Backpropagation

To keep things simple, let us just work with one pattern. In that case the objective function is defined to be

$$E = \frac{1}{2}\|\boldsymbol{x}^K - \boldsymbol{d}\|^2 \tag{1}$$

where K is an index denoting the last layer in the network and d is the desired output.

The equation for updating the states is given by

$$\boldsymbol{x}^{k+1} = g(W^k \boldsymbol{x}^k) \tag{2}$$

where W^k is a matrix used to store the weights between layer k and layer $k+1$,

$$W^k = \begin{bmatrix} w_{11}^k & w_{12}^k & \cdots \\ w_{21}^k & \ddots & \\ \vdots & & w_{nn}^k \end{bmatrix} = \begin{pmatrix} \boldsymbol{w}_1^k \\ \boldsymbol{w}_2^k \\ \vdots \\ \boldsymbol{w}_n^k \end{pmatrix}$$

and the special understanding we shall have is that the function g applied to a vector is just that function applied to its elements; that is,

$$g(W\boldsymbol{x}) = \begin{pmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}) \\ g(\boldsymbol{w}_2 \cdot \boldsymbol{x}) \\ \vdots \end{pmatrix}$$

This is just a version of the optimal control problem where Equation 1 is the optimand and Equation 2 is the dynamics. Here instead of the control $\boldsymbol{u}$, the matrices $W^k, k = 1, \ldots, K$ represent the "control" variables. One difference, of course, is the dimensionality: In networks the dimension of the state space may range into the thousands! But the important thing to realize is that the mathematical treatment is the same.

Thus this problem can be tackled using the Euler-Lagrange formulation. The Hamiltonian is defined to be

$$H = \frac{1}{2}\|\boldsymbol{x}^K - \boldsymbol{d}\|^2 + \sum_{k=0}^{k=K-1} (\boldsymbol{\lambda}^{k+1})^T[-\boldsymbol{x}^{k+1} + g(W^k \boldsymbol{x}^k)]$$

where the first term measures the cost of errors reproducing the pattern and the second term constrains the system to follow its dynamic equations. Differentiating with respect to $\boldsymbol{x}^k$ provides the adjoint system of equations,

$$H_{\boldsymbol{x}^k} = \mathbf{0} = -\boldsymbol{\lambda}^k + [W^{kT}\Lambda^{k+1} g'(W^k \boldsymbol{x}^k)] \text{ for } k = 0, \ldots, K-1 \qquad (3)$$

where

$$\Lambda^{k+1} = \begin{bmatrix} \lambda_1^{k+1} & 0 & \cdots \\ 0 & \lambda_2^{k+1} & \\ \vdots & & \lambda_n^{k+1} \end{bmatrix}$$

and with the final condition given by

$$H_{\boldsymbol{x}^K} = \mathbf{0} = \boldsymbol{x}^K - \boldsymbol{d} - \boldsymbol{\lambda}^K \text{ for } k = K$$

Unpacking the Notation

$$H_{\boldsymbol{x}^k} = \mathbf{0} = -\boldsymbol{\lambda}^k + [W^{kT}\Lambda^{k+1}g'(W^k\boldsymbol{x}^k)] \text{ for } k = 0, \ldots, K-1 \qquad (4)$$

To understand Equation 4, consider the case where the number of units in each layer is just two. Then

$$(\boldsymbol{\lambda}^{k+1})^T g(W^k\boldsymbol{x}^k) = \lambda_1^{k+1} g(w_{11}x_1^k + w_{12}x_2^k) + \lambda_2^{k+1} g(w_{21}x_1^k + w_{22}x_2^k)$$

Taking the partial derivative with respect to x_1^k,

$$\lambda_1^{k+1} g'(w_{11}x_1^k + w_{12}x_2^k)w_{11} + \lambda_2^{k+1} g'(w_{21}x_1^k + w_{22}x_2^k)w_{21}$$

Similarly the partial derivative with respect to x_2^k,

$$\lambda_1^{k+1} g'(w_{11}x_1^k + w_{12}x_2^k)w_{12} + \lambda_2^{k+1} g'(w_{21}x_1^k + w_{22}x_2^k)w_{22}$$

Reassembling these terms into a vector results in

$$= W^T \begin{pmatrix} \lambda_1^{k+1} g'(\boldsymbol{w}_1^k \cdot \boldsymbol{x}^k) \\ \lambda_2^{k+1} g'(\boldsymbol{w}_2^k \cdot \boldsymbol{x}^k) \\ \vdots \end{pmatrix} = W^{kT}\Lambda^{k+1} g'(W^k\boldsymbol{x}^k)$$

