CSE 446: Machine Learning **Lecture**

Multi-layer Perceptrons & the Back-propagation Algorithm

Instructor: Sham Kakade

Please email the staff mailing list should you find typos/errors in the notes. Thank you! The treatment in Bishop is good as well.

1 Terminology

- non-linear decision boundaries and the XOR function (see CIML)
- multi-layer neural networks & multi-layer perceptrons
- # of layers (definitions sometimes not consistent)
- input layer is x. output layer is y. hidden layers.
- activation function or transfer function or link function.
- forward propagation
- back propagation

Issues related to training are:

- non-convexity
- initialization
- weight symmetries and "symmetry breaking"
- saddle points & local optima & global optima
- vanishing gradients

2 Multi-layer perceptrons

We can specify an L-hidden layer network as follows: given outputs $\{z_j^{(l)}\}$ from layer l, the activations are:

$$a_j^{(l+1)} = \left(\sum_{i=1}^{d^{(l)}} w_{ji}^{(l+1)} z_i^{(l)} \right) + w_{j0}^{(l+1)}$$

where $w_{j0}^{(l+1)}$ is a "bias" term. For ease of exposition, we drop the bias term and proceed by assuming that:

$$a_j^{(l+1)} = \sum_{i=1}^{d^{(l)}} w_{ji}^{(l+1)} z_i^{(l)} \, .$$

The output of each node is:

$$z_j^{(l+1)} = h(a_j^{(l+1)})$$

The target function, after we go through L-hidden layers, is then:

$$\widehat{y}(x) = a^{(L+1)} = \sum_{i=1}^{d^{(L)}} w_i^{(L+1)} z_i^{(L)},$$

where saying the output is the activation at level $L+1$. It is straightforward to generalize this to force $\widehat{y}(x)$ to be bounded between 0 and 1 (using a sigmoid transfer function) or having multiple outputs. Let us also use the convention that:

$$z_i^{(0)} = x[i]$$

The parameters of the model are all the weights $w^{(L+1)}, w^{(L)}, \ldots w^{(1)}$.

3 The Back-Propagation Algorithm

In general, the loss function in an L-hidden layer network is:

$$L(w^{(1)}, w^{(2)}, \ldots, w^{(L+1)}) = \frac{1}{N} \sum_n \ell(y, \widehat{y}(x))$$

and for the special case of the square loss we have:

$$\frac{1}{N} \sum_n \ell(y_n, \widehat{y}(x_n)) = \frac{1}{N} \sum_n (y_n - \widehat{y}(x_n))^2$$

Again, we seek to compute:

$$\nabla \ell(y_n, \widehat{y}(x_n))$$

where the gradient is with respect to all the parameters.

The Forward Pass

Starting with the input x, go forward (from the input to the output layer), compute and store in memory the variables $a^{(1)}, z^{(1)}, a^{(2)}, z^{(2)}, \ldots a^{(L)}, z^{(L)}, a^{(L+1)}$

The Backward Pass

Note $\ell(y, \widehat{y})$ depends on all the parameters and we will not write out this functional dependency (e.g. $\widehat{y}$ depends on x and all the weights).

We will compute the derivates by recursion. It useful to do recursion by computing the derivates with respect to the activations and proceeding "backwards" (from the output layer to the input layer). Define:

$$\delta_j^{(l)} := \frac{\partial \ell(y, \widehat{y})}{\partial a_j^{(l)}}$$

First, let us see that if we had all the $\delta_j^{(l)}$'s then we are able to obtain the derivatives with respect to all of our parameters:

$$\frac{\partial \ell(y,\widehat{y})}{\partial w_{ji}^{(l)}} = \frac{\partial \ell(y,\widehat{y})}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$$

where we have used the chain rule and that $\frac{\partial \partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$. To see the latter claim is true, note

$$a_j^{(l)} = \sum_{c=1}^{d^{(l-1)}} w_{jc}^{(l)} z_c^{(l-1)}$$

(the sum is over all nodes c in layer $l-1$). This expression implies $\frac{\partial \partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$

Now let us understand how to start the recursion, i.e. to compute the δ's for the output layer, as there is only one node and $\widehat{y} = a^{(L+1)}$,

$$\delta^{(L+1)} = \frac{\partial \ell(y,\widehat{y})}{\partial a^{(l)}} = -(y - \widehat{y})$$

(so we don't need a subscript of j since there is only one node). Hence, for the output layer,

$$\frac{\partial \ell(y,\widehat{y})}{\partial w_j^{(L+1)}} = \delta^{(L+1)} z_j^{(L)} = -(y-\widehat{y}) z_j^{(L)}$$

where we have used our expression for $\delta^{(L+1)}$.

Thus, we know how to start our recursion. Now let us proceed recursively, computing $\delta_j^{(l)}$ using $\delta_j^{(l+1)}$. Observe that all the functional dependencies on the activations at layer l goes through the activations at $l+1$. This implies, using the chain rule,

$$\begin{aligned}
\delta_j^{(l)} &= \frac{\partial \ell(y,\widehat{y})}{\partial a_j^{(l)}} \\
&= \sum_{k=1}^{d^{(l+1)}} \frac{\partial \ell(y,\widehat{y})}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} \\
&= \sum_{k=1}^{d^{(l+1)}} \delta_k^{(l+1)} \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}}
\end{aligned}$$

To complete the recursion we need to evaluate $\frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}}$. By definition,

$$a_k^{(l+1)} = \sum_{c=1}^{d^{(l)}} w_{kc}^{(l+1)} z_c^{(l)} = \sum_{c=1}^{d^{(l)}} w_{kc}^{(l+1)} h(a_c^{(l)})$$

This implies:

$$\frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} = w_{kj}^{(l+1)} h'(a_j^{(l)}),$$

and, by substitution,

$$\delta_j^{(l)} = h'(a_j^{(l)}) \sum_{k=1}^{d^{(l+1)}} w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

which completes our recursion.

3.1 The Algorithm

We are now ready to state the algorithm.

The Forward Pass:

1. Starting with the input x, go forward (from the input to the output layer), compute and store in memory the variables $a^{(1)}, z^{(1)}, a^{(2)}, z^{(2)}, \ldots a^{(L)}, z^{(L)}, a^{(L+1)}$

The Backward Pass:

1. Initialize as follows:
$$\delta^{(L+1)} = -(y - \widehat{y}) = -(y - a^{(L+1)})$$
and compute the derivatives at the output layer:
$$\frac{\partial \ell(y, \widehat{y})}{\partial w_j^{(L+1)}} = -(y - \widehat{y}) z_j^{(L)}$$

2. Recursively, compute
$$\delta_j^{(l)} = h'(a_j^{(l)}) \sum_{k=1}^{d^{(l+1)}} w_{kj}^{(l+1)} \delta_k^{(l+1)}$$
and also compute our derivatives at layer l:
$$\frac{\partial \ell(y, \widehat{y})}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$$