Neural Network Derivations

Mickey Chao

April 2016

1 Intuition

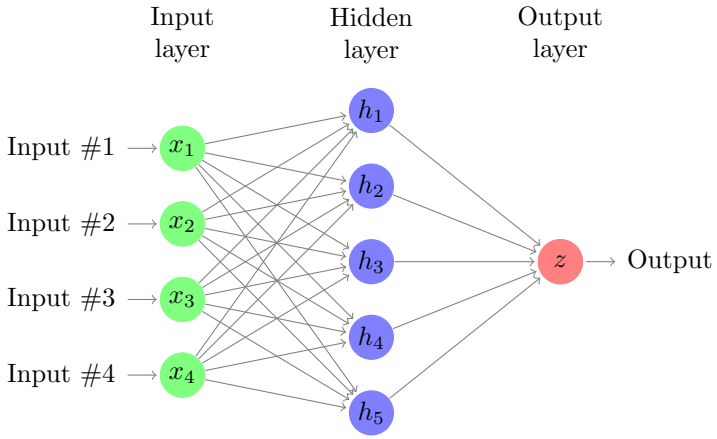

Figure 1: Example of a neural network with 1 hidden layer

An example of a neural network is presented above in figure 1. A neural network consists of nodes and edges. The nodes in an input layer can be connected to nodes in a hidden layer. Nodes in the first hidden layer can be connected to additional hidden layers and finally to the output layer, which outputs a prediction. As can be seen, each layer is completely connected to the next layer. The edges weights can be modified by the learning algorithm. This intuitively allows us to take into account many different combinations of features.

For example, suppose we are trying to predict if a student will receive an A on an upcoming test. Suppose x_1 was the feature "Cumulative GPA," x_2 was the feature "IQ Score", x_3 was the feature "BMI" (body-mass index) and x_4 was the feature "Average Hours of Sleep." It could be possible that that the hidden node h_1 will learn very high weights for inputs x_1 and x_2 and h_2, but very low weights for x_3 and x_4. If that happens, then h_1 might intuitively represent a more-complex feature "Innate Intelligence" that can be approximated by a student's cumulative GPA and IQ score. In addition, it could be possible that h_2 will learn very high weights for x_3 and x_4. If that happens, then h_2 might intuitively represent a more-complex feature "Physical Health" that can be approximated by a student's BMI and sleep amount. The output layer might then take the two more-complex features "Innate Intelligence" and "Physical Health" to make a better educated prediction as to whether or not the student will score an A on the upcoming test.

As can be seen in the previous example, neural networks can be very powerful when it comes to learning underlying features in some training data. In addition, it is also true that neural networks can learn any function on the input data, so it will also be important to regularize.

2 Forward Propagation and Making a Prediction

It is relatively straightforward to make a prediction with a neural network. We use a process called "forward propagation." The idea is to set the inputs in the input layer, then calculate the values of the nodes in the first hidden layer as a function of the values in the input layer. We then apply a thresholding function to the the values of the nodes in the first hidden layer [TODO - THRESHOLDING REASON?]. Then we can calculate the values of the nodes in the second hidden layer as a function of the values in the first hidden layer, and threshold those. This process continues until we reach the output layer.

More formally, let us make the following definitions:

- Let L be the number of hidden layers in the neural network.
- Let the value of node i in the input layer (layer 0) be x_i.
- Let the value of node i in the kth hidden layer (layer k) be $h_i^{(k)}$.
- Let the weight of the edge from node j in layer $k-1$ to node i in layer k be $W_{ij}^{(k)}$. Furthermore, $W_{ij}^{(1)}$ are the weights of edges from the input layer to the first hidden layer and $W_{ij}^{(k+1)}$ are the weights of edges from the last hidden layer to the output layer.
- Let $z_i^{(k)}$ be the value of the node i in layer k and let $h_i^{(k)}$ be the result of applying a thresholding function g to $z_i^{(k)}$. For simplicity, let $z = z_1^{(L+1)}$, or in other words, define z to be the value of the output node before it is thresholded.

