CSE5230: Back-propagation

MONASH UNIVERSITY

Faculty of Information Technology

CSE5230 Data Mining

This document presents an explanation of the mathematics which underpin the back-propagation algorithm (Rumelhart et al.; 1986). The key mathematical concepts required are *partial derivatives*, which are explained in appendix A, and the *chain rule*, which is explained in appendix B.

1 Definition of the network

Consider the standard feed-forward neural network shown in Figure 1, also known as a multilayer perceptron (MLP). Notice that the nodes of the input layer are shown as simple black circles. This is to indicate that no processing occurs in these nodes: they serve only to introduce the inputs to the network.

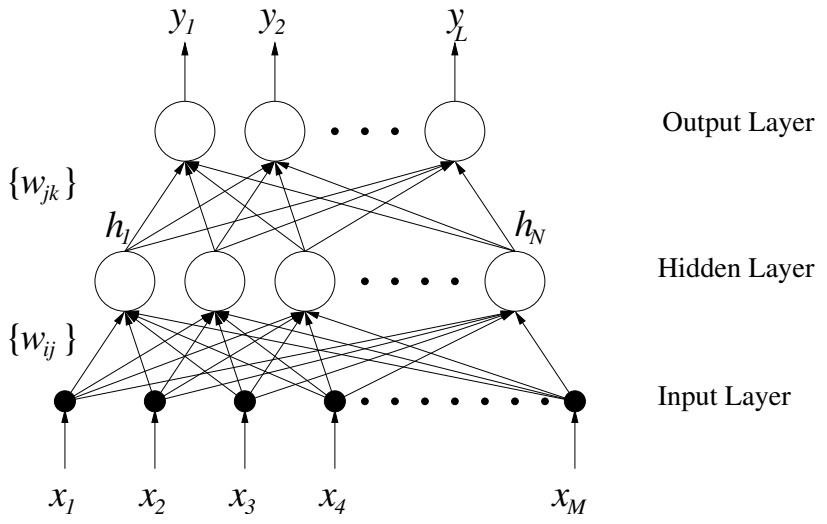

Figure 1: Multilayer Perceptron: a standard feed-forward neural network

Each node computes the weighted sum of its inputs, net, and uses this as the input to its transfer function $f()$. In the classic MLP, the transfer function is the sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Consider a node k in the output layer. Its output, y_k, is given by

$$y_k = f(net_k),$$

where net_k, its *net activation*, is the weighted sum of the outputs of the nodes h_j in the hidden layer:

$$net_k = \sum_{j'=1}^{N} w_{j'k} h_{j'}. \tag{2}$$

Likewise, the output of each node j in the hidden layer is

$$h_j = f(net_j),$$

where

$$net_j = \sum_{i'=1}^{M} w_{i'j} x_{i'} \tag{3}$$

and the x_i are the inputs to the network.

2 Total squared error and gradient descent

For convenience, we can consider the inputs to the network as an *input vector* $\mathbf{x}$, where

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_M \end{bmatrix}^T.$$

Likewise, the output of the network can be considered to be an *output vector* $\mathbf{y}$, where

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_L \end{bmatrix}^T.$$

The *training set* for the network can then be considered to be a set of pairs of K input vectors $\mathbf{x}_l$ and desired output vectors $\mathbf{d}_l$:

$$\text{training set} = \{(\mathbf{x}_1, \mathbf{d}_1), (\mathbf{x}_2, \mathbf{d}_2), \dots, (\mathbf{x}_l, \mathbf{d}_l), \dots, (\mathbf{x}_K, \mathbf{d}_K)\}.$$

Whenever an input vector from the training set $\mathbf{x}_l$ is applied to the network, the network will produce an actual output $\mathbf{y}_l$ according to the equations in section 1. We may thus define the *squared error* for that input vector by summing over the squared errors at each output node:

$$[\text{squared error}]_l = \frac{1}{2} \sum_{k'=1}^{L} (y_{k'} - d_{k'})^2$$

The factor of $1/2$ in front of the sum is introduced for computational convenience only—the aim will be to minimize the error; multiplying by a constant makes no difference to this goal.
We can thus define the *total squared error* E by summing over the all the input/output pairs in the training set:

$$E = \frac{1}{2} \sum_{l'=1}^{K} \sum_{k'=1}^{L} (y_{k'l'} - d_{k'l'})^2 \tag{4}$$

