ML COURSERA: WEEK 5

LEO ROSINOVITCH

NEURAL NETWORK COST FUNCTION:

- m # training examples
- L # layers in network (including I/O layers)
- S_l = # units in layer l (not counting bias unit)

=> Binary $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ or $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ vs. Multi-Class eg. $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$ etc. (k-classes)

↳ $S_L = k$ ($=1$ for binary)

=> Recall Cost Fnt for Logistic Regression

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

=> Neural Network:

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

where $h_\theta(x) \in \mathbb{R}^K \rightarrow (h_\theta(x))_i = i^{th}$ output, $S_L = K$

i here NOT i^{th} training example!

Sum Θ^2 across all i, j, l vals, excluding bias terms

Must minimize -> need gradient (partial derivatives) $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\Theta)$

ex] Just one training example (x, y): (4 layers, $L=4$)

1] 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 $a_0^{(3)}$

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

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

a -> "activations"

2] Back Propagation:

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

⇒ for each output unit ($L=4$) → $\delta_j^{(4)} = a_j^{(4)} - y_j$

$a_j^{(4)} * (h_\theta(x))_j$

↳ vectorized: $\delta^{(4)} = a^{(4)} - y$

↳ $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* \underbrace{g'(z^{(3)})}_{a^{(3)} .* (1 - a^{(3)})}$

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

NO $\delta^{(1)}$ (no error in features of training set!)

⇒ Possible to prove $\underbrace{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}}$ (ignoring λ for now)

Back Prop
⇒ Algorithm: Set $\Delta_{ij}^{(l)} = 0$ for all l, i, j (use to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

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

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

Use forward prop. to compute $a^{(l)}$ for $l = 2, 3, \ldots L$

Use $y^{(i)}$ to compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ ⇒ $\underbrace{\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T}$ VECTORIZED

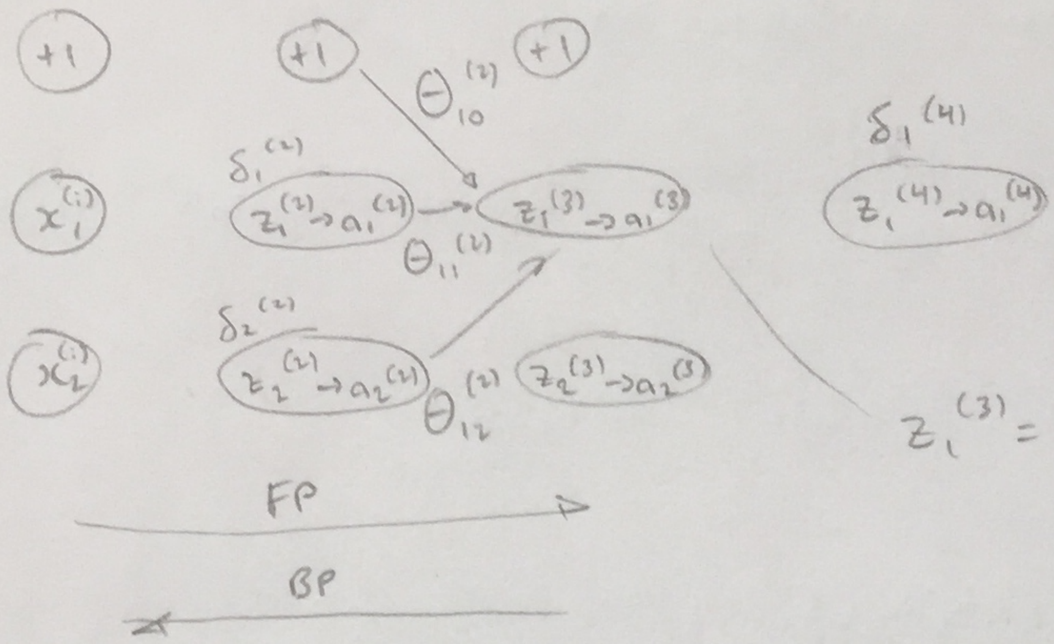
$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$

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

↳ D terms = partial derivatives!