Then we can perform the following calculations:

$$z_i^{(1)} = \sum_j W_{ij}^{(1)} x_i \qquad\qquad h_i^{(1)} = g(z_i^{(1)})$$

$$z_i^{(k)} = \sum_j W_{ij}^{(k)} h_i^{(k-1)} \qquad\qquad h_i^{(k)} = g(z_i^{(k)})$$

$$z = z_1^{(L+1} = \sum_j W_{1j}^{(L+1)} h_j^{(L)} \qquad\qquad h(\bar{x}; W) = g(z) = g\left(\sum_j W_{1j}^{(L+1)} h_j^{(L)}\right)$$

3 Backpropagation and Updating Weights

In order for our algorithm learn the correct weights, $W_{ji}^{(k)}$, we need to first define a cost function that we'd like to minimize. To be general, suppose we have a function $\text{Loss}(g(z), y)$ that calculates the penalty associated with our neural network outputting $h(\bar{x}; W) = g(z)$ when the actual label was y. Then we can define the cost function to be

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(h(\bar{x}; \theta), y)$$

We can apply stochastic gradient descent to minimize the loss function, but note that this will involve computing the partial derivative of $\text{Loss}(h(\bar{x}; \theta), y)$ with respect to every edge weight. Fortunately, these derivatives can be greatly simplified by rewriting them in terms of earlier derivatives. The idea will be to start computing the partial derivative of $\text{Loss}(h(\bar{x}; \theta), y)$ with respect to later components in the network, and then propagate back to the beginning while reusing calculations of derivatives of the later components.

3.1 Partial Derivative With Respect to z

The backpropagation algorithm begins by calculating the derivative of $\text{Loss}(h(\bar{x}; \theta), y)$ with respect to the final output node z. We start with respect to z because z is the closest component to the output. By a simple application of the chain rule, we get

$$\frac{\partial \text{Loss}(h(\bar{x}; \theta), y)}{\partial z} = \frac{\partial \text{Loss}(h(\bar{x}; \theta), y)}{\partial g(z)} \times \frac{\partial g(z)}{\partial z}$$

3.2 Partial Derivative With Respect To $W_{1j}^{(L+1)}$

Next, we want to calculate the derivative of $\text{Loss}(h(\bar{x};\theta),y)$ with respect to $W_{1j}^{(L+1)}$ because $W_{1j}^{(L+1)}$ is the second closest component to the output.

Applying the chain rule again, we get

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial W_{ij}^{(L+1)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial g(z)} \times \frac{\partial g(z)}{\partial z} \times \frac{\partial z}{\partial W_{1j}^{(L+1)}}$$

Notice that the expression for $\frac{\partial \text{Loss}(h(\bar{x};\theta),y}{\partial W_{ij}^{(L+1)}}$ contains the expression for $\frac{\partial \text{Loss}(h(\bar{x};\theta),y}{\partial z}$. Because we've already calculated the partial derivative with respect to z, we can rewrite the above expression as

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial W_{ij}^{(L+1)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z} \times \frac{\partial z}{\partial W_{1j}^{(L+1)}}$$

Note also that the expression for z is

$$z = \sum_j W_{1j}^{(L+1)} h_j^{(L)}$$

so $\frac{\partial z}{\partial W_{ij}^{(L+1)}} = h_j^{(L)}$. Therefore, we have

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial W_{ij}^{(L+1)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z} \times h_j^{(L)}$$

3.3 Partial Derivative With Respect To $W_{ij}^{(k)}$

As we may already be led to conclude, the derivative with respect to an edge $W_{ij}^{(k)}$ can be expressed as a product of the derivative with respect to the node $z_i^{(k)}$ and the derivative of node $z_i^{(k)}$ with respect to the the edge $W_{ij}^{(k)}$. Furthermore, because the neural network was defined to satisfy

$$z_i^{(k)} = \sum_j W_{ij}^{(k)} h_i^{(k-1)}$$

we can directly compute

$$\frac{\partial z_i^{(k)}}{\partial W_{ij}^{(k)}} = h_j^{(k-1)}$$

and if $k = 1$, which means we are in the first hidden layer, the $h_j^{(k-1)}$ term is just the input x_j.

