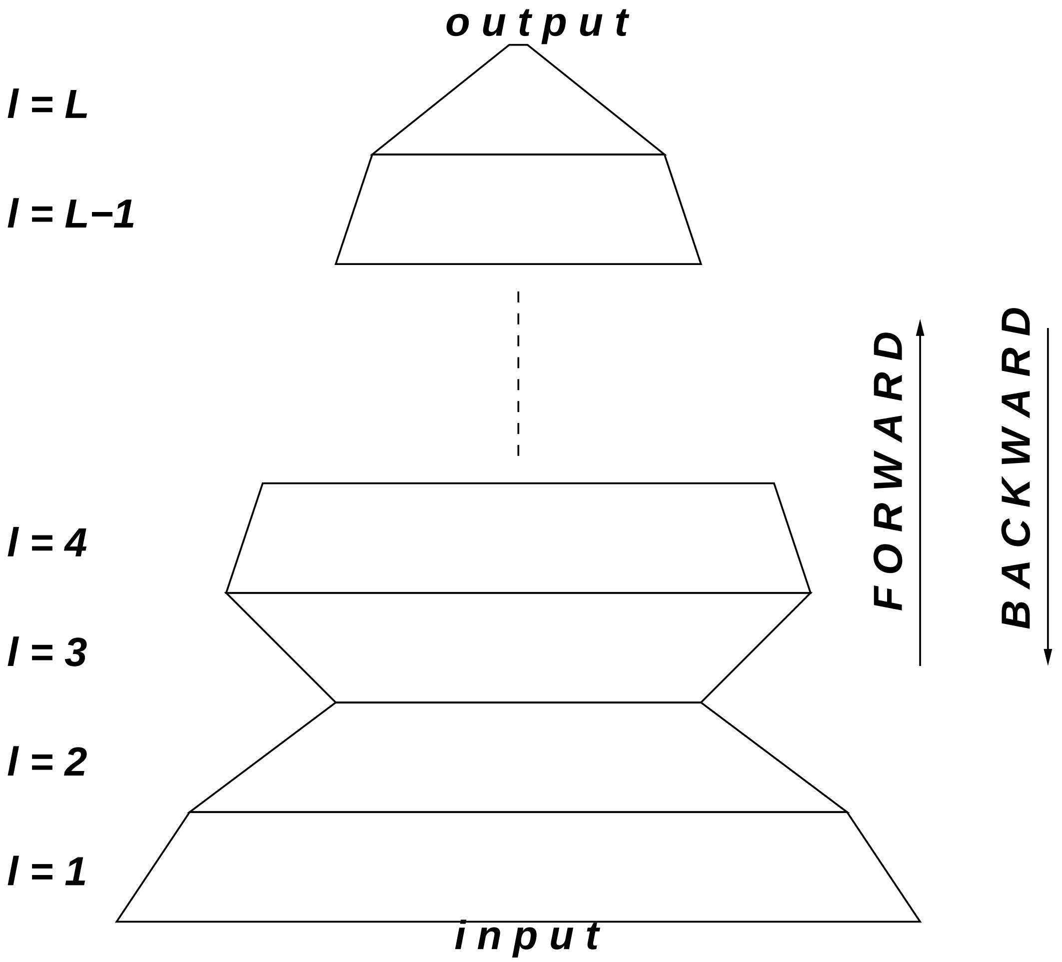
NEURAL NETWORKS AND BACKPROPAGATION

original version provided by Professor Yaser S. Abu-Mostafa in Caltech CS156;
notations slightly modified by Professor Hsuan-Tien Lin

A set of neural networks is a learning model, and backpropagation is the learning algorithm that goes with it.

The Learning Model

We consider a target function $f : \mathbb{R}^d \to \mathbb{R}$ to be learned by a *feedforward neural network* (or *multi-layer perceptron*). The network consists of L layers of neurons. Layer l ($l = 1, ..., L$) has $d^{(l)}$ neurons that connect $d^{(l-1)}$ inputs to $d^{(l)}$ outputs. The outputs of layer $l-1$ are the inputs to layer l. The inputs to the first layer ($l = 1$) are the function inputs (hence $d^{(0)} = d$) and the output of the last layer ($l = L$) is the function output (hence $d^{(L)} = 1$).



Let us denote the inputs and outputs of layer l by $x_i^{(l-1)}$ (where $i = 0, ..., d^{(l-1)}$) and $x_j^{(l)}$ (where $j = 1, ..., d^{(l)}$), respectively. Notice that the zero subscript denotes the fixed -1 'input' in each layer that represents the threshold term. The weights of the neurons in layer l are

$$w_{ij}^{(l)} \text{ for } \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

Neuron j in layer l implements the function

$$x_j^{(l)} = \varphi\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right)$$

where $\varphi(s) = \tanh(s) = (e^s - e^{-s})/(e^s + e^{-s})$. It is convenient to separate the linear and nonlinear portions of the neuron function by defining the intermediate variables $s_j^{(l)}$ such that

$$\begin{aligned} s_j^{(l)} &= \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)} \\ x_j^{(l)} &= \varphi(s_j^{(l)}) \end{aligned}$$

The overall function of the network $g_r(\mathbf{x})$ is obtained by applying $\mathbf{x}$ to the input $x_1^{(0)} \ldots x_{d^{(0)}}^{(0)}$, computing the outputs of each layer recursively from $l = 1$ to $l = L$ according to the above equation, and assigning $g_r(\mathbf{x})$ to the output of the last layer $x_1^{(L)}$. This computation proceeds in the *forward* direction, from $l = 1$ to $l = L$.

