2017

Machine Learning

with professor Andrew Ng, Stanford University

CHRISTOPHER BRYANT

COURSERA | Stanford University

Week 5: Monday, 2017_06_26

Lecture 9: Neural Networks – Learning

We will be focusing now on the application of neural networks to classification problems. Moving forward, we will be representing the number of layers in the network as L and the number of units in a given layer l (**excluding the bias unit**) as s_l. For instance, a binary classification problem has a single output (i.e. a hypothesis between 0 and 1), so $s_L = 1$. A multiclass problem has K classes in its output, so $y \in \mathbb{R}^K$ and $s_L = K$.

Cost Function

The cost function we will use for our neural network is a generalization of the cost function we used for regularized logistic regression.

- *Logistic regression*:

$$J(\theta) = -\frac{1}{m}\left\{\sum_{i=1}^{m}\left[y^{(i)}\log h_\theta^{(i)}\left(x^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta^{(i)}\left(x^{(i)}\right)\right)\right]\right\} + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- *Neural network*:

$$J_{\text{net}}(\boldsymbol{\Theta}) = -\frac{1}{m}\left\{\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log\left(h_\Theta^{(i)}\left(x^{(i)}\right)\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-\left(h_\Theta^{(i)}\left(x^{(i)}\right)\right)_k\right)\right]\right\} + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{p=1}^{s_{(l+1)}}\sum_{q=2}^{s_{(l)}+1}\left\{\left(\Theta_{pq}^{(l)}\right)^2\right\}$$

This formula looks rather complicated but all that has really changed is that we are now summing over all the outputs in the final layer and the squares of all the terms in our Θ matrices. Note that, by convention, we do not sum over the bias units ($q = 1$) because we don't want to regularize them (which would reduce them to 0).

Backpropagation Algorithm *(for a derivation, see next section)*

We'd now like to find parameters for Θ that will minimize the cost function. To utilize pre-written advanced optimization algorithms, as usual, we need to write code which computes both the cost function $J_{\text{net}}(\boldsymbol{\Theta})$ and its gradient $\partial J/\partial\Theta_{uv}^{(l)}$. Rather than compute this gradient analytically for a general set of parameters, we will instead compute it for a given set of $\Theta_{uv}^{(l)}$. To do this, we must first use $\Theta_{uv}^{(l)}$ in forward propagation to generate a set of initial activation values $a_j^{(l)}$ for all the cells in the neural network for a *single* training example.

Once the activation values are calculated, we will use the *backpropagation algorithm* to compute an "error" $\delta_j^{(l)}$ for each unit in each layer. In the final layer L, this error calculation is straightforward:

$$\boxed{\delta_j^{(L)} = a_j^{(L)} - y_j}$$

The errors in earlier layers can be found with the following formula:

$$\boldsymbol{\delta}^{(l)} = \left\{\left(\boldsymbol{\Theta}^{(l)}\right)^{\boldsymbol{T}} \boldsymbol{\delta}^{(\boldsymbol{l+1})}\right\} .* \, g'(\boldsymbol{z}^{(\boldsymbol{l})})$$

$$g'\left(\boldsymbol{z}^{(\boldsymbol{l})}\right) = \boldsymbol{a}^{(\boldsymbol{l})} .* \, (1 - \boldsymbol{a}^{(\boldsymbol{l})})\}$$

The {.∗} notation indicates elementwise multiplication (also sometimes denoted by the Hadamard product symbol ⊙), and the $\{g'(\boldsymbol{z}^{(\boldsymbol{l})})\}$ term is the derivative of the activation function g evaluated at the input values given by $\boldsymbol{z}^{(\boldsymbol{l})}$. Evaluating this derivative, since g is a sigmoid function, we end up with:

$$\boldsymbol{\delta}^{(\boldsymbol{l})} = \left(\boldsymbol{\Theta}^{(\boldsymbol{l})}_{\mathbf{back}}\right)^{\boldsymbol{T}} \boldsymbol{\delta}^{(\boldsymbol{l+1})} \odot \{\boldsymbol{a}^{(\boldsymbol{l})} \odot (1 - \boldsymbol{a}^{(\boldsymbol{l})})\}$$