Our aim in training the network is to minimize E, by finding an appropriate set of weights $\{\{w_{ij}\},\{w_{jk}\}\}$. We will do this by using a *gradient descent* algorithm: we will find out which direction is "downhill" on the error surface and modify each weight w so that we take a step in that direction. Mathematically, this means that each weight w will be modified by a small amount Δw in the direction of decreasing E:

$$w(t+1) = w(t) + \Delta w(t), \quad \text{where} \quad \Delta w(t) = -\epsilon \left.\frac{\partial E}{\partial w}\right|_t . \tag{5}$$

Here $w(t)$ is the weight at time t and $w(t+1)$ is the updated weight. Equation 5 is called the *generalized delta rule*. We see that the crucial thing we need in order to be able to perform gradient descent is the *partial derivative* $\frac{\partial E}{\partial w}$ of the error with respect to each weight.
The generalized delta rule is often augmented with a "momentum" term, which can increase the learning rate and help to avoid oscillations:

$$\Delta w(t) = -\epsilon \left.\frac{\partial E}{\partial w}\right|_t + \alpha \Delta w(t-1) \tag{6}$$

The magnitude of α determines the effect past weight changes have on the current direction of change in weight space. This is known as the "heavy ball method" in numerical analysis (; maintainer).

3 Finding the partial derivatives with respect to the weights

3.1 Weights between the hidden and output layers

Let us begin by considering a weight w_{jk} between a hidden node j and and output node k. We wish to find $\frac{\partial E}{\partial w_{jk}}$. Using the *chain rule*, we may write

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}}. \tag{7}$$

Equation 4 tells us that

$$\frac{\partial E}{\partial y_k} = y_k - d_k. \tag{8}$$

Note that taking the partial derivative selects the particular term of the sum where $k' = k$, since only the output of node k depends on w_{jk}. (The sum over l has been suppressed for clarity).
Using equation 25, derived in appendix C on the derivative of the sigmoid, we see that[1]

$$\frac{\partial y_k}{\partial net_k} = y_k(1 - y_k). \tag{9}$$

From equation 2, we obtain

$$\frac{\partial net_k}{\partial w_{jk}} = h_j \tag{10}$$

[1] Note that the choice of the sigmoid (equation 1) for the transfer function $f()$ is not necessary for back-propagation to work. It is only necessary that $f()$ have a bounded derivative.

(again taking the partial derivative has selected a particular term from the sum). Substituting equations 8, 9 and 10 into equation 7, we see that

$$\frac{\partial E}{\partial w_{jk}} = (y_k - d_k) y_k (1 - y_k) h_j. \tag{11}$$

We have thus found the partial derivative of the error E with respect to weight w_{jk} in terms of known quantities and can employ this result in equation 5 in order to perform gradient descent for the weights $\{w_{jk}\}$ between the hidden and output layers.

3.2 Weights between the input and hidden layers

We now consider the weights w_{ij} between the input layer and the hidden layer. Again we start from equation 4 and apply the chain rule to find an expression for $\frac{\partial E}{\partial w_{ij}}$:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k'=1}^{L} \frac{\partial E}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial net_{k'}} \frac{\partial net_{k'}}{\partial h_j} \frac{\partial h_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}. \tag{12}$$

Note that on this occasion taking the partial derivative does not select a particular k' from the sum, since all the outputs $y_{k'}$ depend on w_{ij}, as shown in Figure 2.

