NEURAL NETWORKS

JHU-EPP 605-447/625-438 — 2017
Dr. Mark Fleischer

Handout On
The Feed-Forward Back-Propagation Algorithm

1 Feedforward-Back-propagation Algorithm

For your edification, below is the basic algorithm discussed in class. The basic idea here is to modify weights ω_{ij} impinging on a node j in proportion to a gradient factor associated with node j and its input value x_i. Let y_j be the output value of an output node j and let x_j be an input to a node. Bear in mind that an input to a node can either be an output of a lower level node (hence an input to the one above) or an input to the entire system of nodes.

1. Initialize weights and bias values to small random values.
2. Present Input and Desired Outputs. In other words, associate inputs with desired outputs and then stimulate inputs of NN.
3. Calculate actual outputs with given weights and biases.
4. Adapt weights $w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j x_i$, where

   $w_{ij}(t)$ is the weight from a node i in the previous layer (or an input), to node j in the next layer at time t.

   y_j is the output (activation) function value of some node j (see below).

   x_i is the output of a node i in the previous layer or is an input i to the system. In other words, the x_i's constitute inputs to a node.

   η is a gain factor.

   δ_j is a gradient estimate associated with some node j:

   - If j is an output node, then $\delta_j = y_j(1-y_j)(d_j - y_j)$.
   - If j is an output of hidden layer, then $\delta_j = x_j(1-x_j)\sum_k \delta_k w_{jk}$ where the sum is over all nodes k in the layer *above* node j.

5. Go to step 2.

2 Mathematical Elements of the Algorithm

Where does this algorithm come from? Let's examine the basic ideas here. Every node in the basic FF–NN receives a weighted input from every input signal. In some cases, these input signals come from system inputs. In other cases, they are the outputs of nodes in a lower layer that then becomes the inputs to a node in the layer above. All nodes are identical except for certain parameters (bias and weights)

2.1 Input/Output Mapping

Every node j computes an *activity* value $A_j = \sum_{i=1}^{n} w_{ij} x_i + \theta_j$ where θ_j is a bias parameter associated with node j. This activity value is sometimes referred to as the *local field* impinging on node j. The output *activation* value of node j is a function of the activity value:

$$y_j = f(A_j)$$

where the function $f(x)$ that is commonly used is called a sigmoidal function given by

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

Figure 1 depicts the Perceptron Model and figure 2 illustrates the sigmoid function.

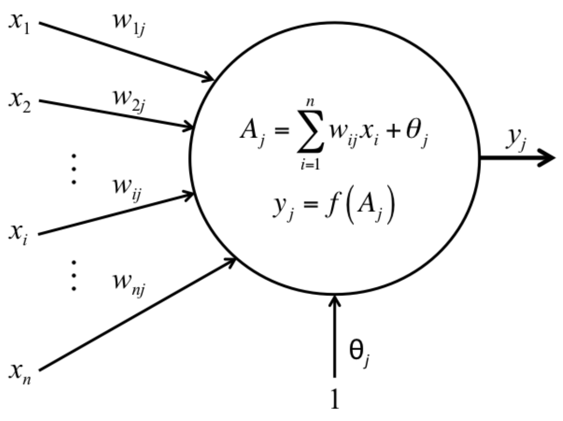

Figure 1: The Perceptron.

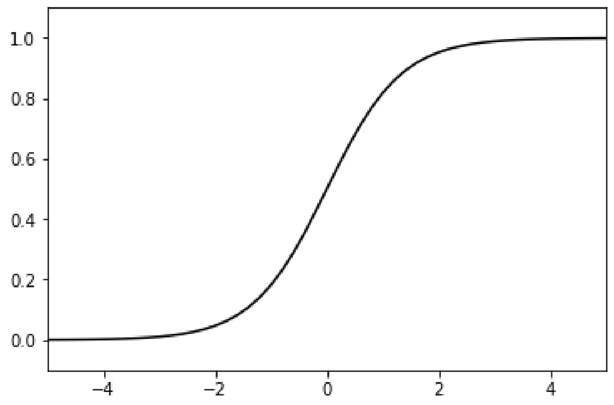

Figure 2: The Sigmoid Function.