This process is repeated from the back end to the front end of the network (from right to left) until the second layer is reached (there is no error associated with the training set, so the last error we need to calculate is $\boldsymbol{\delta}^{(\mathbf{2})}$). Via the chain rule, it can be shown that if we ignore regularization (i.e. $\lambda = 0$), the gradient of the cost function given a single training example is as follows:

$$\frac{\partial}{\partial \Theta^{(l)}_{pq}} J(\boldsymbol{\Theta}) = a^{(l)}_q \delta^{(l+1)}_p$$

To find the gradient given an arbitrary number of training examples, we will have to add up the contributions from each training example. To do this, we'll build up a matrix for each layer, adding to it after each training example like so (starting with a matrix of zeros):

$$\Delta^{(l)}_{pq} \coloneqq \Delta^{(l)}_{pq} + a^{(l)}_q \delta^{(l+1)}_p$$

Next, after all the training examples have been forward propagated and backpropagated, and their error contributions accumulated in $\Delta^{(l)}_{pq}$, we define a new matrix $D^{(l)}_{pq}$ as:

$$D^{(l)}_{pq} \coloneqq \frac{1}{m}\Delta^{(l)}_{pq} + \lambda\Theta^{(l)}_{pq}\,, \qquad \text{if } \; q \neq 1$$

$$D^{(l)}_{pq} \coloneqq \frac{1}{m}\Delta^{(l)}_{pq} \qquad\quad , \qquad \text{if } \; q = 1$$

Notice that during regularization, we don't add the contributions from the bias units. It can be shown that this matrix is the desired gradient of the cost function:

$$\frac{\partial}{\partial \Theta^{(l)}_{pq}} J(\boldsymbol{\Theta}) = D^{(l)}_{pq}$$

Backpropagation Intuition/Derivation
(see http://neuralnetworksanddeeplearning.com/chap2.html)

The backpropagation algorithm is quite a bit more mathematically complicated than linear or logistic regression, so we'll pause here to try to gain some intuition for how these steps in the algorithm fit together and why they work. (Even Prof. Ng, who has used backpropagation for many years, sometimes finds backpropagation a bit difficult to understand, so it's okay if it still seems somewhat magical to us—we'll still be able to successfully implement it).

The goal of backpropagation: now that we have defined the cost function for a neural net, compute the **gradient** of this cost function with respect to the network's parameters; i.e.:

$$J_{\text{net}}(\boldsymbol{\Theta}) = -\frac{1}{m}\left\{\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log\left(h_\Theta^{(i)}(\boldsymbol{x}^{(i)})\right)_k + (1-y_k^{(i)})\log\left(1-\left(h_\Theta^{(i)}(\boldsymbol{x}^{(i)})\right)_k\right)\right]\right\} + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{p=1}^{s_{(l+1)}}\sum_{q=2}^{s_{(l)}+1}\left\{\left(\Theta_{pq}^{(l)}\right)^2\right\}$$

$$\frac{\partial}{\partial\Theta_{uv}^{(l)}}J_{\text{net}}(\boldsymbol{\Theta}) = ???$$

The gradient of the cost J_{net} is the collection of partial derivatives of J_{net} with respect to the parameters $\boldsymbol{\Theta}$, but it's difficult to see from the neural network cost function above how exactly this J_{net} depends on the network's individual parameters. Since differentiation is linear, we can simplify the problem by separating the cost function into smaller pieces, differentiating those pieces, and summing the results together at the end (rather than tackling the whole beast all at once). First, let's rewrite J_{net}:

$$J_{\text{net}} = \frac{1}{m}\left\{\sum_{i=1}^{m}J(i)\right\} + J_{\text{reg}},$$

$$J(i) \equiv -\sum_{k=1}^{K}\left[y_k^{(i)}\log\left(h_\Theta^{(i)}(\boldsymbol{x}^{(i)})\right)_k + (1-y_k^{(i)})\log\left(1-\left(h_\Theta^{(i)}(\boldsymbol{x}^{(i)})\right)_k\right)\right], \quad \text{and}$$

$$J_{\text{reg}} \equiv \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{p=1}^{s_{(l+1)}}\sum_{q=2}^{s_{(l)}+1}\left\{\left(\Theta_{pq}^{(l)}\right)^2\right\}$$

We'll ignore the regularization component J_{reg} of the cost for now, and just focus on the difficult problem of differentiating $J(i)$. For the remainder of our discussion here, I will drop the (i)-argument for notational simplicity, and we will consider the contribution J to the cost given by just *a single training example* (we'll add these all up at the end to get the full quantity we want).