The Learning Algorithm

Given a neural network with a certain architecture, we would like to learn a target function $f : \mathbb{R}^d \to \mathbb{R}$ represented to us by examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, where $y_n = f(\mathbf{x}_n)$. We define an error function $E_n(w) = (g_r(\mathbf{x}_n) - y_n)^2$ that measures how well the network (i.e., with the current values of the weights) approximates the target function on the n^{th} example. We would like to compute the gradient of this error $\partial E_n / \partial w_{ij}^{(l)}$ for $i = 0, \ldots, d^{(l-1)}$, $j = 1, \ldots, d^{(l)}$, $l = 1, \ldots, L$. Once we have the example-wise gradient, we are going to apply stochastic gradient descent to modify the weights

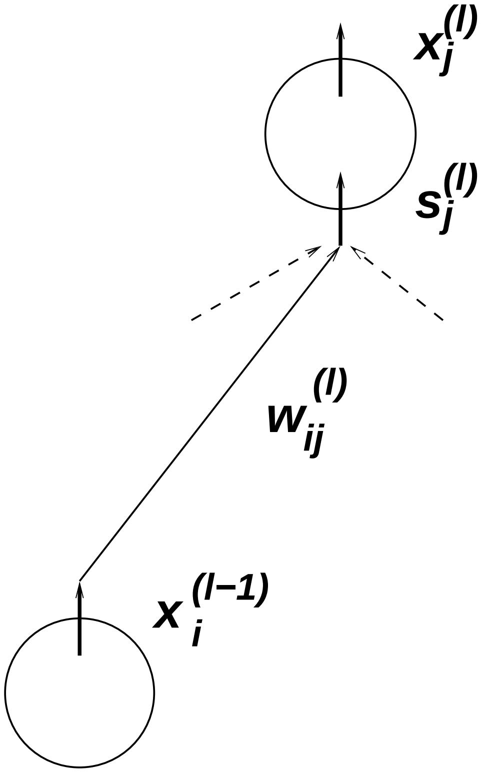
$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial E_n}{\partial w_{ij}^{(l)}}$$

where η is the learning rate. We will repeat this for a sufficiently large number of iterations.

The key to doing this efficiently is to be able to evaluate $\partial E_n / \partial w_{ij}^{(l)}$ quickly, and this is the essence of the backpropagation algorithm. We assume that we applied the input of the example to the network and carried out the forward computation to the network output. In doing so, we have computed all the intermediate $x_j^{(l)}$ and $s_j^{(l)}$. To compute the partial derivatives that we now need, we start by writing

$$\frac{\partial E_n}{\partial w_{ij}^{(l)}} = \frac{\partial E_n}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$$

which is evident since $w_{ij}^{(l)}$ affects the output only through $s_j^{(l)}$ (the linear sum involving $w_{ij}^{(l)}$ that goes through the nonlinearity $\varphi(\cdot)$ to become an input to the next layer).



We notice that $\partial s_j^{(l)} / \partial w_{ij}^{(l)} = x_i^{(l-1)}$ which is readily available, so the trick is to compute $\partial E_n / \partial s_j^{(l)}$ which we will call $\delta_j^{(l)}$. Here is how to do it. At the last layer, $l = L$, we have $E_n = (x_1^{(L)} - y)^2$ (the error measure on some example). Hence, we can compute

$$\delta_1^{(L)} = \frac{\partial E_n}{\partial s_1^{(L)}} = \frac{\partial E_n}{\partial x_1^{(L)}} \times \frac{\partial x_1^{(L)}}{\partial s_1^{(L)}}$$

The first factor is $2(x_1^{(L)} - y)$ and the second factor is $\varphi'(s_1^{(L)})$. Thus we have the delta for the last layer. We now recursively compute the deltas for layer $l-1$ given the deltas for layer l (backward).

$$\begin{aligned} \delta_i^{(l-1)} &= \frac{\partial E_n}{\partial s_i^{(l-1)}} \\ &= \sum_{j=1}^{d^{(l)}} \frac{\partial E_n}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}} \\ &= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \varphi'(s_i^{(l-1)}) \end{aligned}$$

Noticing that $\varphi'(s) = 1 - \varphi^2(s)$ and that $\varphi(s_i^{(l)}) = x_i^{(l)}$, the deltas can be expressed recursively for $l = L, L-1, ...$ as

$$\delta_i^{(l-1)} = \left(1 - (x_i^{(l-1)})^2\right) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \delta_j^{(l)}$$

All quantities are available from the forward pass, hence we can compute the deltas (notice the similarity of the delta computation in the backward pass to the x computation in the forward pass). With the deltas, we have the gradient of the error with respect to all the weights, and we can implement stochastic gradient descent.