

# Neural Networks, Forward and Back Propagation

## 1 Notation

1. LAYERS:  $0, 1, 2, \dots, L$ . Superscripts  $(l, m, n)$  denote which layer we are talking about.

INPUT LAYER  $l = 0$   
 OUTPUT LAYER  $l = L$   
 HIDDEN LAYERS  $0 < l < L$

2. NODES: the nodes in layer  $l$  are numbered  $0, 1, 2, \dots, d^l$ . The number of nodes in layer  $l$  is  $d^l + 1$ . Subscripts  $(i, j, k)$  denote which node we are talking about.

**example:**  $(\cdot)_i^l$  refers to  $(\cdot)$  in layer  $l$ , node  $i$ .

3. INPUT to node  $i$  in layer  $l$ :  $s_i^l, l = 1, \dots, L, i = 1, \dots, d^l$ .

4. OUTPUT of node  $i$  in layer  $l$ :  $x_i^l, l = 0, \dots, L, i = 0, \dots, d^l$ .

5. WEIGHT from node  $i$  in layer  $l-1$  to node  $j$  in layer  $l$ :  $w_{ij}^l, l = 1, \dots, L, i = 0, \dots, d^{l-1}, j = 1, \dots, d^l$ .

6. TRANSFORMATION/ACTIVATION function

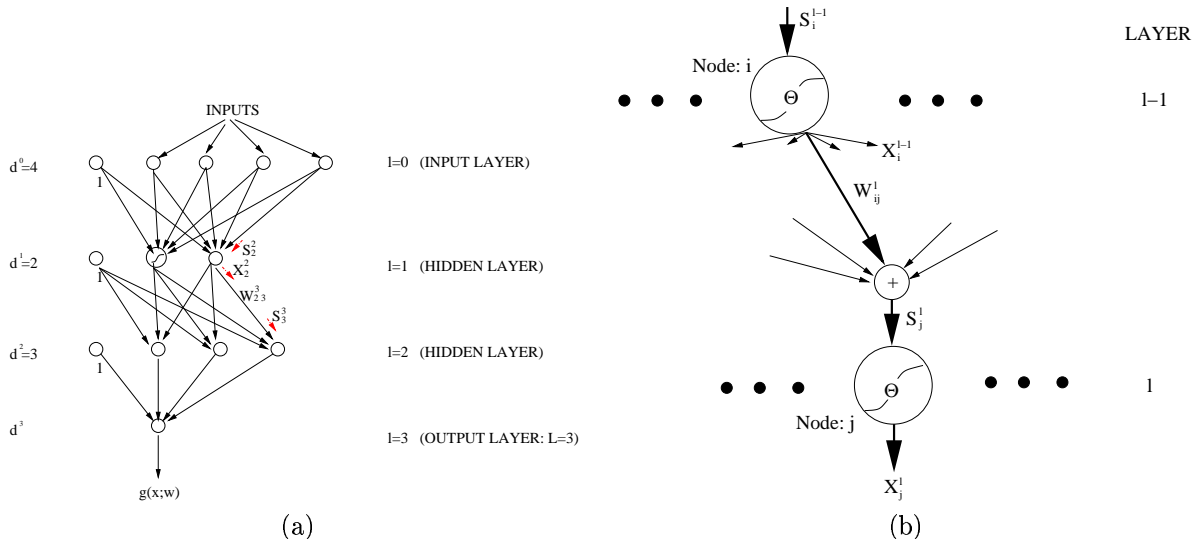
HIDDEN node:  $\Theta(\cdot) = \tanh(\cdot)$

OUTPUT node:  $S(\cdot) = \begin{cases} \tanh(\cdot) & (\pm 1 \text{ output}) \\ (\cdot) & (\text{continuous output}) \end{cases}$

7. INPUT LAYER OUTPUT:  $x_0^0 = 1$  and  $x_i^0 = x_i$  for  $i = 1, \dots, d^0 = d = \text{input dimension}$ .

8. ZERO NODE OUTPUT:  $x_0^l = 1$

9. FINAL OUTPUT  $y$ :  $y = x_1^L, d^L = 1$



In the figure, (a) gives an example Neural Network with  $L = 3, d_0 = 4, d^1 = 2, d^2 = 3$ ; (b) shows the detailed structure of a pair of nodes.

## 2 Forward Propagation: Computation of Output

We will refer to the entire set of  $w_{ij}^l$  for  $l = 1, \dots, L$ ,  $i = 0, \dots, d^{l-1}$ ,  $j = 1, \dots, d^l$  by the vector  $\mathbf{w}$ . Given a particular set of weights, the architecture defines a function of the input vector  $\mathbf{x}$ . Whilst this function is not easy to write down symbolically, its computation is easily represented in a recursive form. However while the recursive form of the function gives the most compact way of representing the function, it is more efficient and easier to compute the function from the bottom up (or inductively).