To find the partial derivatives of J in terms of the theta parameters, our goal is to use the **chain rule** to write our desired quantity as a product of quantities we know how to compute. We will start off by defining an "error" value $\delta_j^{(l)}$ at a given node j in layer l as the rate of change of the cost with respect to a change in the weighted input value $z_j^{(l)}$ at that node.

To warm up, we will calculate the error $\delta_j^{(L)}$ of the output layer given our specific cost function J. This is relatively easy to calculate, since J depends on $z_j^{(L)}$, the outermost weighted inputs, in a simple fashion:

$$
\begin{aligned}
J &= -\sum_{k=1}^{K}\left[y_k \log\left(h_\Theta(\boldsymbol{x})\right)_k + (1-y_k)\log\left(1-\left(h_\Theta(\boldsymbol{x})\right)_k\right)\right] \\
&= -\sum_{k=1}^{K}\left[y_k \log\left(g\left(z_k^{(L)}\right)\right)_k + (1-y_k)\log\left(1-\left(g\left(z_k^{(L)}\right)\right)_k\right)\right] \\
\frac{\partial J}{\partial z_j^{(L)}} &= -\sum_{k=1}^{K}\left\{\frac{\partial}{\partial z_j^{(L)}}\left[y_k \log\left(g\left(z_k^{(L)}\right)\right)_k + (1-y_k)\log\left(1-\left(g\left(z_k^{(L)}\right)\right)_k\right)\right]\delta_{jk}\right\} \\
\delta_j^{(L)} &= -\frac{\partial}{\partial z_j^{(L)}}\left[y_j \log\left(g\left(z_j^{(L)}\right)\right)_j + (1-y_j)\log\left(1-\left(g\left(z_j^{(L)}\right)\right)_j\right)\right] \\
&\equiv -\frac{\partial}{\partial z_j^{(L)}}\left[y_j \log g_j + (1-y_j)\log(1-g_j)\right] \\
&= -y_j\frac{g_j'}{g_j} + (y_j-1)\frac{-g_j'}{1-g_j} \\
&= -y_j\frac{g_j*(1-g_j)}{g_j} + (y_j-1)\frac{-g_j*(1-g_j)}{1-g_j} \\
&= y_j*(g_j-1) + (y_j-1)*(-g_j) \\
&= -y_j + y_j g_j - y_j g_j + g_j \\
&= g_j - y_j \\
\delta_j^{(L)} &= a_j^{(L)} - y_j
\end{aligned}
$$

$$\boldsymbol{\delta^{(L)} = a^{(L)} - y}$$

The error of the final layer is simply the difference between the hypothesis and the correct value! That's easy enough to understand: if the final activation differs from the actual value by a certain amount, then the hypothesis is "wrong" by that amount.

For a more arbitrary cost function, we can generalize this result using the chain rule:

$$\delta_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}}$$

$$= \sum_{k=1}^{K_L} \frac{\partial J}{\partial a_k^{(L)}} \frac{\partial a_k^{(L)}}{\partial z_j^{(L)}}$$

