Week 5

Neural Network : Back propagation

Neural Network (classification

L = total no. of layers in network

S_l = no. of units (not counting bias unit) in layer l

Binary classification

$y = 0$ or 1

1 output unit

$S_L = 1$

output layer

Multi-class classification (K classes)

$y \in \mathbb{R}^K$

K output units

$h_\theta(x) \in \mathbb{R}^K$

$S_L = K \quad (K \geq 3)$

out layer

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

for neural network

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}} (\theta_{ji}^{(l)})^2$$

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

Note:

- the double sum simply adds up the logistic regression costs calculated for each cell in the output layer.
- the triple sum simply adds up the squares of all the individuals θs in the entire network.
- the i in the triple sum does not refer to training example i.

$\Theta_{ij}^{(l)} \in \mathbb{R}$

$-J(\theta)$

$-\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

Gradient Computation

Given one training example (x, y):

Forward propagation:

$a^{(1)} = x$

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

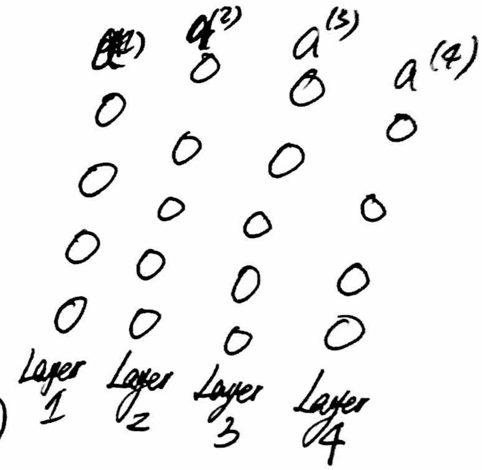
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

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

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

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

$a^4 = h_\theta(x) = g(z^{(4)})$



Backpropagation algorithm

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l.

For each output unit (layer $L = 4$)

$\delta_j^{(4)} = a_j^{(4)} - y_j$

$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$

$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$

No $\delta^{(1)}$

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)}$$

Backpropagation algorithm.

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

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j) (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

— For $t = 1$ to m $(x^{(t)}, y^{(t)})$

Set $a^{(1)} = x^{(t)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \cdots, L$

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

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

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

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

包括了 bias unit 的 Gradient Decent 值

— $D_{ij}^{(l)} := \frac{1}{m}(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)})$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

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

$$J(\theta) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$ 求导 Neural Network怎么出来的.

$$\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}} \quad ③$$

$$\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}} \quad ④$$

$$\delta^{(L)} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \quad ①$$

$$\delta^{(L-1)} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \quad ②$$

plug ① into ②:

$$\delta^{(L-1)} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \quad ⑤$$

use ① ⑤ back into ③.④

$$\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}}$$

$$\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \delta^{(L-1)} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}}$$

$$\boxed{\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}}}$$

$$\delta^{(L)} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$$

where $a^{(L)} = h_\theta(x)$

$$J(\theta) = -y\log(a^{(L)}) - (1-y)\log(1-a^{(L)})$$

$$\frac{\partial J(\theta)}{\partial a^{(L)}} = \frac{1-y}{1-a^{(L)}} - \frac{y}{a^{(L)}}$$

$$a = g(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = a^L(1-a^L)$$

$$\delta^{(L)} = \left(\frac{1-y}{1-a^{(L)}} - \frac{y}{a^{(L)}}\right) a^L(1-a^L)$$

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

$$\text{So } \frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}} = (a^L - y)\, a^{(L-1)}$$

$$\boxed{\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \delta^{(L-1)} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}}}$$

由⑤, $\delta^{(L-1)} = (a^L - y)\,\theta^{(L-1)} a^{(L-1)} (1-a^{(L-1)})$

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = \theta^{(L-1)}$$

$$\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} = a^{(L-1)}(1-a^{(L-1)})$$

$$\text{So } \frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \delta^{(L-1)} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}}$$

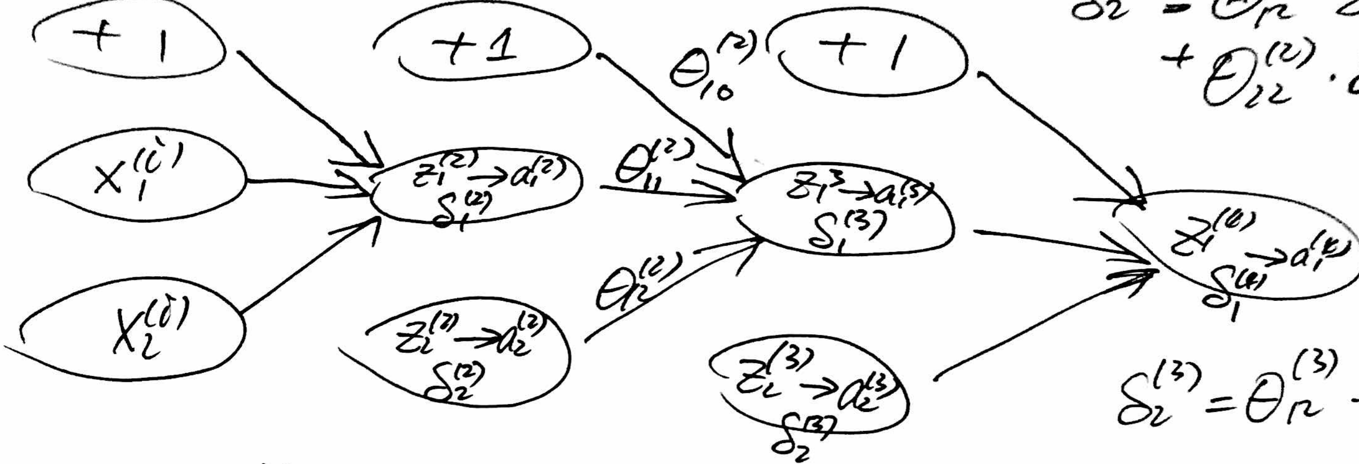
$$= (a^L - y)\,\theta^{(L-1)} a^{(L-1)} (1-a^{(L-1)})\, a^{(L-2)}$$