2.2 Perceptron Delta Rule

We want to assign the weights w_{ij} so that the output of a node j is close to some *desired* output given a specified input. We design a simple rule therefore based on minimizing some error value by using the method of steepest decent. To accomplish this, define the error for a node j to be

$$e_j = d_j - y_j$$

where d_j is the desired output. For each of m output nodes $j, 1 \ldots m$ in the output layer, define the error associated with each output node j with

$$E_j = \frac{1}{2}e_j^2 = \frac{1}{2}(d_j - y_j)^2$$

Using the concept of steepest decent, we modify the weights in proportion to the gradient of the error. Thus, (this pertains to Step 4 of the algorithm):

$$\Delta w_{ij}(t+1) = w_{ij}(t+1) - w_{ij}(t) = \eta \frac{\partial E_j}{\partial w_{ij}(t)}.$$

or in vector form

$$(\Delta w_{i1}, \Delta w_{i2}, \ldots, \Delta w_{in}) = \eta \nabla(E).$$

Thus, we must evaluate the partial derivative or gradient vector. Now, as we did in class, and using the chain rule, for an output node

$$\frac{\partial E_j}{\partial w_{ij}} = \frac{\partial E_j}{\partial e_j} \times \frac{\partial e_j}{\partial y_j} \times \frac{\partial y_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}}. \tag{1}$$

Let's examine each of these four factors in turn:

2.2.1 Factor 1

Note that

$$\frac{dE_j}{de_j} = e_j$$

2.2.2 Factor 2

It is easy to see that

$$\frac{\partial e_j}{\partial y_j} = -1.$$

2.2.3 Factor 3

Now note that for the sigmoid function, if

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

then

$$\begin{aligned} 1 - f(x) &= 1 - \frac{1}{1+e^{-x}} \\ &= \frac{e^{-x}}{1+e^{-x}}. \end{aligned}$$

Note also that

$$\begin{aligned} \frac{d\,f(x)}{dx} &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\ &= \frac{f'g - fg'}{g^2} = \frac{-e^{-x}(-1)}{(1+e^{-x})^2} \\ &= \frac{e^{-x}}{(1+e^{-x})} \times \frac{1}{(1+e^{-x})} \\ &= [1 - f(x)]f(x) \end{aligned}$$

Using the notation for the output y_j, this factor reduces to

$$\frac{\partial y_j}{\partial A_j} = [1 - y_j]y_j.$$

2.2.4 Factor 4

Finally, for a node j,

$$\frac{\partial A_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{i=1}^{n} w_{ij} x_i + \theta = x_i$$

Substituting all these four factors into (1) we get

$$\frac{\partial E_j}{\partial w_{ij}} = -e_j[1 - y_j] y_j x_i$$

and letting $\delta_j = e_j[1 - y_j]y_j$ then

$$\Delta w_{ij} = \eta \frac{\partial E_j}{\partial w_{ij}} = \eta \delta_j x_i.$$

Thus, the change in the weights going to output nodes in the algorithm should be clear: Associated with each output node j is a gradient factor δ_j which is then used in conjunction with an input x_i to modify the weights w_{ij}. Notice the designation of the indices i and j for a given weight w_{ij} are associated with an input from a node i and an output of the node j. By analogy, we can further state that

$$\Delta \theta_j = \eta \frac{\partial E_j}{\partial \theta_j} = \eta \delta_j. \tag{2}$$

Can you see how we get this from equation (1)?

But what about weights that affect the local field for a *hidden node*, *i.e.,* one that does not directly produce an output of the network but instead produces an input to another node in a higher layer?

2.3 Hidden Layers

The above text tells us how to modify the weights w_{ij} associated with nodes in the output layer. We still want to calculate $\partial E_j / \partial w_{ij}$ for the weights associated with nodes in hidden layers as well. Unfortunately, the effects of weights in lower levels on the error function are more indirect because we don't have a direct functional dependence of the error at output nodes from the w_{ij} in a hidden layer. We need to modify (1) to suit our needs for the weights in a hidden layer.

To do this in a way that is clear and consistent with the algorithm presented above, we must shift our perspective a bit and define indices appropriately and conveniently. Along these lines, w_{ij} will now correspond to a weight for a *hidden* layer node that produces an output y_j and which will also be denoted as x_j to remind us that this output will be used as an *input* to the outlayer set of nodes. We define the set of output layer nodes using a generic subscript k and further index specific output layer nodes with a further subscript l where $l = 1, \ldots, m$ for m output nodes. Thus, y_{k_l} is the output of an output layer node specified by index k_l, *i.e.,* the l^{th} node in the output layer k.

Using this notational convenience, the weights for output layer nodes we previously described as w_{ij} now correspond to w_{jk_l}, and the δ_{k_l} are therefore the gradient factors associated with output nodes k_l. Figure 3 illustrates this new notational perspective.

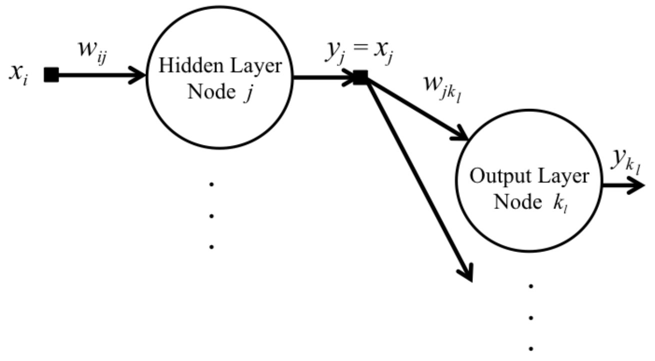

Figure 3: New index convention for hidden/output layers.

