Coursera - Week 5 - ML 10/ Third / 2021

ML: Neural Networks: Learning:

Cost function:

Let's first define a few variables that we will need to use:

a) L = total number of layers in the network

b) s_l = number of units (not counting bias units) in layer l

c) k = number of ~~outputs~~ output units / classes.

Recall that in neural networks, we may have many output nodes. We denote $h_\Theta(x)_k$ as being a hypothesis that results in the k^{th} output.

Our cost function for a neural network is going to be a generalization of the one we used for logistic regression.

Recall that the cost function for regularized logistic regression was:

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

For neural networks, it is going to be slightly more complicated:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log\left((h_\Theta(x^{(i)}))_k\right) + \left(1-y_k^{(i)}\right)\log\left(1-(h_\Theta(x^{(i)}))_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\Theta_{j,i}^{(l)}\right)^2$$

We have added a few nested summations to account for our multiple output nodes. In the first part of the equation, before the square brackets, we have an additional nested summation that loops through the number of output nodes.

In the regularization part, after the square brackets, we must account for multiple theta matrices. The number of columns in our current

Note matrix is equal to the number of nodes in our current layer (including the bias unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit). As before with logistic regression, we ~~add~~ square every term.

Note:

- The double sum simply adds up the logistic regression costs calculated for each cell in the output layer; and
- The triple sum simply adds up the squares of all the individual Θ_{ij} in the entire network.
- The i in the triple sum does <u>not</u> refer to training example i

Backpropogation Intuition:

The cost function is:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log\left(h_\Theta(x^{(i)})\right)_k + (1-y_k^{(i)})\log\left(1-h_\Theta(x^{(i)})_k\right)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\Theta_{j,i}^{(l)}\right)^2$$

If we consider simple non-multiclass classification ($k=1$) and disregard regularization, the cost is computed with:

$$cost(t) = y^{(t)}\log\left(h_\Theta(x^{(t)})\right) + (1-y^{(t)})\log\left(1-h_\Theta(x^{(t)})\right)$$

More intuitively you can think of this equation roughly as:

$$cost(t) \approx \left(h_\Theta(x^{(t)}) - y^{(t)}\right)^2$$

Intuitively, $\delta_j^{(l)}$ is the "error" for $a_j^{(l)}$ (unit j in layer l)

More formally, the delta values are actually the derivatives of the cost function:

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

Recall that our derivative is the slope of a line tangent to the cost function, so the steeper the slope the more incorrect we are.

Note: I before, sometimes i is used to index a training example. Sometimes it is used to index a unit in a layer. In the Back Propogation Algorithm described here, i is used to index a training example rather than overloading the use of i.

ckpropagation Algorithm:

"Backpropagation" is neural-network terminology for minimizing our cost function, just like what we were doing with gradient descent in logistic and linear regression. Our goal is to compute:

$\min_\Theta J(\Theta)$

That is, we want to minimize our cost function J using an optimal set of parameters in theta. In this section we'll look at the equations we use to compute the partial derivative of $J(\Theta)$:

$$\frac{\partial}{\partial \Theta^{(l)}_{i,j}} J(\Theta)$$

To do so, we use the following algorithm:

~~Intg~~ Imagi

back propagation Algorithm

Given training set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$

- Set $\Delta^{(l)}_{i,j} := 0$ for all (l,i,j), (hence you end up having a matrix full of zeros)

For training example $t = 1$ to m:

1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Gradient computation:

Given one training example (x, y):

Forward Propagation:

$a^{(1)} = x$

$z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

$z^{(3)} = \Theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = h(x) = g(z^{(4)})$

3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$

Where L is our total number of layers and $a^{(L)}$ is the vector of outputs of the activation units for the last layer. So our "error values" for the last layer are simply the differences of our actual results in the last layer and the correct output in y. To get the delta values of the layers before the last layer, we can use an equation that steps us back from right to left:

4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ using $\delta^{(l)} = \left(\left(\Theta^{(l)}\right)^T \delta^{(l+1)}\right) .* a^{(l)} .* \left(1 - a^{(l)}\right)$

The delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l. We then element-wise multiply that with a function called g', or g-prime, which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$.

The g-prime derivative terms can also be written out as:

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

5. $\Delta^{(l)}_{i,j} := \Delta^{(l)}_{i,j} + a^{(l)}_j \delta^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T$

Hence we update our new Δ matrix:

- $D^{(l)}_{i,j} := \frac{1}{m}\left(\Delta^{(l)}_{i,j} + \lambda \Theta^{(l)}_{i,j}\right)$, if $j \neq 0$.
- $D^{(l)}_{i,j} := \frac{1}{m} \Delta^{(l)}_{i,j}$ if $j = 0$

The capital-delta matrix D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \Theta^{(l)}_{i,j}} J(\Theta) = D^{(l)}_{i,j}$