QED

Intuition: Forward

$\delta_1^{(4)} = \cancel{y^{(i)}}\, a_1^{(4)} - y$

$\delta_2^{(2)} = \Theta_{12}^{(2)} \cdot \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$



$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$

$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} \cdot a_2^{(2)}$$

back~~ward~~ propagation

Focusing on a single example $x^{(i)}, y^{(i)}$, the case of 1 output unit, and ingnoring regularization ($\lambda = 0$),

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

Think of $\left(cost(i) \approx (h_\theta(x^{(i)}) - y^{(i)})^2\right)$

i.e. how well is the network doing on example i?

$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).

[Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$ (for $j \geq 0$), where

$cost(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log h_\theta(x^{(i)})$]

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

Unrolling Parameters: Learning Algorithm

Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$

Unroll to get initialTheta to pass to

fminunc (@costFunction, initialTheta, options)

function [jval, gradientVec] = costFunction (thetaVec)

From thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$

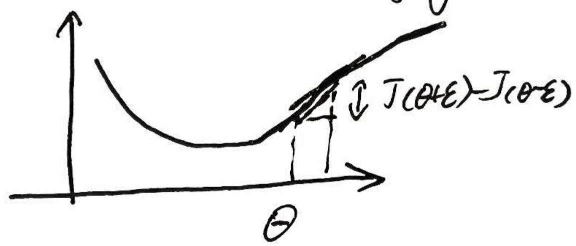
Use forward prop / back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$

unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec.

Gradient checking:

Numerical estimation of gradients

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$



$\theta \in \mathbb{R}$

$\varepsilon = 10^{-4}$

with multiple theta matrices,

$$\frac{\partial}{\partial \theta_j} J(\theta) \approx \frac{J(\theta_1, \cdots, \theta_j + \varepsilon, \cdots, \theta_n) - J(\theta_1, \cdots \theta_j - \varepsilon, \cdots \theta_n)}{2\varepsilon}$$

(suggest $\varepsilon = 10^{-4}$)

```
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) = thetaPlus(i) + EPSILON;
  thetaMinus = theta;
  thetaMinus(i) = thetaMinus(i) - EPSILON;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                   / (2 * EPSILON);
end;
```

Check that $\boxed{\text{gradApprox}} \approx \boxed{\text{DVec}}$

From backprop

Implementation Note:

- Implement backprop to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$)
- Implement numerical gradient check to compute gradApprox.
- Make sure they give similar values
- Turn off gradient checking, Using back prop code for learning

Important:

- Be sure to disable your gradient checking code before training you classifier, If you run numerical gradient computation on every iteration of gradient decent (or in the inner loop of costFunction(…)) you code will be very slow.

Initial value of Θ

For gradient descent and advanced optimization method, need initial value for Θ.

Opt Theta = fminunc(@costFunction, initialTheta, options)

Consider gradient desent

Set initialTheta = zeros (n,1)? if $\Theta_{ij}^{(l)} = 0$ for all i, j, l.

zero initialization:

$$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta) \qquad \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$$

After each update, parameters corresponding to inputs going into each of two hidden units are identical.

Random Initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$

(i.e. $-\epsilon \le \Theta_{ij}^{(l)} \le \epsilon$)

Eg.

Theta1 = rand(10, 11) * (2 * INIT_EPSILON) - INIT_EPSILON

Theta2 = rand (1, 11) * (2 * INIT_EPSILON) - INIT_EPSILON

Training a neural network

Pick a network architecture (connectivity pattern between neurons)

i h o	i h h o	i h h h o
3 5 4	3 5 5 4	3 5 5 5 4

No. of input units: Dimension of features $x^{(i)}$

No. output units. Number of classes

No. of hidden units per layer

Reasonable defaul: 1 hidden layer, or if > 1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

Steps to train a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

   ~~for~~ for i = 1:m
   Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
   (Get activation $a^{(l)}$ and delta term $\delta^{(l)}$ for $l = 2, \dots, L$)
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$

   Then disable gradient checking code.
6. use gradient decent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