Deriving Back Propagation

Reg Dodds
Department of Computer Science
University of the Western Cape
rdodds@uwc.ac.za

March 31, 2015

Abstract

These notes originated from the text transcripts of the videos Andrew Ng's Coursera course on Machine Learning. Reading the transcripts is a rapid way to review the videos. Working through these transcripts, looking at the videos, doing the programming exercises, answering the review questions, participating in the discussion forums and learning OCTAVE—especially vectorization—should familiarize you with enough concepts and commands to form a basis for proficiency in machine learning. These notes on the derivation of backpropagation are based on Thomas Schultz's book (Schultz, 2003, Chapter 2) and have been adapted to correspond with Andrew Ng's notation as much as possible.

Derivation of Back propagation

I Introduction

Conceptually, a network forward propagates activation to produce an output and it backward propagates error to determine weight changes, as shown in Figure 1. The weights on the connections between neurons mediate the passed values in both directions. The back propagation algorithm is used to

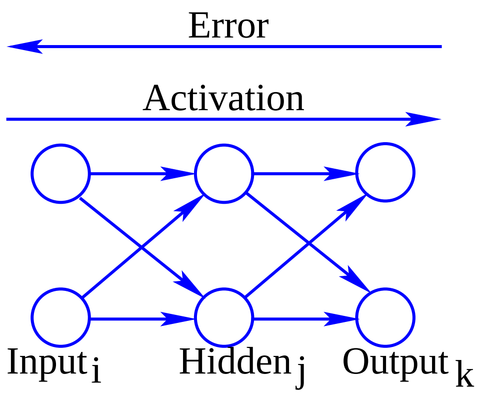

Figure 1: Neural network processing

learn the weights of a multilayer neural network with a fixed architecture. It performs gradient descent to try to minimize the sum squared error between the network's output values and the given target values.

Figure 2 on Page 3 depicts the network components which affect a particular weight change. Notice that all the necessary components are locally related to the weight being updated. This is one feature of back propagation that seems biologically plausible. However, brain connections appear to be unidirectional and not bidirectional as would be required to implement back propagation.

II Notation

For the purpose of this derivation, we will use the following notation:

- The subscript k denotes the output layer.
- The subscript j denotes the hidden layer.

- The subscript i denotes the input layer.

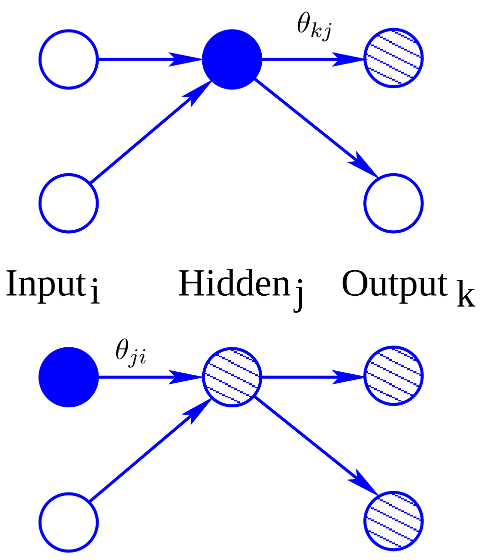

Figure 2: Neural network changes

The change to a hidden-to-output weight depends on the error—lined pattern—at the output node and the activation—solid pattern—at the hidden node. While the change to an input-to-hidden weight depends on error at the hidden node (which in turn depends on error at all the output nodes) and activation at the input node.

- θ_{kj} denotes a weight from the hidden to the output layer.
- θ_{ji} denotes a weight from the input to the hidden layer.
- a denotes an activation value.
- y denotes a target value.
- z denotes the weighted input to a node.