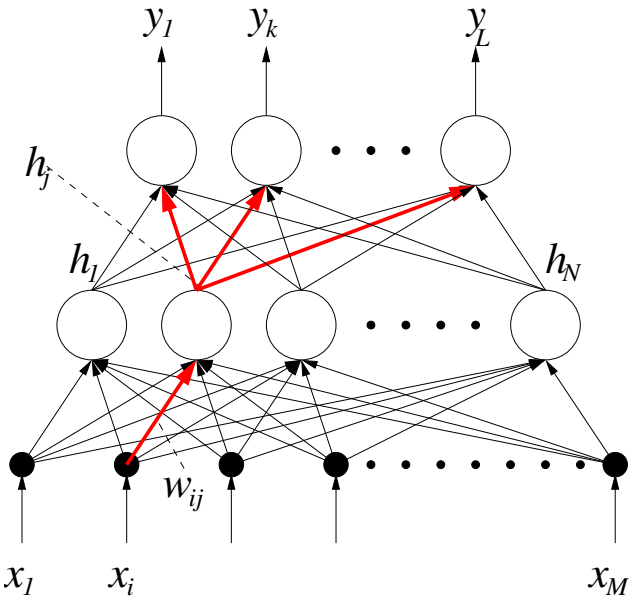

Figure 2: All outputs y_k depend on each weight w_{ij} between the input and the hidden layer.

We have already found the first two terms of this sum in section 3.1. From equation 2 we obtain

$$\frac{\partial net_{k'}}{\partial h_j} = w_{jk'}, \tag{13}$$

and by analogy with equation 9 we see that

$$\frac{\partial h_j}{\partial net_j} = h_j(1 - h_j). \tag{14}$$

Finally, using equation 3, we find that

$$\frac{\partial net_j}{\partial w_{ij}} = x_i. \tag{15}$$

Substituting all these results into equation 12, we obtain the desired result:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k'=1}^{L} (y_{k'} - d_{k'}) y_{k'} (1 - y_{k'}) w_{jk'} h_j (1 - h_j) x_i \tag{16}$$

We have thus found the partial derivative of the error E with respect to weight w_{ij} in terms of known quantities (many of which we had already calculated in obtaining $\frac{\partial E}{\partial w_{jk}}$). Together with equation 11 this gives us all the $\frac{\partial E}{\partial w}$ needed so that equation 5 can be used to perform gradient descent for all the weights in the network.

4 Why "Back-propagation"?

The back-propagation algorithm, so named in (Rumelhart et al.; 1986), gets its name from the fact that partial derivatives of the error with respect to the activations of nodes are "propagated back" through the network to enable partial derivatives to be calculated for weights between earlier layers of the network. This can be seen more clearly by rewriting equation 12 with terms not dependent on k' outside the sum:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial h_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \sum_{k'=1}^{L} \frac{\partial E}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial net_{k'}} \frac{\partial net_{k'}}{\partial h_j}.$$

Collapsing one level of chain rule expansion and using equation 13, we obtain

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial h_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \sum_{k'=1}^{L} w_{jk'} \frac{\partial E}{\partial net_{k'}}.$$

Noting that

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial h_j} \frac{\partial h_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}},$$

we may write

$$\frac{\partial E}{\partial h_j} = \sum_{k'=1}^{L} w_{jk'} \frac{\partial E}{\partial net_{k'}}. \tag{17}$$

Equation 17 shows us that the partial derivatives of the error with respect to the activations of the output nodes are propagated back through the network (using the same weights on the connections w_{jk}) to permit the calculation of the partial derivatives of the error with respect to the outputs of the hidden

nodes. Once these are known it is trivial to find the partial derivatives of the error with respect to the input weights of these nodes using equations 14 and 15.
It is important to note that equation 17 shows us how the partial derivatives can be propagated back between *any* pair of layers: so long as $\frac{\partial E}{\partial net}$ is known at a layer, then $\frac{\partial E}{\partial z}$ can be calculated for the earlier layer of network, where the $\{z\}$ are the outputs of the nodes in that layer. Back-propagation can thus cope with *any* number of layers. In practice it is never necessary to use a network with more than three layers, since three-layer MLPs have been shown to be universal approximators (Hornik et al.; 1989).

