Notes on Backpropagation

Peter Sadowski
Department of Computer Science
University of California Irvine
Irvine, CA 92697
peter.j.sadowski@uci.edu

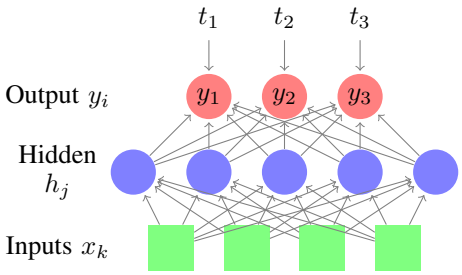
Abstract

This document derives backpropagation for some common neural networks.

1 Cross Entropy Error with Logistic Activation

In a classification task with two classes, it is standard to use a neural network architecture with a single logistic output unit and the cross-entropy loss function (as opposed to, for example, the sum-of-squared loss function). With this combination, the output prediction is always between zero and one, and is interpreted as a probability. Training corresponds to maximizing the conditional log-likelihood of the data, and as we will see, the gradient calculation simplifies nicely with this combination.

We can generalize this slightly to the case where we have multiple, independent, two-class classification tasks. In order to demonstrate the calculations involved in backpropagation, we consider a network with a single hidden layer of logistic units, multiple logistic output units, and where the total error for a given example is simply the cross-entropy error summed over the output units.



The cross entropy error for a single example with $nout$ independent targets is given by the sum

$$E = -\sum_{i=1}^{nout} \left(t_i \log(y_i) + (1 - t_i) \log(1 - y_i)\right) \tag{1}$$

where $\mathbf{t}$ is the target vector, $\mathbf{y}$ is the output vector. In this architecture, the outputs are computed by applying the logistic function to the weighted sums of the hidden layer activations, $\mathbf{s}$,

$$y_i = \frac{1}{1 + e^{-s_i}} \tag{2}$$

$$s_i = \sum_{j=1} h_j w_{ji}. \tag{3}$$

We can compute the derivative of the error with respect to each weight connecting the hidden units to the output units using the chain rule.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial w_{ji}} \tag{4}$$

Examining each factor in turn,

$$\frac{\partial E}{\partial y_i} = \frac{-t_i}{y_i} + \frac{1-t_i}{1-y_i}, \tag{5}$$

$$= \frac{y_i - t_i}{y_i(1-y_i)}, \tag{6}$$

$$\frac{\partial y_i}{\partial s_i} = y_i(1-y_i) \tag{7}$$

$$\frac{\partial s_i}{\partial w_{ji}} = h_j \tag{8}$$

where x_j is the activation of the j node in the hidden layer. Combining things back together,

$$\frac{\partial E}{\partial s_i} = y_i - t_i \tag{9}$$

and

$$\frac{\partial E}{\partial w_{ji}} = (y_i - t_i)h_j \tag{10}$$

.

The above gives us the gradients of the error with respect to the weights in the last layer of the network, but computing the gradients with respect to the weights in *lower* layers of the network (i.e. connecting the inputs to the hidden layer units) requires another application of the chain rule. This is the backpropagation algorithm.

Here it is useful to calculate the quantity $\frac{\partial E}{\partial s_j^1}$ where j indexes the hidden units, s_j^1 is the weighted input sum at hidden unit j, and $h_j = \frac{1}{1+e^{-s_j^1}}$ is the activation at unit j.

$$\frac{\partial E}{\partial s_j^1} = \sum_{i=1}^{nout}\frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial h_j}\frac{\partial h_j}{\partial s_j^1} \tag{11}$$

$$= \sum_{i=1}^{nout}(y_i - t_i)(w_{ji})(h_j(1-h_j)) \tag{12}$$

$$\frac{\partial E}{\partial h_j} = \sum_{i=1}\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial x_j} \tag{13}$$

$$= \sum_i \frac{\partial E}{\partial y_i} y_i(1-y_i)w_{ji} \tag{14}$$

Then a weight w_{kj}^1 connecting input unit k to hidden unit j has gradient

$$\frac{\partial E}{\partial w_{kj}^1} = \frac{\partial E}{\partial s_j^1}\frac{\partial s_j^1}{\partial w_{kj}^1} \tag{15}$$

$$= \sum_{i=1}^{nout}(y_i - t_i)(w_{ji})(h_j(1-h_j))(x_k) \tag{16}$$

By recursively computing the gradient of the error with respect to the activity of each neuron, we can compute the gradients for all weights in a network.