III Review of Calculus Rules

$$\frac{d(f+g)}{dx} = \frac{df}{dx} + \frac{dg}{dx}, \quad \frac{d(f \times g)}{dx} = g\frac{df}{dx} + f\frac{dg}{dx}, \quad \frac{d(f^n)}{dx} = nf^{n-1}\frac{df}{dx}, \quad \frac{d(e^z)}{dx} = e^z\frac{dz}{dx}.$$

IV Gradient Descent on Error

We can motivate the back propagation learning algorithm as gradient descent on sum-squared error—we square the error because we are interested in its magnitude, not its sign. The total error in a network is given by the following equation—the $\frac{1}{2}$ will simplify things later,

$$J = \tfrac{1}{2} \sum_i (y_i - a_i)^2.$$

Since we write the final activation as $h_{\mathbf{\Theta}}(x)$ this is also written as

$$J(\mathbf{\Theta}) = \tfrac{1}{2} \sum_i (y_i - h_{\mathbf{\Theta}}(x)_i)^2.$$

We want to adjust the network's weights to reduce this overall error

$$\Delta \mathbf{\Theta} \propto -\frac{\partial J}{\partial \mathbf{\Theta}}.$$

We will begin at the output layer with a particular weight.

$$\Delta \theta_{kj} \propto -\frac{\partial J}{\partial \theta_{kj}}.$$

However error is not directly a function of a weight. We expand this as follows.

$$\Delta \theta_{kj} = -\varepsilon \frac{\partial J}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial \theta_{kj}}.$$

Let's consider each of these partial derivatives in turn. Note that only one term of the J summation will have a non-zero derivative: the one associated with the particular weight we are considering.

IV.1 Derivative of the error with respect to the activation

$$\frac{\partial J}{\partial a_k} = \frac{\partial(\frac{1}{2}(y_k - a_k)^2)}{\partial a_k} = -(y_k - a_k).$$

Now we see why the $\frac{1}{2}$ in the J term was useful.

IV.2 Derivative of the activation with respect to the net input

$$\frac{\partial a_k}{\partial z_k} = \frac{\partial (1+e^{-z_k})^{-1}}{\partial z_k} = \frac{e^{-z_k}}{(1+e^{-z_k})^2}$$

We'd like to be able to rewrite this result in terms of the activation function. Notice that:

$$1 - \frac{1}{1+e^{-z_k}} = \frac{e^{-z_k}}{1+e^{-z_k}}.$$

Using this fact, we can rewrite the result of the partial derivative as:

$$a_k(1-a_k).$$

IV.3 Derivative of the net input with respect to a weight

Note that only one term of the net summation will have a non-zero derivative: again the one associated with the particular weight we are considering.

$$\frac{\partial z_k}{\partial \theta_{kj}} = \frac{\partial (\theta_{kj} a_j)}{\partial \theta_{kj}} = a_j.$$

IV.4 Weight change rule for a hidden to output weight

Now substituting these results back into our original equation we have

$$\Delta\theta_{kj} = \varepsilon \overbrace{(y_k - a_k) a_k (1-a_k)}^{\delta_k} a_j \ .$$

Notice that this looks very similar to the Perceptron Training Rule. The only difference is the inclusion of the derivative of the activation function. This equation is typically simplified as shown below where the δ term represents the product of the error with the derivative of the activation function,

$$\Delta\theta_{kj} = \varepsilon \delta_k a_j.$$

IV.5 Weight change rule for an input to hidden weight

Now we have to determine the appropriate weight change for an input to hidden weight. This is more complicated because it depends on the error at all of the nodes this weighted connection can lead to.

$$
\begin{aligned}
\Delta\theta_{ji} &\propto -\Big[\sum_k \frac{\partial J}{\partial a_k}\frac{\partial a_k}{\partial z_k}\frac{\partial z_k}{\partial a_j}\Big]\frac{\partial a_j}{\partial z_k}\frac{\partial z_k}{\partial \theta_{ji}}, \\
&= \varepsilon\Big[\sum_k \overbrace{(y_k - a_k)a_k(1-a_k)}^{\delta_k}\,\theta_{kj}\Big]a_j(1-a_j)a_i, \\
&= \varepsilon\,\overbrace{\Big[\sum_k \delta_k\theta_{kj}\Big]a_j(1-a_j)}^{\delta_j}\,a_i, \\
&= \varepsilon\delta_j a_i.
\end{aligned}
$$

(**?**)

References

Thomas R. Schultz. *Computational Developmental Psychology*. MIT Press, Cambridge, Mass., 2003.

Contents