**Inductive Computation of  $y = x_1^L = g(\mathbf{x}, \mathbf{w})$ :**

1. Start with:  $x_0^0 = 1$ ,  $x_i^0 = x_i$

Inductive step:

**for  $l=1$  to  $L$**

    Compute  $x_i^l$  for  $i = 0, \dots, d^l$  as follows

- 2.

$$\begin{cases} x_0^l = 1 \\ x_i^l = \Theta(s_i^l) = \tanh(s_i^l), & 1 \leq l < L \text{ and } 1 \leq i \leq d^l \\ x_1^l = S(s_1^l), & l = L \end{cases}$$

$$s_i^l = \sum_{k=0}^{d^{l-1}} w_{ki}^l x_k^{l-1}$$

**end for loop**

3. output  $x_1^L$

It is easy to see that the computation algorithm ripples through the network computing successively

$$x_{i=0, \dots, d^0}^0 \rightarrow x_{i=0, \dots, d^1}^1 \rightarrow \dots \rightarrow x_1^L$$

which is the final output. In the process of the forward propagation, all the following quantities have been computed:

1.  $s_j^l$  for  $l = 1, \dots, L$  and  $j = 1, \dots, d^l$ .
2.  $x_j^l$  for  $l = 0, \dots, L$  and  $j = 0, \dots, d^l$ .

These will be used in the backward step to get the gradients.

### 2.1 Computation of $R_{emp}(\mathbf{w})$

Assume that we are given a data set,  $D = \{\mathbf{x}_i, y_i\}_{i=1}^N$ .

$$R_{emp}(\mathbf{w}) = \frac{1}{4N} \sum_{i=1}^N (g(\mathbf{x}_i, \mathbf{w}) - y_i)^2 = \frac{1}{4N} \sum_{i=1}^N (x_1^L(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

function  $R_{emp}(D, \mathbf{w})$

```
sum=0;
for i=1 to N
    compute  $x_j^l$  and  $s_j^l$  for input  $\mathbf{x}_i$  and  $\mathbf{w}$ 
    sum=sum+( $x_1^L - y_i$ )2/4N
end for loop
return sum
```

The goal is to find a set of weights  $\mathbf{w}$  that minimizes  $R_{emp}$ . To do this we implement the gradient descent algorithm, for which we will need derivatives.

### 3 Backpropagation: Computation of the Gradients

In order to implement the gradient descent algorithm, we need

$$\frac{\partial R_{emp}(\mathbf{w})}{\partial w_{\alpha\beta}^m}$$

which can be computed as follows:

$$\frac{\partial R_{emp}(\mathbf{w})}{\partial w_{\alpha\beta}^m} = \frac{1}{2N} \sum_{i=1}^N (x_1^L(\mathbf{x}_i, \mathbf{w}) - y_i) \frac{\partial x_1^L(\mathbf{x}_i, \mathbf{w})}{\partial w_{\alpha\beta}^m} \quad (1)$$

thus, one needs  $\partial x_1^L(\mathbf{x}_i, \mathbf{w})/\partial w_{\alpha\beta}^m$  for  $m = 1, \dots, L$ ,  $\alpha = 0, \dots, d^{m-1}$  and  $\beta = 1, \dots, d^m$ .

$$\frac{\partial x_1^L}{\partial w_{\alpha\beta}^m} = \frac{\partial x_1^L}{\partial s_\beta^m} \frac{\partial s_\beta^m}{\partial w_{\alpha\beta}^m} = \delta_\beta^m x_\alpha^{m-1} \quad (2)$$

where  $\delta_\beta^m = \partial x_1^L/\partial s_\beta^m$  and we have suppressed the dependence of the various quantities on  $\mathbf{x}_i$  and  $\mathbf{w}$ . Thus since the  $x_\alpha^m$  have been computed in the forward step, all we need to compute the gradients is to get the  $\delta_\beta^m$ 's. After some iterations of the chain rule, one finds that

$$\delta_\beta^m = \frac{\partial x_1^L}{\partial s_\beta^m} = \frac{\partial x_1^L}{\partial x_\beta^m} \frac{\partial x_\beta^m}{\partial s_\beta^m} = \frac{\partial x_\beta^m}{\partial s_\beta^m} \sum_{k=1}^{d^{m+1}} \frac{\partial x_1^L}{\partial s_k^{m+1}} \frac{\partial s_k^{m+1}}{\partial x_\beta^m} = \Theta'(s_\beta^m) \sum_{k=1}^{d^{m+1}} \delta_k^{m+1} w_{\beta k}^{m+1}$$

where  $\Theta$  is the activation function for the  $\bigcirc_\beta^m$  node. Thus we see the makings of a recursive computation. To get  $\delta_\beta^m$  all we need to know are the  $\delta_k^{m+1}$ 's for the higher layer. We thus start the recursion off at  $\delta_1^L$ .

**Backward computation of the  $\delta_\beta^m$ 's:**

1. Initialize the backward propagation.

$$\delta_1^L = \frac{\partial x_1^L}{\partial s_1^L} = S'(s_1^L) = \begin{cases} 1 & \text{if } S(x) = x \\ 1 - (x_1^L)^2 & \text{if } S(x) = \tanh(x) \end{cases}$$

where  $S(\cdot)$  is the output activation function. We consider the two cases that  $S(\cdot)$  is either  $\tanh(\cdot)$  or the identity.