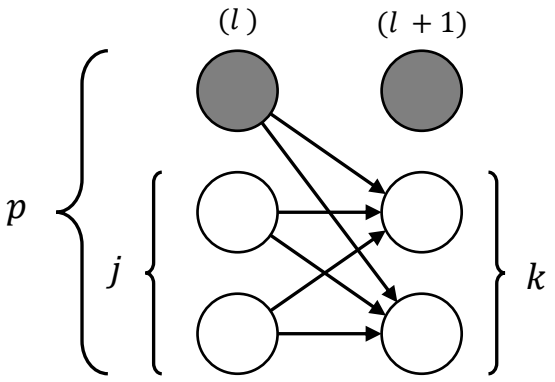
$$= \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \quad , \qquad a_j^{(L)} = g(z_j^{(L)})$$

$$\delta_j^{(L)} = \frac{\partial J}{\partial a_j^{(L)}} g'(z_j^{(L)})$$

$$\boxed{\boldsymbol{\delta^{(L)}} = \left\{\boldsymbol{\nabla_{a^{(L)}}} J\right\} \odot \boldsymbol{g'(z^{(L)})}}$$

Of course, for a given node, $z_j^{(l)}$ depends only on the activation of the same node $a_j^{(l)}$ in that layer (not any of the other nodes in the same layer), so the sum is multiplied by a Kronecker delta, leaving only the terms where $j = k$.

Finding the error for nodes in earlier layers of the network is more difficult because the cost only *implicitly* depends on the activations of those earlier-layer nodes (whereas it explicitly depends on the activations of the output layer). Since we already know the errors of the output layer, we'd somehow like to relate them to the errors of the hidden layers. We can do this by *propagating* the error from the output backward through the network layer by layer. Again, using the chain rule, we find an expression for this error below. **IMPORTANT NOTE***: $\left\{\boldsymbol{\Theta_{back}^{(l)}}: \Theta_{kj}^{(l)} \in \mathbb{R}^{k \times j}\right\} \neq \left\{\boldsymbol{\Theta^{(l)}}: \Theta_{kp}^{(l)} \in \mathbb{R}^{k \times p}\right\}$



*The math in this part gets tricky because bias nodes create a sort of inconsistency in the dimensionality of the forward propagating and backpropagating matrices. To make things easier to follow, I have drawn a schematic diagram above labeling the range spanned by each index referenced (bias nodes are shaded). Notice that "error" only applies to non-bias nodes. We can make sense of this by remembering that, like the training example data (which also have no associated "error"), bias node values are always fixed in value.

$$\delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}}$$

$$= \sum_{k=1}^{s_{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

$$= \sum_{k=1}^{s_{(l+1)}} \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

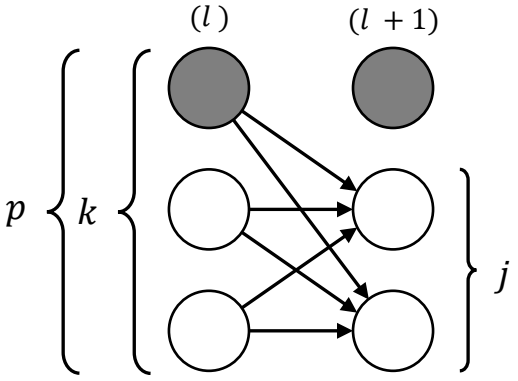
$$\left(z_k^{(l+1)} = \sum_{p=1}^{s_{(l)}+1} \Theta_{kp}^{(l)} a_p^{(l)} = \sum_{p=1}^{s_{(l)}+1} \Theta_{kp}^{(l)} * g\left(z_p^{(l)}\right)\right)$$

$$\delta_j^{(l)} = \sum_{k=1}^{s_{(l+1)}} \left\{\delta_k^{(l+1)} \frac{\partial}{\partial z_j^{(l)}} \left(\sum_{p=1}^{s_{(l)}+1} \Theta_{kp}^{(l)} * g\left(z_p^{(l)}\right)\right)\right\}$$

$$\delta_j^{(l)} = \sum_{k=1}^{s_{(l+1)}} \left\{\delta_k^{(l+1)} \Theta_{kj}^{(l)} * g'\left(z_j^{(l)}\right)\right\}$$

$$\boxed{\boldsymbol{\delta^{(l)}} = \left\{\left(\boldsymbol{\Theta_{back}^{(l)}}\right)^{\boldsymbol{T}} \boldsymbol{\delta^{(l+1)}}\right\} \odot \boldsymbol{g'\left(z^{(l)}\right)}}$$

