

An Introduction to Backpropagation Networks

A simple backpropagation network consists of three or more layers of cells: an input layer, an output layer, and one or more hidden layers. Every cell in a layer (except the input layer) has a weighted connection from all of the cells in the layer below it. A system with N cells in the input layer, M cells in the output layer, and a single hidden layer with L cells is shown in Figure 1.

The basic idea is to train the network by randomly presenting a fixed set of input vectors $\vec{X}_p = (x_{p1}, x_{p2}, \dots, x_{pN})$ together with their desired outputs $\vec{Y}_p = (y_{p1}, y_{p2}, \dots, y_{pM})$. After each input, the weights between cells are incremented in a way that minimizes the mean-square error

$$E = \sum_{k=1}^M (y_{pk} - o_{pk})^2,$$

where o_{pk} is the computed output at the k th output cell when the input \vec{X}_p is presented to the input layer. This is an example of “supervised learning”.

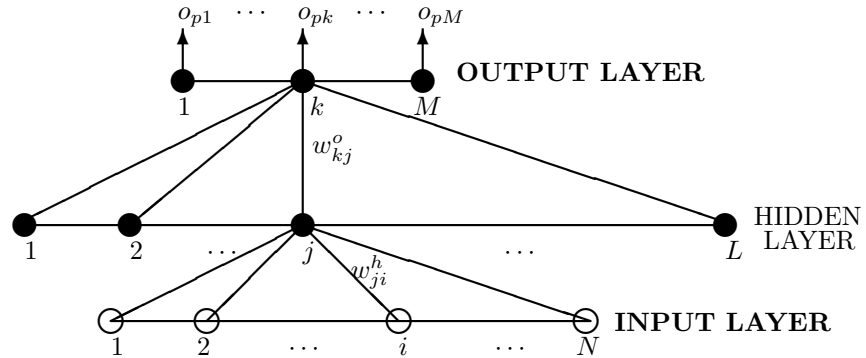


Figure 1: 3-layer network

Before deriving a formula for incrementing the weights, it needs to be decided how the output at each cell depends on its input. To keep quantities from growing out of bounds, each cell is treated like an “integrate and fire” neuron which takes in weighted inputs from other cells and produces a response close to one if the total input is greater than a certain threshold value, and otherwise has output close to zero. This could be accomplished by using a step function

$$H(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}.$$

Then $H(\text{input} - \theta)$ would be the desired output function for a cell. Unfortunately, the method used to minimize the error function E requires that the

output function be differentiable, so in place of $H(z)$ the function

$$f(z) = \frac{1}{1 + e^{-z}} \quad (1)$$

is used. The function f is called a **sigmoidal function**, and has a stylized S-shape like a smoothed version of the function $H(z)$. If z is the input to a cell, the function $f(z - \theta)$ shown in Figure 2 has the desired shape for the output. Notice that if C is a large positive constant, the function $f(C(z - \theta))$ lies even closer to the step function H .

The function f has an easily computable derivative:

$$\begin{aligned} f'(z) &= \frac{d}{dz}((1 + e^{-z})^{-1}) = -(1 + e^{-z})^{-2} \frac{d}{dz}(e^{-z}) \\ &= \frac{e^{-z}}{(1 + e^{-z})^2} = \left(\frac{1}{1 + e^{-z}} \right) \left(\frac{e^{-z}}{1 + e^{-z}} \right) = f(z)(1 - f(z)). \end{aligned}$$

Figure 2 compares the graphs of $H(z - \theta)$ and $f(z - \theta)$ with θ arbitrarily chosen to be 2.

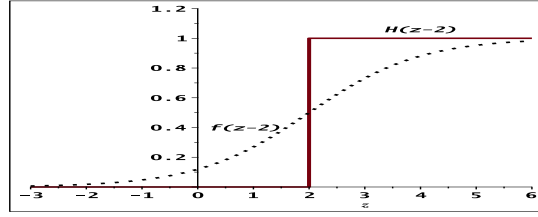


Figure 2: Graphs of $H(z - 2)$ and $f(z - 2)$

The method used for minimizing the error function E is called the **Method of Steepest Descent**. If G is a differentiable function of n variables, say $G = G(v_1, v_2, \dots, v_n)$, then at any point (v_1, v_2, \dots, v_n) , it can be shown mathematically that the gradient vector $\vec{\nabla}G = \left(\frac{\partial G}{\partial v_1}, \frac{\partial G}{\partial v_2}, \dots, \frac{\partial G}{\partial v_n} \right)$ points in the direction of greatest increase in the function G . To go in the direction of greatest decrease in the function, one would go in the direction of $-\vec{\nabla}G$. Since the aim is to make the error as close to zero as possible, and the mean square error is always positive (a sum of squares), it makes sense to go a small way in the direction of $-\vec{\nabla}G$ after each presentation of a test vector. Going too far could possibly take you beyond the minimum, so letting each variable be incremented by the formula $v_i = v_i - \eta \frac{\partial G}{\partial v_i}$ is done by choosing a small positive constant for η and experimenting to see how it works.