2. Recursively compute  $\delta_\beta^m$  for  $m = L-1, L-2, \dots, 1$  and  $\beta = 1, \dots, d^m$  as follows

$$\delta_\beta^m = (1 - (x_\beta^m)^2) \sum_{k=1}^{d^{m+1}} \delta_k^{m+1} w_{\beta k}^{m+1}$$

For example

$$\begin{aligned} \delta_\beta^{L-1} &= (1 - (x_\beta^{L-1})^2) \sum_{k=1}^{d^L} \delta_k^L w_{\beta k}^L \\ &= (1 - (x_\beta^{L-1})^2) \delta_1^L w_{\beta 1}^L \end{aligned}$$

Once all the  $\delta_\beta^m$ 's have been calculated, it is now a simple matter to combine them with the  $x_j^m$ 's that were computed in the forward step to get the gradient in (2) which can then be used in (1).

## 4 Complete Algorithm for Gradient Descent on $R_{emp}$

Given the data set  $D = \{\mathbf{x}_i, y_i\}_{i=1}^N$ , it is necessary to perform the following iterative algorithm for obtaining a set of weights  $\mathbf{w}$  that is a local minimum of the error function  $E(\mathbf{w}) = R_{emp}(\mathbf{w})$  as follows.

1. Initialize:  $t = 0$ . Set  $\mathbf{w}(0)$  to a random starting point.
2. Determine direction to move:  $\mathbf{d}(t) = -\nabla E(\mathbf{w}(t))$ . In component form, this reads

$$d_{\alpha\beta}^l = -\frac{\partial R_{emp}(\mathbf{w})}{\partial w_{\alpha\beta}^l}$$

for all  $l = 1, \dots, L$ ,  $\alpha = 0, \dots, d^{l-1}$ ,  $\beta = 1, \dots, d^l$ .

3. Update weights:  $\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + \eta \mathbf{d}(t)$ , i.e.,

$$w_{\alpha\beta}^l(t+1) \leftarrow w_{\alpha\beta}^l(t) - \eta \frac{\partial R_{emp}(\mathbf{w})}{\partial w_{\alpha\beta}^l}$$

for all  $l = 1, \dots, L$ ,  $\alpha = 0, \dots, d^{l-1}$ ,  $\beta = 1, \dots, d^l$ .

4. Advance:  $t \leftarrow t+1$  and return to step 2 until you decide to intuitively stop.

Where, given  $D$  and  $\mathbf{w}$ , one computes  $\partial R_{emp}(\mathbf{w}) / \partial w_{\alpha\beta}^l$  in step 2 as follows

```

{
Set  $D_{\alpha\beta}^l = 0$  for all  $l = 1, \dots, L$ ,  $\alpha = 0, \dots, d^{l-1}$ ,  $\beta = 1, \dots, d^l$ .
For  $i = 1$  to  $N$ 
  Select Data point  $(\mathbf{x}_i, y_i)$ 
  Forward Step:
    Compute  $x_\alpha^l$ 's and  $s_\alpha^l$ 's for all the appropriate  $l$  and  $\alpha$  in each case.
  Backward Step:
    Compute  $\delta_\alpha^l$ 's for all the appropriate  $l$  and  $\alpha$ 
  Update:
     $D_{\alpha\beta}^l \leftarrow D_{\alpha\beta}^l + (x_1^L - y_i) \delta_\beta^l x_\alpha^{l-1} / 2N$  for all  $l = 1, \dots, L$ ,  $\alpha = 0, \dots, d^{l-1}$ ,  $\beta = 1, \dots, d^l$ .
End For Loop
Return  $\partial R_{emp}(\mathbf{w}) / \partial w_{\alpha\beta}^l = D_{\alpha\beta}^l$  for all  $l = 1, \dots, L$ ,  $\alpha = 0, \dots, d^{l-1}$ ,  $\beta = 1, \dots, d^l$ .
}

```

### 4.1 Sequential Version

What is described above is the *batch* version of the gradient descent algorithm. A frequently used approximation because of speed considerations is a *sequential* version that differs from this by changing the point at which one updates the weights. In the batch version, one computes the average gradient over the entire data set and then one updates the weights. In the sequential version one computes the gradient of the contribution to  $R_{emp}$  for a given data point and then updates  $\mathbf{w}$  according to that gradient and then one cycles through the data points. Once one has cycled through all the data points once, that constitutes one iteration which takes about the same time as one iteration of the batch version.

Operationally this means that in the above pseudo code, the step

$$D_{\alpha\beta}^l \leftarrow D_{\alpha\beta}^l + (x_1^L - y_i) \delta_\beta^l x_\alpha^{l-1} / 2N$$

is replaced by

$$w_{\alpha\beta}^l \leftarrow w_{\alpha\beta}^l - \eta (x_1^L - y_i) \delta_\beta^l x_\alpha^{l-1}$$

where one has compounded step 2 and 3 in the gradient descent algorithm and one directly returns the updated weight vector.