

Backpropagation in Deep Neural Nets – A Derivation

By Christopher M. Bryant | September 27, 2017

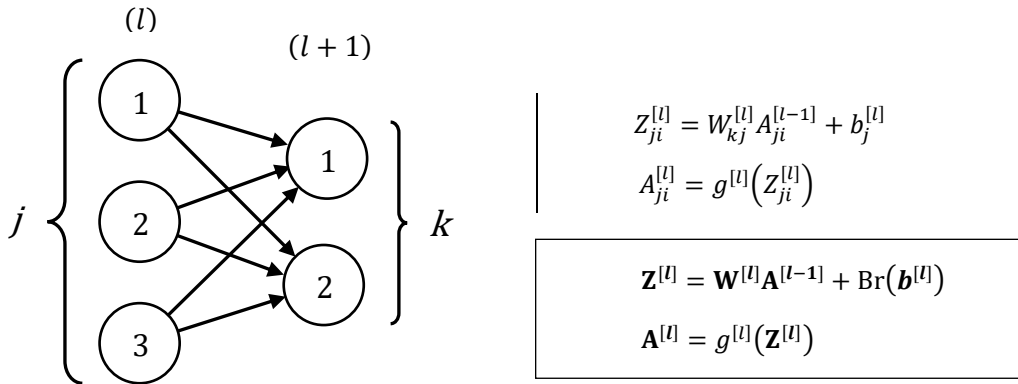
This is an updated version of my previous derivation of the backpropagation algorithm, now with clearer network structure and notation that better match the current standard in the field.

Forward Propagation:

A feature vector gets input into a neural network. The values of that vector are propagated through a neural network through a series of pairs of linear transformations and non-linear activations. Specifically, a vector $\mathbf{a}^{[l]}$ (represented by a column of nodes) in layer l is mapped to a new vector $\mathbf{z}^{[l+1]}$ in layer $l + 1$ through a linear transformation $\mathbf{z}^{[l+1]} = \mathbf{W}^{[l+1]}\mathbf{a}^{[l]} + \mathbf{b}^{[l+1]}$. The values in $\mathbf{z}^{[l+1]}$ are then activated by a non-linear function $g^{[l+1]}$ to create the new vector $\mathbf{a}^{[l+1]}$ in the same layer. With each new layer, a new set of transformations propagates the values forward through the network.

$$\left| \begin{aligned} z_j^{[l]}(i) &= W_{kj}^{[l]} a_j^{[l-1]}(i) + b_j^{[l]} \\ a_j^{[l]}(i) &= g^{[l]}(z_j^{[l]}(i)) \end{aligned} \right.$$

Rather than explicitly repeat this computation for each i , we can absorb the index and treat each stack of input vectors as a matrix. ("Br(\mathbf{x})" means that \mathbf{x} is automatically broadcast across the relevant dimensions).



Backpropagation:

The general idea of backpropagation is that to compute the derivative of the network's cost function with respect to the network parameters $\mathbf{W}^{[l]}$ and $\mathbf{b}^{[l]}$, we must employ the chain rule recursively from the last to the first layer of the network. Recall that the regularized cost function of a neural network is given by the following:

$$J_{\text{net}} = \frac{1}{m} \left\{ \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) \right\} + \frac{\lambda}{2m} \sum_{l=1}^L \sum_{p=1}^{n^{(l)}} \sum_{q=1}^{n^{(l-1)}} \{ (W_{pq}^{[l]})^2 \}$$

We can start off our analysis of backpropagation simply by trying to differentiate the cost with respect to the parameters and put it into terms that we know how to evaluate. First, we can try:

$$\frac{\partial J}{\partial W_{jk}^{[l]}}(i) = \frac{\partial J}{\partial z_j^{[l]}(i)} \frac{\partial z_j^{[l]}(i)}{\partial W_{jk}^{[l]}}$$