2 Classification with Softmax Transfer and Cross Entropy Error

When a classification task has more than two classes, it is standard to use a softmax output layer. The softmax function provides a way of predicting a discrete probability distribution over the classes. We again use the cross-entropy error function, but it takes a slightly different form. The softmax activation of the ith output unit is

$$y_i = \frac{e^{s_i}}{\sum_c^{nclass} e^{s_c}} \tag{17}$$

and the cross entropy error function for multi-class output is

$$E = -\sum_i^{nclass} t_i \log(y_i) \tag{18}$$

Thus, computing the gradient yields

$$\frac{\partial E}{\partial y_i} = -\frac{t_i}{y_i} \tag{19}$$

$$\frac{\partial y_i}{\partial s_k} = \begin{cases} \frac{e^{s_i}}{\sum_c^{nclass} e^{s_c}} - (\frac{e^{s_i}}{\sum_c^{nclass} e^{s_c}})^2 & i = k \\ -\frac{e^{s_i} e^{s_k}}{(\sum_c^{nclass} e^{s_c})^2} & i \neq k \end{cases} \tag{20}$$

$$= \begin{cases} y_i(1 - y_i) & i = k \\ -y_i y_k & i \neq k \end{cases} \tag{21}$$

$$\frac{\partial E}{\partial s_i} = \sum_k^{nclass} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial s_i} \tag{22}$$

$$= \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} - \sum_{k \neq i} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial s_i} \tag{23}$$

$$= -t_i(1 - y_i) + \sum_{k \neq i} t_k y_i \tag{24}$$

$$= -t_i + y_i \sum_k t_k \tag{25}$$

$$= y_i - t_i \tag{26}$$

$$\tag{27}$$

Note that this is the same formula as in the case with the logistic output units! The values themselves will be different, because the predictions $\mathbf{y}$ will take on different values depending on whether the output is logistic or softmax, but this is an elegant simplification. The gradient for weights in the top layer is again

$$\frac{\partial E}{\partial w_{ji}} = \sum_i \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}} \tag{28}$$

$$= (y_i - t_i) h_j \tag{29}$$

and for units in the hidden layer, indexed by j,

$$\frac{\partial E}{\partial s_j^1} = \sum_i^{nclass} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j^1} \tag{30}$$

$$= \sum_i^{nclass} (y_i - t_i)(w_{ji})(h_j(1 - h_j)) \tag{31}$$

3 Regression with Linear Output and Mean Squared Error

Note that performing regression with a linear output unit and the mean squared error loss function also leads to the same form of the gradient at the output layer, $\frac{\partial E}{\partial s_i} = (y_i - t_i)$.

4 Algebraic trick for cross-entropy calculations

There are some tricks to reducing computation when doing cross-entropy error calculations when training a neural network. Here are a couple.

For a single output neuron with logistic activation, the cross-entropy error is given by

$$
\begin{aligned}
E &= -\left(t \log y + (1-t) \log (1-y)\right) && (32)\\
&= -\left(t \log (\frac{y}{1-y}) + \log(1-y)\right) && (33)\\
&= -\left(t \log (\frac{\frac{1}{1+e^{-s}}}{1-\frac{1}{1+e^{-s}}}) + \log (1 - \frac{1}{1+e^{-s}})\right) && (34)\\
&= -\left(ts + \log (\frac{1}{1+e^{s}})\right) && (35)\\
&= -ts + \log (1+e^{s}) && (36)
\end{aligned}
$$

For a softmax output, the cross-entropy error is given by

$$
\begin{aligned}
E &= -\sum_i \left(t_i \log \frac{e^{s_i}}{\sum_j e^{s_j}}\right) && (37)\\
&= -\sum_i \left(t_i \left(s_i - \log \sum_j e^{s_j}\right)\right) && (38)\\
& && (39)\\
& && (40)
\end{aligned}
$$

Also note that in this softmax calculation, a constant can be added to each row of the output with no effect on the error function.