Untitled

October 7, 2019

1 Backprop. Important Notes - Andrew Ng's Course

2.4 Backpropagation

the intuition behind the backpropagation algorithm is as follows. Given a training example $(x^{(t)}, y^{(t)})$, we will first run a "forward pass" to compute all the activations throughout the network, including the output value of the hypothesis $h_\theta(x)$. Then, for each node j in layer l, we would like to compute an "error term" $\delta_j^{(l)}$ that measures how much that node was "responsible" for any errors in our output.

For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to define $\delta_j^{(3)}$ (since layer 3 is the output layer). For the hidden units, you will compute $\delta_j^{(l)}$ based on a weighted average of the error terms of the nodes in layer $(l+1)$. In detail, here is the backpropagation algorithm (also depicted in the figure above). You should implement steps 1 to 4 in a loop that processes one example at a time. Concretely, you should implement a for-loop `for t in range(m)` and place steps 1-4 below inside the for-loop, with the t^{th} iteration performing the calculation on the t^{th} training example $(x^{(t)}, y^{(t)})$. Step 5 will divide the accumulated gradients by m to obtain the gradients for the neural network cost function.

1. Set the input layer's values $(a^{(1)})$ to the t^{th}training example $x^{(t)}$. Perform a feedforward pass, computing the activations $(z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)})$ for layers 2 and 3. Note that you need to add a `+1` term to ensure that the vectors of activations for layers $a^{(1)}$ and $a^{(2)}$ also include the bias unit. In `numpy`, if a 1 is a column matrix, adding one corresponds to `a_1 = np.concatenate([np.ones((m, 1)), a_1], axis=1)`.

2. For each output unit k in layer 3 (the output layer), set

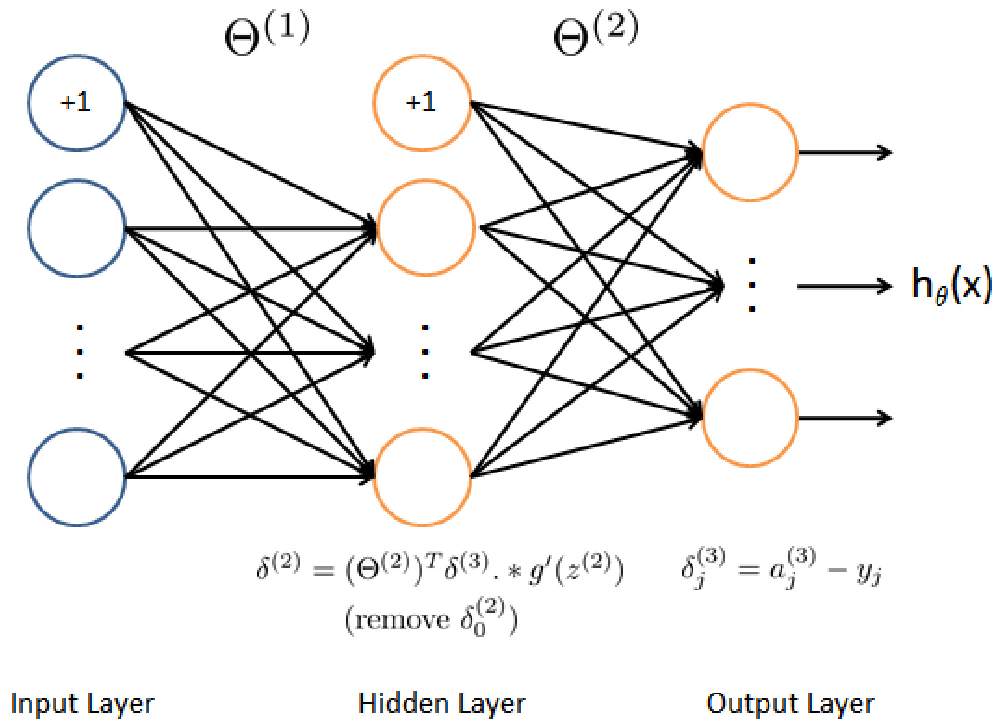
$$\delta_k^{(3)} = \left(a_k^{(3)} - y_k\right)$$

   where $y_k \in \{0, 1\}$ indicates whether the current training example belongs to class k $(y_k = 1)$, or if it belongs to a different class $(y_k = 0)$. You may find logical arrays helpful for this task (explained in the previous programming exercise).

3. For the hidden layer $l = 2$, set

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} * g'\left(z^{(2)}\right)$$

   Note that the symbol $*$ performs element wise multiplication in `numpy`.



4. Accumulate the gradient from this example using the following formula. Note that you should skip or remove $\delta_0^{(2)}$. In `numpy`, removing $\delta_0^{(2)}$ corresponds to `delta_2 = delta_2[1:]`.

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^{(T)}$$

5. Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $\frac{1}{m}$:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

Python/Numpy tip: You should implement the backpropagation algorithm only after you have successfully completed the feedforward and cost functions. While implementing the backpropagation alogrithm, it is often useful to use the `shape` function to print out the shapes of the variables you are working with if you run into dimension mismatch errors.

Note: If the iterative solution provided above is proving to be difficult to implement, try implementing the vectorized approach which is easier to implement in the opinion of the moderators of this course. You can find the tutorial for the vectorized approach here.

```
In [ ]:
```