We will see the quantity $\partial J / \partial z_j^{[l]}(i)$ frequently in our analysis, so we're going to give it a name and its own variable right now (and work on evaluating it later). We will call this object $\delta_j^{[l]}(i)$, or equivalently, $\Delta^{[l]}$, and we will refer to it as the "error" of layer l for example i . This is the amount by which a perturbation in $z_j^{[l]}(i)$ changes the value of the unregularized cost function. In mathematical terms, $\delta_j^{[l]}(i)$ is the partial derivative of J with respect to $z_j^{[l]}(i)$. Returning to the derivation now, I'll draw a schematic diagram of two layers of the network so that it is more obvious what nodes and layers all the indices (j , k , p , and l) refer to (Note: $n_{(l)}$ is the number of nodes in layer l):

$$\begin{aligned} \frac{\partial J}{\partial W_{jk}^{[l]}}(i) &= \frac{\partial J}{\partial z_j^{[l]}(i)} \frac{\partial z_j^{[l]}(i)}{\partial W_{jk}^{[l]}} \\ &= \delta_j^{[l]} \frac{\partial}{\partial W_{jk}^{[l]}} \left\{ \sum_{p=1}^{n_{(l-1)}} W_{jp}^{[l]} a_p^{[l-1]} + b_j^{[l]} \right\} \\ &= \delta_j^{[l]} \left\{ \sum_{p=1}^{n_{(l-1)}} \delta_{kp} a_p^{[l-1]} \right\}, \text{ where } \delta_{kp} \equiv \begin{cases} 1, & k = p \\ 0, & k \neq p \end{cases} \end{aligned}$$

$$\frac{\partial J}{\partial W_{jk}^{[l]}}(i) = a_k^{[l-1]}(i) * \delta_j^{[l]}(i)$$

$$\frac{\partial J}{\partial \mathbf{W}^{[l]}}(i) = \boldsymbol{\delta}^{[l]}(i) \mathbf{a}^{[l-1]T}(i)$$

To extend this result to many samples, we can average over all i :

$$\begin{aligned} \frac{\partial J}{\partial W_{jk}^{[l]}} &= \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial W_{jk}^{[l]}}(i) \\ \frac{\partial J}{\partial W_{jk}^{[l]}} &= \frac{1}{m} \sum_{i=1}^m a_k^{[l-1]}(i) * \delta_j^{[l]}(i) \\ \frac{\partial J}{\partial W_{jk}^{[l]}} &= \frac{1}{m} \sum_{i=1}^m A_{ki}^{[l-1]} \Delta_{ji}^{[l]} \end{aligned}$$

$$\frac{\partial J}{\partial \mathbf{W}^{[l]}} = \frac{1}{m} \boldsymbol{\Delta}^{[l]} (\mathbf{A}^{[l-1]})^T$$

Solving for the derivative of cost with respect to the bias term, now, we will have to broadcast the vector $\mathbf{b}^{[l]}$ over all the dimensions spanned by $\mathbf{W}^{[l]}$. To get our final vector answer, we'll average over all the samples in the matrix:

$$\begin{aligned}
 \frac{\partial J}{\partial b_j^{[l]}}(i) &= \frac{\partial J}{\partial z_j^{[l]}(i)} \frac{\partial z_j^{[l]}(i)}{\partial b_j^{[l]}} \\
 &= \delta_j^{[l]}(i) \frac{\partial}{\partial b_j^{[l]}} \left\{ \sum_{p=1}^{n^{(l-1)}} W_{jp}^{[l]} a_p^{[l-1]}(i) + b_j^{[l]} \right\} \\
 \left(\frac{\partial J}{\partial b_j^{[l]}} \right)_i &= \Delta_{ji}^{[l]} \\
 \boxed{\frac{\partial J}{\partial \mathbf{b}^{[l]}} = \frac{1}{m} \sum_{i=1}^m \Delta_{ji}^{[l]}}
 \end{aligned}$$

Now that we have both desired derivatives in terms of only the variable $\Delta^{[l]}$ and things we know, let's put $\Delta^{[l]}$ in terms of things we know.