Generating the Solution

To generate the solution, assume an initial W matrix. Then solve the dynamic equations going forward in levels in the network. This solution provides a value for $\boldsymbol{x}^K$. This value allows the adjoint system to be solved backward for $k = K$ to $k = 0$:

$$\boldsymbol{\lambda}^K = \boldsymbol{x}^K - \boldsymbol{d} \text{ for } k = K$$

$$\boldsymbol{\lambda}^k = -W^T \Lambda^{k+1} g'(W\boldsymbol{x}^k) \text{ for } k = 0, \ldots, K-1$$

where g' denotes differentiation with respect to its argument. Now improve the estimate for W using gradient descent. To do so, calculate the adjustment for the weights as $\frac{\partial H}{\partial w_{ij}}$:

$$\frac{\partial H}{\partial w_{ij}^k} = \lambda_i^{k+1} x_j^k g'(\boldsymbol{w}_i^k \cdot \boldsymbol{x}^k) \tag{5}$$

Equation 5 is very compact and so is worth unpacking in order to understand it. The multiplier λ_i^{k+1} can be understood by realizing that at level K it just keeps track of the output error. This is its role for the internal units also; it appropriately keeps track of how the internal weight change affects output error. The derivative g' is also simple, especially when using

$$g(u) = \frac{1}{1 + e^{-u}}$$

since it can be easily verified that

$$g'(u) = g(1 - g)$$

Thus to calculate g' for a given unit, just determine how much input the x_j^kth unit receives, and apply that as an argument to $g'()$.

Putting all this together results in the following algorithm.

Backpropagation Algorithm

Until the error is sufficiently small, do the following for each pattern:

1. Apply the pattern to the network and calculate the state variables using Equation 2.
2. Solve the adjoint Equation 3.
3. Adjust the weights W^k using

$$\Delta w_{ij}^k = -\eta \frac{\partial H}{\partial w_{ij}^k}$$

where $\frac{\partial H}{\partial w_{ij}^k}$ is given by Equation 5.

Notice the change in the formulation in the backpropagation equations. The original problem formulation used time as a dependent variable. Translating to the network formulation, the dependent variable became the different hidden state variables in the network. Time is translated into space.

In developing this algorithm only one pattern was used, but the extension to multiple patterns is straightforward. The strictly correct thing to do would be to accumulate the weight changes for each pattern separately, add them up, and then adjust the weight vector accordingly. The problem with this approach is that the calculations for each pattern are laborious. To make faster progress one can approximate the strict method by adjusting the weights after each pattern is used. This technique is known as *stochastic gradient descent*. The understanding is that the sum of the successive corrections should approximate the sum of all the corrections that are computed simultaneously.

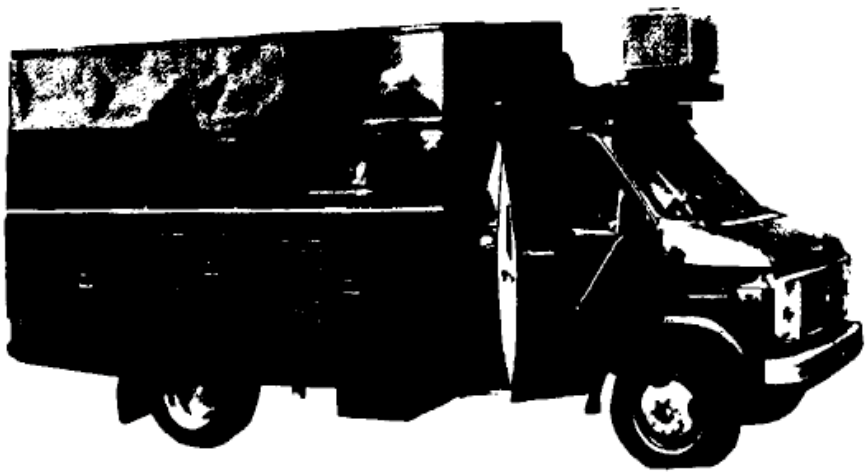

Figure 1:

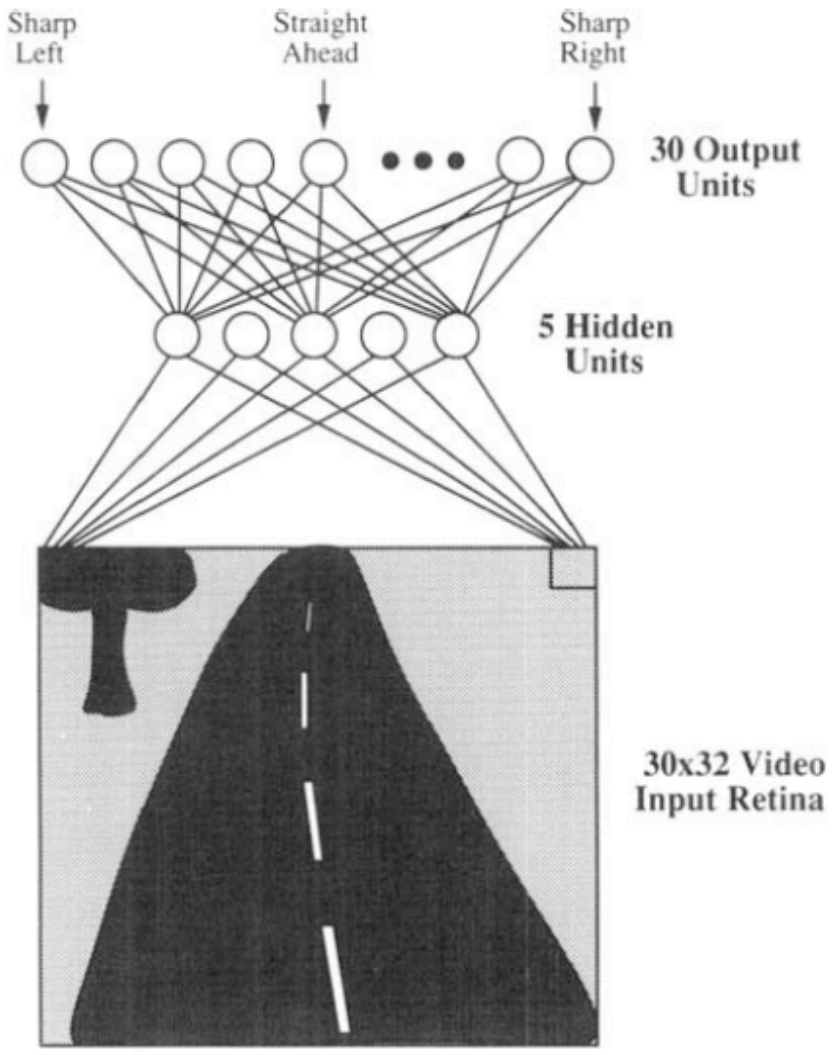
Sharp
Left
Straight
Ahead
Sharp
Right
30 Output
Units
5 Hidden
Units
30x32 Video
Input Retina

Figure 2:

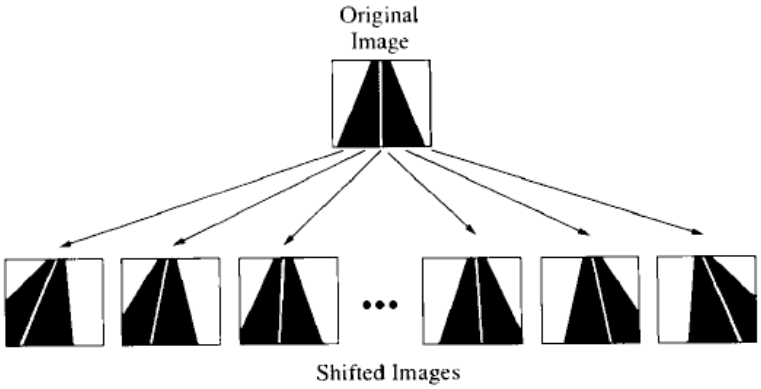

Figure 3:

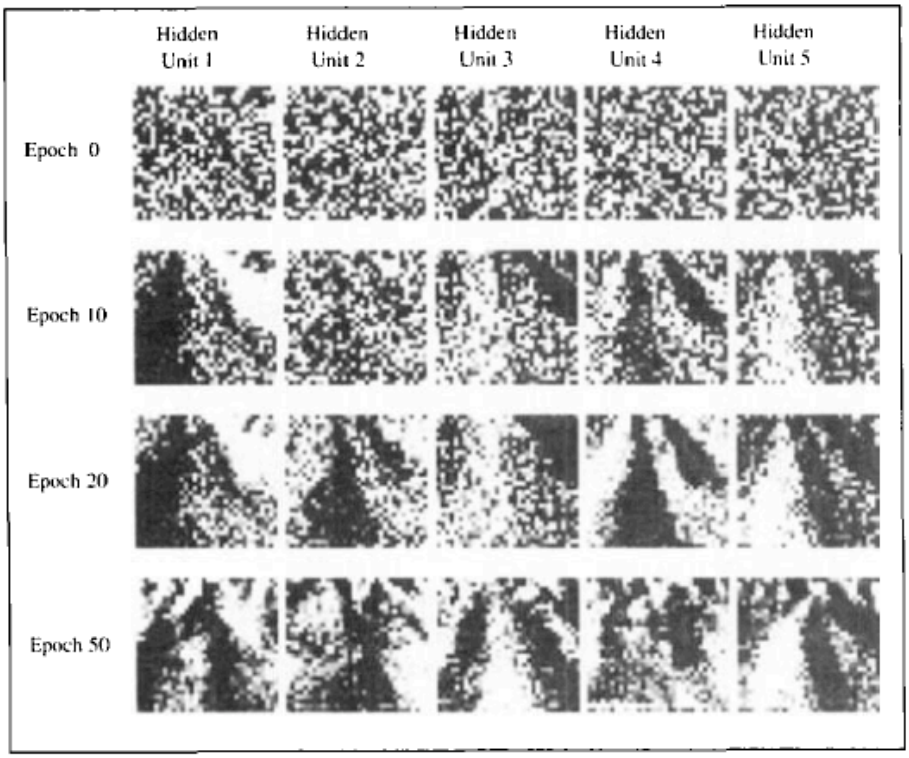

Figure 4:

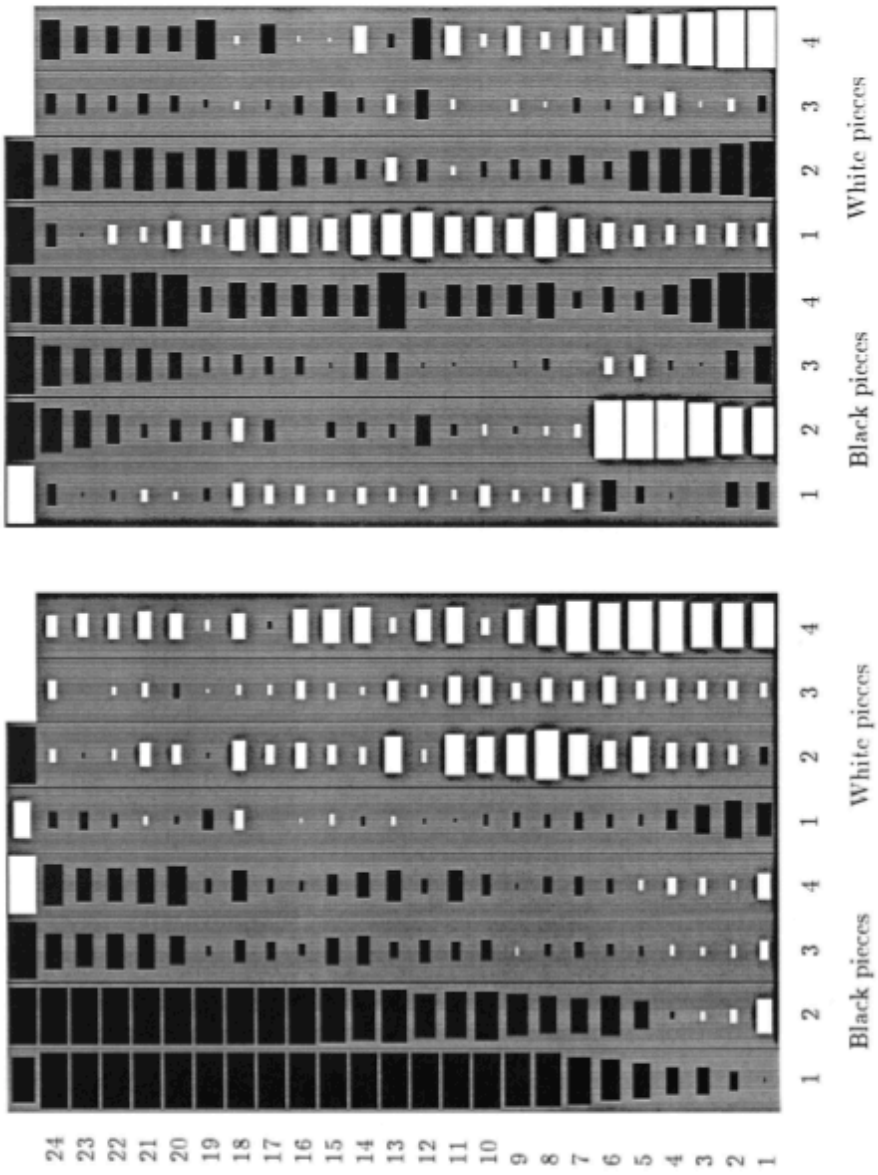
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
1 2 3 4
Black pieces
1 2 3 4
White pieces
1 2 3 4
Black pieces
1 2 3 4
White pieces

Figure 5: