionality of the input vectors, and if a constant *fraction* of the associations are retrained after adding noise to the weights, the ratio of the initial improvement on the unretrained associations to the improvement on the retrained associations is independent of the dimensionality.

5.3. How the transfer effect decreases during retraining

The analyses we have presented are for the transfer between random associations during the earliest stage of retraining. As retraining proceeds, the transfer effect diminishes. Figure 4 presents a simple geometrical explanation of why this occurs. At the start of retraining, the point in weight space lies

somewhere on the circle, and the probability that the transfer will be positive is simply the fraction of the circumference that lies in the two larger arcs of the circle. Retraining on association 1 moves each point on the circle perpendicularly towards the line 1 by an amount proportional to its distance from 1, so after a given amount of retraining, each point on the circle will move to the corresponding point on the ellipse. The probability of positive transfer is now equal to the fraction of the circumference of the ellipse that lies in the two larger arcs.

6. Conclusions

This paper demonstrates and analyses a powerful and surprising transfer effect that was first described (but not analysed) by Hinton and Sejnowski (1986). The analysis applies just as well if there are no fast weights and the relearning takes place in the slow weights. The advantage of fast weights is that they allow this effect to be used for temporarily deblurring old memories without causing significant interference to new memories. When the fast weights decay away the newer memories are restored.

There are many anecdotal descriptions of phenomena that could be explained by the kind of mechanism we have proposed, but we know of no well controlled studies. We predict that if a two unrelated sets of associations are learned at the same time, and if the internal representations used for different associations share the same units, then retraining on one set of associations will substantially improve performance on the other set. One problem with this prediction is that with sufficient learning, connectionist networks tend to use different sets of units for representing unrelated items. A second problem is that even if the unretrained associations are enhanced, the effect may be masked by response competition due to facilitation of the responses to the retrained items. Nevertheless, the type of effect we have described can be very large in simulated networks and so a well designed experiment should be able to detect whether or not it occurs in people.

References

Feldman J.A. Dynamic connections in neural networks. *Biological Cybernetics*, 1982, *46*, 27-39.

Hartzell H.C. Mechanisms of slow synaptic potentials. *Nature*, 1981, *291*, 539-543.

Hinton G.E. and Sejnowski T.J. Learning and relearning in Boltzmann Machines. In D.E. Rumelhart, J.L. McClelland, and the PDP research group (Eds.) *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, Cambridge, MA: MIT Press, 1986.

Kupferman I. Modulatory actions of neurotransmitters. *Annual Review of Neuroscience*, 1979, *2*, 447-465.

Little W.A. and Shaw G.L. Statistical-theory of short and long-term memory. *Behavioral Biology*, 1975, *14(2)*, 115-133.

McClelland J.L. and Kawamoto A.H. Mechanisms of sentence processing: Assigning roles to constituents of sentences. In J.L. McClelland, D.E. Rumelhart, and the PDP research group (Eds.) *Parallel distributed processing: Explorations in the microstructure of cognition. Volume II: Psychological and biological models*, Cambridge, MA: MIT Press, 1986.

Rumelhart D.E., Hinton G.E. and Williams R.J. Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, and the PDP research group (Eds.) *Parallel distributed processing: Explorations in the microstructure of cognition. Volume I: Foundations*, Cambridge, MA: MIT Press, 1986a.

Rumelhart D.E., Hinton G.E. and Williams R.J. Learning representations by back-propagating errors. *Nature*, 1986b, *323(9)*, 533-536.

Touretzky D.S. and Hinton G.E. Symbols among the neurons: Details of a connectionist inference architecture. *Proceedings, 9th International Joint Conference on Artificial Intelligence*, Los Angeles, August, 1985.

Von der Malsburg C. *The correlation theory of brain function*. Internal Report 81-2, Department of Neurobiology, Max-Plank-Institute for Biophysical Chemistry, P.O. Box 2841, Gottingen, F.R.G., 1981.