We will now apply this method to minimize the error function $\bar{E} = \frac{1}{2}E$. The multiplier $\frac{1}{2}$ is used to simplify the calculation, and minimizing a positive quantity $\frac{1}{2}E$ is equivalent to minimizing E . The function \bar{E} depends on all of the weights: $w_{rs}^h, 1 \leq r \leq L, 1 \leq s \leq N$ and $w_{rs}^o, 1 \leq r \leq M, 1 \leq s \leq L$.

The computation of $\frac{\partial \bar{E}}{\partial w_{rs}}$ is most efficient if it is done first for the weights in the upper level layer and then for the layers below. Some quantities are reused. This is why the method was given the name “back” propagation.

To derive the formulas for incrementing the weights, we are going to need the following definitions. These hold each time an input vector \vec{X}_p is presented at the input layer. Note that the function f is the sigmoidal function (1) defined above.

$$net_{pj}^h = \sum_{i=1}^N w_{ji}^h x_{pi} = \text{input to cell } j \text{ in layer } h.$$

$$i_{pj} = f(net_{pj}^h) = \text{output at cell } j \text{ in layer } h.$$

$$net_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} = \text{input to cell } k \text{ in layer } o.$$

$$o_{pk} = f(net_{pk}^o) = \text{output of cell } k \text{ in layer } o.$$

Incrementing the weight matrix between layer h and layer o

Let the error function be $\bar{\mathbf{E}} = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2$.

For each weight w_{rs}^o in the matrix

$$\mathbf{W}^o = \begin{pmatrix} w_{11}^o & w_{12}^o & \cdots & w_{1L}^o \\ w_{21}^o & w_{22}^o & \cdots & w_{2L}^o \\ \vdots & \vdots & \vdots & \vdots \\ w_{M1}^o & w_{M2}^o & \cdots & w_{ML}^o \end{pmatrix}$$

we need to compute $\frac{\partial \bar{\mathbf{E}}}{\partial w_{rs}^o}$. It is clear that the only term in $\bar{\mathbf{E}}$ that contains the variable w_{rs}^o is the term $\frac{1}{2}(y_{pr} - o_{pr})^2$. For all of the other terms, the partial with respect to w_{rs}^o is zero. Therefore, using the chain rule for differentiation,

$$\frac{\partial \bar{\mathbf{E}}}{\partial w_{rs}^o} = -\frac{1}{2} \cdot 2(y_{pr} - o_{pr}) \cdot \frac{\partial o_{pr}}{\partial w_{rs}^o}.$$

Using the formula

$$o_{pr} = f(net_{pr}^o) = f(w_{r1}^o i_{p1} + w_{r2}^o i_{p2} + \cdots + w_{rs}^o i_{ps} + \cdots + w_{rL}^o i_{pL})$$

we see, using the chain rule for differentiation once more, that

$$\frac{\partial o_{pr}}{\partial w_{rs}^o} = f'(net_{pr}^o) \cdot i_{ps} = f(net_{pr}^o)(1 - f(net_{pr}^o))i_{ps} = o_{pr}(1 - o_{pr})i_{ps}.$$

This implies that

$$\frac{\partial \bar{\mathbf{E}}}{\partial w_{rs}^o} = -(y_{pr} - o_{pr})o_{pr}(1 - o_{pr})i_{ps}.$$

We will define the quantities $\delta_{pr}^o = (y_{pr} - o_{pr})o_{pr}(1 - o_{pr})$, $r = 1 \cdots M$ and save them in a vector $\vec{\delta}$, since they will be used in incrementing the lower level weights.

Note that we want to move in the negative gradient direction, and there is a minus sign in $\frac{\partial \bar{\mathbf{E}}}{\partial w_{rs}^o}$; therefore, the formula for incrementing the weight w_{rs}^o is

$$w_{rs}^o(t+1) = w_{rs}^o(t) + \eta \delta_{pr}^o i_{ps}.$$

DO NOT ADD THESE INCREMENTS TO THE WEIGHTS yet. The increments to the lower level weights need to use the *current* values of all of the weights.

