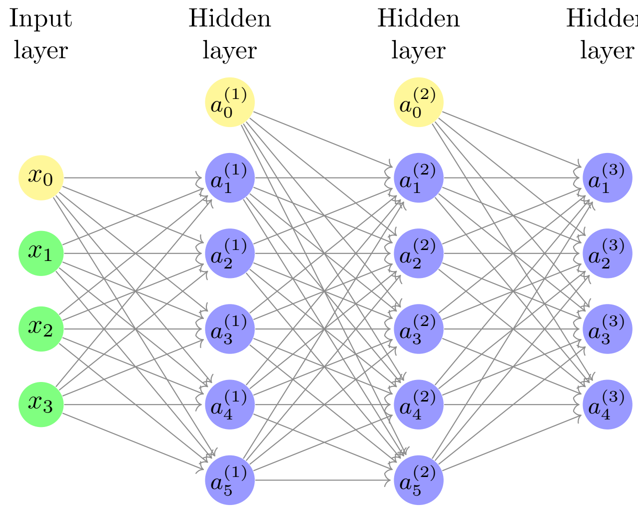
Week 9

July 17, 2021

Types of classification problems with NNs

- Training set is $(x^1, y^1), (x^2, y^2), ..., (x^n, y^n)$.
- L = number of layers in the network.
- s_l = number of units (not counting bias unit) in layer l.



So here we have:

- $L = 4$.
- $s_1 = 3$.
- $s_2 = 5$.
- $s_3 = 5$.
- $s_4 = 4$.

Binary classification

- 1 output (0 or 1).
- So single output node - value is going to be a real number.
- $k = 1$.
- $s_l = 1$

Multi-class classification

- k distinct classifications.
- y is a k-dimensional vector of real numbers.
- $s_l = k$

Cost function

Logistic regression cost function is as follows:

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

For neural networks our cost function is a generalization of this equation above, so instead of one output we generate k outputs:

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

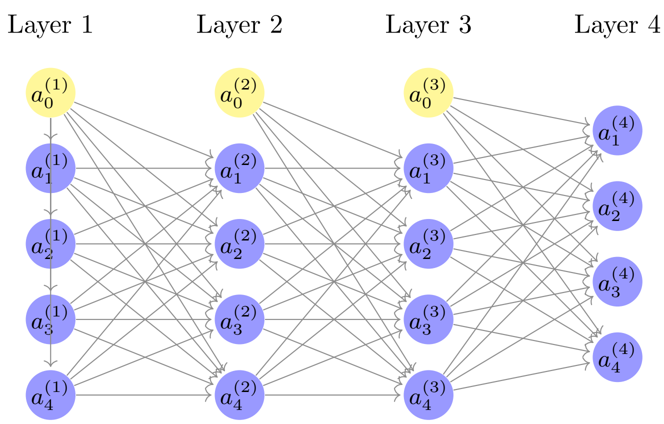
- Our cost function now outputs a k dimensional vector.
- Costfunction $J(\Theta)$ is $-1/m$ times a sum of a similar term to which we had for logic regression.
- But now this is also a sum from $k = 1$ through to K (K is number of output nodes).
- Summation is a sum over the k output units - i.e. for each of the possible classes.
- We don't sum over the bias terms (hence starting at 1 for the summation).

Partial derivative terms

- $\Theta^{(1)}$ is the matrix of weights which define the function mapping from layer 1 to layer 2.
- $\Theta_{10}^{(1)}$ is the real number parameter which you multiply the bias unit (i.e. 1) with for the bias unit input into the first unit in the second layer.
- $\Theta_{11}^{(1)}$ is the real number parameter which you multiply the first (real) unit with for the first input into the first unit in the second layer.
- $\Theta_{21}^{(1)}$ is the real number parameter which you multiply the first (real) unit with for the first input into the second unit in the second layer.

Gradient computation

- One training example.
- Imagine we just have a single pair (x,y) - entire training set.
- The following is how the forward propagation method works:
- Layer 1: $a^{(1)} = x$ and $z^{(2)} = \Theta^{(1)} a^{(1)}$.
- Layer 2: $a^{(2)} = g(z^{(2)}) + a_0^{(2)}$ and $z^{(3)} = \Theta^{(2)} a^{(2)}$.
- Layer 3: $a^{(3)} = g(z^{(3)}) + a_0^{(3)}$ and $z^{(4)} = \Theta^{(3)} a^{(3)}$.
- Ouptut: $a^{(4)} = h_\Theta(x) = g(z^{(4)})$.



Calculate backpropagation

- δ_j is $Lx1$ vector.
- First we compute the LAST element: $\delta_j^{(L)} = a_j^{(L)} - y_j$.
- Value of each element is based on the value of the next element:

$$\delta_j^{(l)} = (\Theta_j^{(l)})^T \delta_j^{(l+1)} \cdot g'(z_j^{(l)})$$

- Finally, use Δ to accumulate the partial derivative terms:

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

- l = layer.
- j = node in that layer.
- i = the error of the affected node in the target layer

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \begin{cases} \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{m}\Delta_{ij}^{(l)} & \text{if } j = 0 \end{cases}$$

Gradient checking

Backpropagation contains a lot of details, and tiny flaws can break it. As a result, employing a numerical approach to verify the gradient can aid in the quick diagnosis of a problem.

- Have a function $J(\Theta)$.
- Compute $\Theta + \epsilon$.
- Compute $\Theta - \epsilon$.
- Join them by a straight line.
- Use the slope of that line as an approximation to the derivative.