$$\begin{aligned}
 \delta_j^{[l]}(i) &= \frac{\partial J}{\partial z_j^{[l]}(i)} \\
 &= \sum_{k=1}^{n_l} \frac{\partial J}{\partial a_k^{[l]}(i)} \frac{\partial a_k^{[l]}(i)}{\partial z_j^{[l]}(i)} \\
 &= \frac{\partial J}{\partial a_j^{[l]}(i)} \frac{\partial a_j^{[l]}(i)}{\partial z_j^{[l]}(i)} \quad , \quad a_j^{[l]}(i) = g^{[l]}(z_j^{[l]}(i)) \\
 \delta_j^{[l]}(i) &= \frac{\partial J}{\partial a_j^{[l]}(i)} g^{[l]'}(z_j^{[l]}(i)) \\
 \Delta_{ji}^{[l]} &= \frac{\partial J}{\partial A_{ji}^{[l]}} g^{[l]'}(z_{ji}^{[l]}) \\
 \boxed{\Delta^{[l]} = \{\nabla_{\mathbf{A}^{[l]}} J\} \odot g^{[l]'}(\mathbf{Z}^{[l]})}
 \end{aligned}$$

In the second line, the chain rule required that we sum over all k , but since each activation depends explicitly on only the z -value of the *same* node, each term of the summation vanished where $k \neq j$. In the penultimate step, like in previous calculations, we absorbed the functional dependence on i into each vector to create tensors of rank 2 instead. This allowed us to better vectorize the results in the final step so that the information for all the examples could be encapsulated in matrix form. When we eventually implement this in code, this matrix vectorization will be more efficient, since the program will not have to loop explicitly over all i .

All that is left now is to solve for $\nabla_{\mathbf{A}^{[l]}} J$ in terms of things we know. For $l = L$, this is easy to evaluate since J depends explicitly on $\mathbf{A}^{[L]}$. The cost function can vary, depending on the network architecture and implementation, however, so we'll leave $\nabla_{\mathbf{A}^{[L]}} J$ in general terms (in practice, we'll know how to calculate it, so we can treat it now like we already know what it is).

It is when we tackle the case of $l \neq L$ that we finally see why backpropagation is necessarily recursive. To effectively use the chain rule, we must evaluate $\Delta^{[L-1]}$ in terms of the quantity $\Delta^{[L]}$, which we already know. Once we have found $\Delta^{[L-1]}$, we can find $\Delta^{[L-2]}$, then $\Delta^{[L-3]}$, and so on. Again, I have drawn a schematic diagram below to help keep track of indices during the derivation.

(l) $(l+1)$
 p j k

$$\frac{\partial J}{\partial A_{ji}^{[l]}} = \sum_{k=1}^{n^{(l+1)}} \frac{\partial J}{\partial Z_{ki}^{[l+1]}} \frac{\partial Z_{ki}^{[l+1]}}{\partial A_{ji}^{[l]}} \quad , \quad l \neq L$$

$$= \sum_{k=1}^{n^{(l+1)}} \Delta_{ki}^{[l+1]} \frac{\partial Z_{ki}^{[l+1]}}{\partial A_{ji}^{[l]}}$$

$$\left(Z_k^{[l+1]}(i) = \sum_{p=1}^{n^{(l)}} W_{kp}^{[l+1]} a_p^{[l]}(i) + b_k^{[l+1]} \right)$$

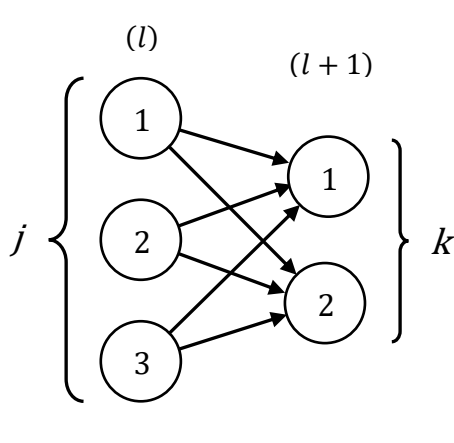
$$= \sum_{k=1}^{n^{(l+1)}} \Delta_{ki}^{[l+1]} \frac{\partial}{\partial A_{ji}^{[l]}} \left(\sum_{p=1}^{n^{(l)}} W_{kp}^{[l+1]} A_{pi}^{[l]} + b_k^{[l+1]} \right)$$

$$\frac{\partial J}{\partial A_{ji}^{[l]}} = \sum_{k=1}^{n^{(l+1)}} \Delta_{ki}^{[l+1]} W_{kj}^{[l+1]}$$

$$\nabla_{\mathbf{A}^{[l]}} J = \mathbf{W}^{[l+1]T} \Delta^{[l+1]} \quad , \quad l \neq L$$

Thus, to calculate each derivative, we need to first calculate the derivative of the next layer up, and work our way backward through the network from the output layer to the input layer. Since this process seems to be unavoidably recursive, when we implement this algorithm, we will use a for-loop to iterate over each layer.