Using this notation, (1) is modified thusly:

$$\frac{\partial E_{k_l}}{\partial w_{ij}} = \frac{\partial E_{k_l}}{\partial x_j} \times \frac{\partial x_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}}. \tag{3}$$

One important consideration is to note that weights w_{ij}, the weights connecting input i to hidden layer node j, will ultimately affect *all* the output nodes in layer k, not just a single output node as suggested in figure 3. Furthermore, the outputs of *all* the output nodes will similarly affect the gradient estimates for weights w_{ij}. Consequently, it will be useful (as will become apparent) to define an overall error E as the sum of errors in the output layer, thus:

$$E = \sum_{k_l=k_1}^{k_m} E_{k_l}$$

2.3.1 Factor 1

Consider what the gradient of the error E is with respect to the *input* x_j to a node, *i.e.,* , the output of a node j from a lower level (or an input to the system). Thus,

$$\begin{aligned}\frac{\partial E}{\partial x_j} &= \frac{\partial}{\partial x_j}\sum_{k_l} E_{k_l} = \frac{\partial}{\partial x_j}\frac{1}{2}\sum_{k_l} e_{k_l}^2 \\ &= \sum_{k_l} e_{k_l}\frac{\partial e_{k_l}}{\partial x_j} \\ &= \sum_{k_l} e_{k_l}\frac{\partial e_{k_l}}{\partial A_{k_l}}\frac{\partial A_{k_l}}{\partial x_j} \qquad (4)\end{aligned}$$

Since $e_{k_l} = d_{k_l} - y_{k_l} = d_{k_l} - f_{k_l}(A_{k_l})$ where $f_{k_l}(x)$ is the activation function of node k_l, then

$$\frac{\partial e_{k_l}}{\partial A_{k_l}} = -f'_{k_l}(A_{k_l}).$$

Because the local field impinging on a node k in the layer above is $A_{k_l} = \sum_j w_{jk_l}x_j + \theta_{k_l}$,

$$\frac{\partial A_{k_l}}{\partial x_j} = w_{jk_l}$$

hence inserting these last two equations into (4) we get

$$\frac{\partial E}{\partial x_j} = -\sum_{k_l} e_{k_l}f'_{k_l}(A_{k_l})w_{jk_l} = -\sum_{k_l}\delta_k w_{jk_l}$$

where $\delta_k = e_{k_l}f'_{k_l}(A_{k_l})$ is the local gradient for some node k_l in the output layer which, presumably, has already been computed along with the values w_{jk_l}. We now can use these previously computed values for computing $\partial E/\partial x_j$!

2.4 Factors 2 & 3 in (3)

Now, the 2nd and 3rd factor in (3) simply and conveniently corresponds to the last two factors in (1). Bearing in mind that the y_j's are the outputs of the output layer and the outputs and inputs in lower layers are now designated as x_j's, these two factors become

$$\frac{\partial x_j}{\partial A_j}\frac{\partial A_j}{\partial w_{ij}} = [1 - x_j]x_j x_i$$

where here the x_i is an input to the system or is an output of another node in yet another layer of nodes. Hence

$$\frac{\partial E}{\partial \omega_{ij}} = -[1 - x_j]x_j \left(\sum_{k_l} \delta_{k_l} w_{jk_l} \right) x_i = -\delta_j x_i.$$

where for this node j in a hidden layer, $\delta_j = [1 - x_j]x_j \sum_{k_l} \delta_{k_l} w_{jk_l}$ and therefore for all weights $\Delta w_{ij} = \eta \delta_j x_i$ as indicated in the algorithm.

3 Final Thoughts

The recursive feature of this algorithm is what makes it elegant and practical. Nevertheless, there are many other ways in which to assign weights that reduce the error better. Keep in mind that because this algorithm is based on the steepest decent paradigm, there is no guarantee that it will lead to a global minimum. Also, the step size η needs to be "tuned". One approach for improving the convergence of this algorithm so as to achieve faster convergence is to use a *momentum* term where

$$\Delta \omega_{ij}(t) = \eta \delta_j x_i + \lambda \Delta \omega_{ij}(t-1)$$

where $0 < \lambda < 1$. Another approach is to use another algorithm that can possibly find those weights that yield a *global* minimum. But that's for later in the course.

Finally, remember, that these gradient descent estimates are all predicated on using the sigmoid function as the activation function. Using other activation functions will require making appropriate changes to the above equations.