In general, for $k \geq 2$, we have

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial W_{ij}^{(k)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k)}} \times \frac{\partial z_i^{(k)}}{\partial W_{ij}^{(k)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k)}} \times h_j^{(k-1)}$$

and for $k = 1$, we just replace $h_j^{(k-1)}$ with x_j.

3.4 Partial Derivative With Respect To $z_i^{(L)}$

We've expressed the partial derivative of the loss function with respect to the edges. Now we need to express the partial derivative of the loss function with respect to the values of the hidden nodes. Here, we will compute the expression for the partial derivative with respect to $z_i^{(L)}$ and hopefully generalize this to any arbitrary hidden node $z_i^{(k)}$.

Applying the chain rule gives

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(L)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial g(z)} \times \frac{\partial g(z)}{\partial z} \times \frac{\partial z}{\partial h_i^{(L)}} \times \frac{\partial h_i^{(L)}}{\partial g(z_i^{(L)})} \times \frac{\partial g(z_i^{(L)})}{\partial z_i^{(L)}}$$

Notice that our expression for $\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial h_i^{(L)}}$ again contains the expression for $\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{z}$. We can rewrite it as

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(L)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z} \times \frac{\partial z}{\partial h_i^{(L)}} \times \frac{\partial h_i^{(L)}}{\partial g(z_i^{(L)})} \times \frac{\partial g(z_i^{(L)})}{\partial z_i^{(L)}}$$

Recall that our expression for z was

$$z = \sum_j W_{1j}^{(L+1)} h_j^{(L)}$$

Therefore,

$$\frac{\partial z}{\partial h_i^{(L)}} = W_{1i}^{(L+1)}$$

Also recall that our expression for $h_i^{(L)}$ was

$$h_i^{(k)} = g(z_i^{(k)})$$

Therefore,

$$\frac{\partial h_i^{(L)}}{\partial g(z_i^{(L)})} = 1$$

Our final more concise expression for the partial derivative with respect to $z_i^{(L)}$ is

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(L)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z} \times W_{1i}^{(L)} \times \frac{\partial g(z_i^{(L)})}{\partial z_i^{(L)}}$$

3.5 Partial Derivative With Respect To $z_j^{(k)}$

As we did with edges, we can now generalize the derivative with respect to the value of an arbitrary hidden node. We can look at the partial derivative with respect to $z_j^{(L)}$ and attempt to generalize it.

However, note that each hidden node may be included in multiple expressions in the next layer. That is, every node $z_i^{(k+1)}$, depends on every node $z_j^{(k)}$ in the previous layer. So, the partial derivative will be slightly different here. Instead, we need to take into account $\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k+1)}}$ for all nodes i in hidden layer $k+1$.

We have

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_j^{(k)}} = \sum_i \left(\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k+1)}} \times W_{ij} \times \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}} \right)$$

In general, for $k \geq 1$, we have

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_j^{(k)}} = \sum_i \left(\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k+1)}} \times W_{ij} \times \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}} \right)$$

3.6 Combining Derivatives With Respect To Edges and Nodes

Now, starting from the back, we can backpropagate for $k = L \dots 1$ and alternate between calculating

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial W_{ij}^{(k)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k)}} \times \frac{\partial z_i^{(k)}}{\partial W_{ij}^{(k)}} = \frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k)}} \times h_j^{(k-1)}$$

and

$$\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_j^{(k)}} = \sum_i \left(\frac{\partial \text{Loss}(h(\bar{x};\theta),y)}{\partial z_i^{(k+1)}} \times W_{ij} \times \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}} \right)$$

until we reach the input layer. Then, we simply update the weights as we would when applying stochastic gradient descent on any other kind of classifier:

$$W_{ij} \leftarrow W_{ij} + \eta_k \frac{\partial \text{Loss}(h(\bar{x}^{(t)};\theta),y^{(t)})}{\partial W_{ij}}$$

where $\bar{x}^{(t)}$ is the randomly selected training sample with label $y^{(t)}$ and η_k is the learning rate.