Lastly, we can easily calculate the regularization contribution to the gradient of J_{net} , since it explicitly depends on just $\mathbf{W}_{pq}^{(l)}$ in each layer:



$$J_{\text{reg}} = \frac{\lambda}{2m} \sum_{n=1}^L \sum_{p=1}^{n(s)} \sum_{q=1}^{n(s-1)} \left\{ \left(\mathbf{W}_{pq}^{[s]} \right)^2 \right\}$$

$$\frac{\partial J_{\text{reg}}}{\partial \mathbf{W}_{jk}^{[l]}} = \frac{\lambda}{2m} \sum_{n=1}^L \sum_{p=1}^{n(s)} \sum_{q=1}^{n(s-1)} 2 \mathbf{W}_{pq}^{[s]} \delta_{jp} \delta_{kq} \delta_{nl}$$

$$\frac{\partial J_{\text{reg}}}{\partial \mathbf{W}_{jk}^{[l]}} = \frac{\lambda}{m} \mathbf{W}_{jk}^{[l]}$$

$$\frac{\partial J_{\text{reg}}}{\partial \mathbf{W}^{[l]}} = \frac{\lambda}{m} \mathbf{W}^{[l]}$$

With this, we can now complete the entire partial derivative of the network cost with respect to the \mathbf{W} parameters by simply adding the main and regularization components together:

$$\frac{\partial J_{\text{net}}}{\partial \mathbf{W}_{jk}^{[l]}} = \frac{\partial J}{\partial \mathbf{W}_{jk}^{[l]}} + \frac{\partial J_{\text{reg}}}{\partial \mathbf{W}_{jk}^{[l]}}$$

$$\frac{\partial J_{\text{net}}}{\partial \mathbf{W}^{[l]}} = \frac{\partial J}{\partial \mathbf{W}^{[l]}} + \frac{\partial J_{\text{reg}}}{\partial \mathbf{W}^{[l]}}$$

Putting everything in the same place now:

The Equations of Backpropagation

$$\frac{\partial J_{\text{net}}}{\partial \mathbf{W}^{[l]}} = \frac{1}{m} \Delta^{[l]} (\mathbf{A}^{[l-1]})^T + \frac{\lambda}{m} \mathbf{W}^{[l]}$$

$$\frac{\partial J}{\partial \mathbf{b}^{[l]}} = \frac{1}{m} \sum_{i=1}^m \Delta_{ji}^{[l]}$$

$$\Delta^{[l]} = \{\nabla_{\mathbf{A}^{[l]}} J\} \odot g^{[l]'}(\mathbf{Z}^{[l]})$$

$$\nabla_{\mathbf{A}^{[l]}} J = \mathbf{W}^{[l+1]T} \Delta^{[l+1]}, \quad l \neq L$$

Python Implementation:

When we implement forward propagation and backpropagation in code, we'll redefine the variables like so:

$$dW[l] \equiv \frac{\partial J}{\partial \mathbf{W}^{[l]}}$$

$$db[l] \equiv \frac{\partial J}{\partial \mathbf{b}^{[l]}}$$

$$dZ[l] \equiv \Delta^{[l]}$$

$$dA[l] \equiv \nabla_{\mathbf{A}^{[l]}} J$$

Thus, in pseudocode, the equations look like the following (where `func` is $g^{[l]}'()$, the derivative of the activation function for layer l):

```
dZ[l] = dA[l] * func(Z[l])  
dW[l] = 1/m * np.dot(dZ[l], A[l-1].T) + lambda/m * W[l]  
db[l] = 1/m * np.sum(dZ[l], axis = 1, keepdims = True)  
dA[l-1] = np.dot(W[l].T, dZ[l])
```