With just one more application of the chain rule, we can relate these error values to the gradient of the cost with respect to the parameters like so:



$$\frac{\partial J}{\partial \Theta_{jk}^{(l)}} = \frac{\partial J}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial \Theta_{jk}^{(l)}}$$

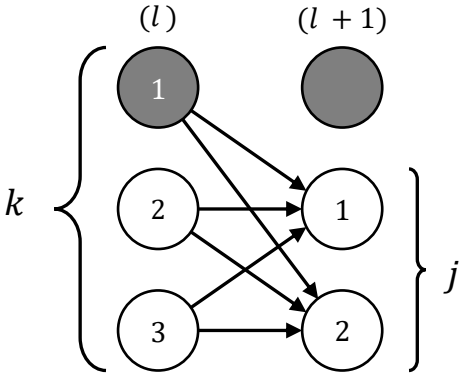
$$= \delta_j^{(l+1)} \frac{\partial}{\partial \Theta_{jk}} \left\{ \sum_{p=1}^{s_{(l)}+1} \Theta_{jp}^{(l)} a_p^{(l)} \right\}$$

$$= \delta_j^{(l+1)} \left\{ \sum_{p=1}^{s_{(l)}+1} \delta_{kp} a_p^{(l)} \right\}, \text{ where } \delta_{kp} \equiv \begin{cases} 1 \ , & k = p \\ 0 \ , & k \neq p \end{cases}$$

$$\frac{\partial J}{\partial \Theta_{jk}^{(l)}} = a_k^{(l)} \delta_j^{(l+1)}$$

$$\boxed{\frac{\partial J}{\partial \boldsymbol{\Theta}^{(l)}} = \boldsymbol{\delta}^{(l+1)} \left(\boldsymbol{a}^{(l)}\right)^{\mathrm{T}}}$$

At last, we have an expression for the difficult part of the J_{net} gradient! We can easily calculate the regularization contribution to the gradient of J_{net}, since the regularization term explicitly depends on just $\Theta_{pq}^{(l)}$ in each layer:



$$J_{\text{reg}} = \frac{\lambda}{2m} \sum_{n=1}^{L-1} \sum_{p=1}^{s_{(n+1)}} \sum_{q=2}^{s_{(n)}+1} \left\{ \left(\Theta_{pq}^{(n)}\right)^2 \right\}$$

$$\frac{\partial J_{\text{reg}}}{\partial \Theta_{jk}^{(l)}} = \frac{\lambda}{2m} \sum_{n=1}^{L-1} \sum_{p=1}^{s_{(n+1)}} \sum_{q=2}^{s_{(n)}+1} 2\Theta_{pq}^{(n)} \delta_{jp} \delta_{kq} \delta_{nl}$$

$$\frac{\partial J_{\text{reg}}}{\partial \Theta_{jk}^{(l)}} = \frac{\lambda}{m} \Theta_{jk}^{(l)} * \begin{cases} 1 \ , & k \neq 1 \\ 0 \ , & k = 1 \end{cases}$$

$$\boxed{\frac{\partial J_{\text{reg}}}{\partial \boldsymbol{\Theta}^{(l)}} = \frac{\lambda}{m} \boldsymbol{\Theta}^{(l)} \left[\mathbf{I} - \hat{\boldsymbol{e}}_{\mathbf{1}} \hat{\boldsymbol{e}}_{\mathbf{1}}^{\mathrm{T}}\right]}$$

Great! Now that we have all the pieces, we can put them together in an expression for the gradient of the total cost J_{net}, which averages over all m training examples and includes regularization:

$$\boxed{\frac{\partial J_{\text{net}}}{\partial \Theta_{jk}^{(l)}} = \frac{1}{m} \left\{ \sum_{i=1}^{m} \frac{\partial J(i)}{\partial \Theta_{jk}^{(l)}} \right\} + \frac{\partial J_{\text{reg}}}{\partial \Theta_{jk}^{(l)}}}$$

Implementing this requires that we first propagate the inputs forward through the network to calculate the activations at each node in each layer (for a fixed set of parameters), then backpropagate the error from the output nodes back to the first hidden layer. We then accumulate the results for each training example in a process elaborated above in the previous section.