Incrementing the weights between layer i and layer h

Now consider a weight w_{st}^h on the connection from cell t in the input layer to cell s in the hidden layer. We want to compute $\frac{\partial \bar{\mathbf{E}}}{\partial w_{st}^h}$. Writing

$$\bar{\mathbf{E}} = \frac{1}{2} \left((y_{p1} - o_{p1})^2 + (y_{p2} - o_{p2})^2 + \cdots + (y_{pM} - o_{pM})^2 \right),$$

it is clear that each term in the sum has w_{st}^h in it, because each output cell is connected to neuron s in layer h , and hence to neuron t in the input layer. This implies that

$$\frac{\partial \bar{\mathbf{E}}}{\partial w_{st}^h} = -2 \cdot \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial w_{st}^h}.$$

Using $o_{pk} = f(\sum_{j=1}^L w_{kj}^o i_{pj})$ and applying the chain rule for differentiation once more,

$$\frac{\partial o_{pk}}{\partial w_{st}^h} = f\left(\sum_{j=1}^L w_{kj}^o i_{pj}\right) (1 - f\left(\sum_{j=1}^L w_{kj}^o i_{pj}\right)) \frac{\partial (\sum_{j=1}^L w_{kj}^o i_{pj})}{\partial w_{st}^h}.$$

This makes

$$\frac{\partial \bar{\mathbf{E}}}{\partial w_{st}^h} = - \sum_{k=1}^M (y_{pk} - o_{pk}) o_{pk} (1 - o_{pk}) \frac{\partial}{\partial w_{st}^h} \left(\sum_{j=1}^L w_{kj}^o f\left(\sum_{i=1}^N w_{ji}^h x_{pi}\right) \right).$$

The only term in the final sum that contains w_{st}^h is the term with $j = s$ and $i = t$; that is, the term

$$T = \frac{\partial}{\partial w_{st}^h} \left(w_{ks}^o f\left(\sum_{i=1}^N w_{si}^h x_{pi}\right) \right).$$

One more use of the chain rule, using the fact that w_{st}^h only appears in one term in the inner sum, gives

$$T = w_{ks}^o i_{ps} (1 - i_{ps}) x_{pt}.$$

Therefore, the desired increment for the weight w_{st}^h can be written as

$$-\frac{\partial \bar{E}}{\partial w_{st}^h} = i_{ps}(1 - i_{ps})x_{pt} \left(\sum_{k=1}^M (y_{pk} - o_{pk})o_{pk}(1 - o_{pk})w_{ks}^o \right).$$

Using the stored values $\delta_{pk}^o = (y_{pk} - o_{pk})o_{pk}(1 - o_{pk})$, this means that we should increment w_{st}^h by

$$\Delta w_{st}^h = \left(\sum_{k=1}^M \delta_{pk}^o w_{ks}^o \right) i_{ps}(1 - i_{ps})x_{pt};$$

that is,

$$w_{st}^h(t+1) = w_{st}^h(t) + \eta \Delta w_{st}^h.$$

Training the system

Two matrices, an $M \times L$ matrix W^o and an $L \times N$ matrix W^h , can be defined and initialized. One suggested way to initialize the individual weights is by setting each one equal to a random value drawn from a normal distribution with mean equal to 0.5. Given a set of P input vectors \vec{X}^p together with their desired outputs \vec{Y}^p , these are presented randomly to the system. The values of \vec{i}^p and \vec{o}^p are each obtained by a single matrix multiplication and a single application of the function f to each element of the resulting vector product. For example,

$$\vec{i}^p = \begin{pmatrix} i_{p1} \\ i_{p2} \\ \vdots \\ i_{pL} \end{pmatrix} = \begin{pmatrix} f(net_{p1}^h) \\ f(net_{p2}^h) \\ \vdots \\ f(net_{pL}^h) \end{pmatrix} \text{ where } \begin{pmatrix} net_{p1}^h \\ net_{p2}^h \\ \vdots \\ net_{pL}^h \end{pmatrix} = W^h \vec{X}^p.$$

Similarly, applying f to the elements of the vector $W^o \vec{i}^p$ produces the output vector \vec{o}^p .

After updating the weight matrices W^o and W^h , the error \bar{E} can be computed. The labeled inputs should be randomly presented, and the weights incremented, until the value of \bar{E} stabilizes (hopefully at a very small value). At that point the system can be used to recognize the given patterns. It may also recognize patterns which are slight variations of the patterns it was trained on.

Remember that the small constant η , used in the learning process, can be varied to see if better results can be obtained. Other ways to improve the results involve increasing the size of the hidden layer. With a fair amount of work, additional hidden layers may be added to the system. This has led to the idea of “deep learning systems”.