A Partial Derivatives

Partial derivatives are defined as derivatives of a function of multiple variables when all but the variable of interest are held fixed during the differentiation (Weisstein; accessed August 24, 2000).

$$\frac{\partial f}{\partial x_m} \equiv \lim_{h \to 0} \frac{f(x_1, \ldots, x_m + h, \ldots, x_n) - f(x_1, \ldots, x_m, \ldots, x_n)}{h}. \tag{18}$$

This is probably easiest to understand via examples. Here we find partial derivatives with respect to x, so the other variables (here y and z) are treated as constants:

$$\begin{array}{lll} f(x,y) = x + y & g(x,y) = x^2 + y^2 & h(x,y,z) = x^2 y + x y^2 + x^3 z^2 \\ \frac{\partial f}{\partial x} = 1 & \frac{\partial g}{\partial x} = 2x & \frac{\partial h}{\partial x} = 2xy + y^2 + 3x^2 z^2 \end{array}$$

B The Chain Rule

If $g(x)$ is differentiable at the point x and $f(x)$ is differentiable at the point $g(x)$, then $f \circ g$ is differentiable at x. Furthermore, let $y = f(g(x))$ and $u = g(x)$, then

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}. \tag{19}$$

There are a number of related results which also go under the name of "chain rules." For example, if $z = f(x, y)$, $x = g(t)$, and $y = h(t)$, then

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}. \tag{20}$$

The "general" chain rule applies to two sets of functions

$$\begin{aligned} y_1 &= f_1(u_1, \ldots, u_p) \\ &\vdots \\ y_m &= f_m(u_1, \ldots, u_p) \end{aligned} \tag{21}$$

and

$$\begin{aligned} u_1 &= g_1(x_1, \ldots, x_n) \\ &\vdots \\ u_p &= g_p(x_1, \ldots, x_n). \end{aligned} \tag{22}$$

Defining the $m \times n$ Jacobi matrix by

$$\left(\frac{\partial y_i}{\partial x_j}\right) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}, \tag{23}$$

and similarly for $(\frac{\partial y_i}{\partial u_j})$ and $(\frac{\partial u_i}{\partial x_j})$ then gives

$$\left(\frac{\partial y_i}{\partial x_j}\right) = \left(\frac{\partial y_i}{\partial u_j}\right)\left(\frac{\partial u_i}{\partial x_j}\right). \tag{24}$$

Equation 19 also applies for partial derivatives, as is indicated in matrix form by equation 24. This explanation is adapted from (Weisstein; accessed August 24, 2000).
Again, simple examples should help to made this clear. Consider the function

$$y = \sin(x^2).$$

Using the notation of equation 19, we may write

$$y = f(g(x))$$

where

$$f(u) = sin(u)$$

and

$$g(x) = u = x^2.$$

We then have

$$\frac{dy}{du} = \cos(u) \qquad\qquad \frac{du}{dx} = 2x.$$

Bringing these together and using equation 19, we obtain

$$\begin{aligned} \frac{dy}{dx} &= \frac{dy}{du} \cdot \frac{du}{dx} \\ &= \cos(u) \cdot 2x \\ &= 2x\cos(x^2). \end{aligned}$$

Here is a second example:

$$y = (x - d)^2$$

We write

$$y = f(u)$$

where

$$f(u) = u^2$$

and

$$u = x - d.$$

Differentiating,

$$\frac{dy}{du} = 2u \qquad \frac{du}{dx} = 1.$$

We thus obtain

$$\begin{aligned} \frac{dy}{dx} &= 2u \cdot 1 \\ &= 2(x - d) \end{aligned}$$

C Derivative of the sigmoid

Examining equation 1 we see that

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

therefore

$$\begin{aligned} \frac{df}{dx} &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\ &= f(x) \left(\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \right) \\ &= f(x) \left(1 - f(x) \right). \end{aligned} \tag{25}$$

It is particularly convenient computationally that the derivative of $f(x)$ can be expressed solely in terms of $f(x)$ itself, since this is usually already known.

References

Hornik, K., Stinchcombe, M. and White, H. (1989). Multilayer feedforward networks are universal approximators, *Neural Networks* **2**: 359–366.

(maintainer), W. S. S. (accessed August 11, 2000). Neural networks FAQ, `ftp://ftp.sas.com/pub/neural/FAQ.html`.

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

Weisstein, E. W. (accessed August 24, 2000). Eric Weisstein's world of mathematics, `http://mathworld.wolfram.com/`.