=> Back Propagation not easy to understand!

=> Forward Prop example:



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

=> Single example $(x^{(i)}, y^{(i)})$, $\lambda = 0$

↳ $cost(i) = y^{(i)} \log[h_\theta(x^{(i)})] + (1 - y^{(i)}) \log[h_\theta(x^{(i)})]$ $\quad (\approx [h_\theta(x^{(i)}) - y^{(i)}]^2)$

↳ "how well is network doing on example i?"

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

↳ $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$ for $j \geq 0$

=> $\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$ -> set $\delta_1^{(3)}$, $\delta_1^{(2)}$, $\delta_2^{(3)}$, $\delta_2^{(2)}$

↳ $\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)}$ ← WEIGHTED SUM!

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

Back Prog Implementation:

```
function [jval, gradient] = costFunction(theta)

optTheta = fminunc(@costFunction, initialTheta, options)
```

} OCTAVE

⇒ for N.N. of L=4 → $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$; $D^{(1)}, D^{(2)}, D^{(3)}$ } Matrices → "unroll" into vectors

↳ `thetaVec = [Theta1(:); Theta2(:); Theta3(:)];`

`DVec = same ↑`

Then: `Theta1 = reshape(thetaVec(1:110), 10, 11);`

`Theta2 = " " "(111:220) " "`

`Theta3 = " " "(221:231), 1, 11);`

⇒ unroll Θ's to get initialTheta for fminunc!

⇒ in cost function → ① reshape thetaVec to get $\Theta^{(l)}$'s

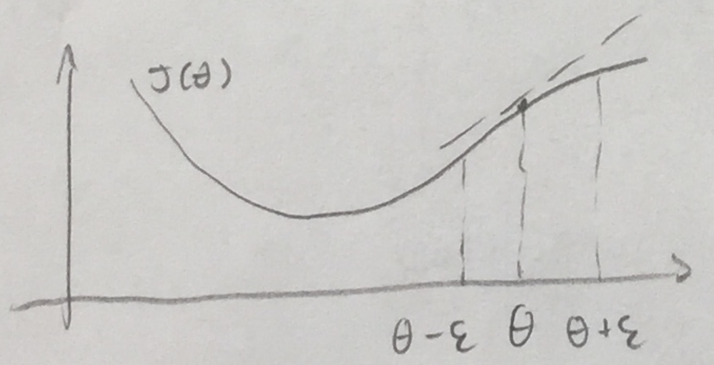
② use FP & BP to get $D^{(l)}$'s & $J(\Theta)$

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

Gradient Checking:

↳ Not necessarily good if $J(\Theta)$ is decreasing → makes sure everything O.K.

ex)



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

use $\varepsilon \simeq 10^{-4}$

implementation: `gradApprox = (J(theta + epsilon) - J(theta - epsilon)) / (2 * epsilon);`

$\theta \in \mathbb{R}^n \quad \Theta = [\theta_1, \theta_2 \dots \theta_n]$

$$\frac{\partial}{\partial \theta_1} J(\theta) \simeq \frac{J(\theta_1+\varepsilon, \theta_2, \dots \theta_n) - J(\theta_1-\varepsilon, \theta_2 \dots \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \simeq \frac{J(\theta_1, \theta_2 \dots \theta_n+\varepsilon) - J(\theta_1, \theta_2 \dots \theta_n-\varepsilon)}{2\varepsilon}$$

implementation: for i = 1:n

thetaPlus = theta;

thetaPlus(i) = thetaPlus(i) + ε;

thetaMinus = theta;

thetaMinus(i) = thetaMinus(i) - ε;

gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2ε);

end;

⇒ check if gradApprox ≃ DVec ! (up to small values)

⇒ gradient checking is just to make sure BP implentation is correct

↳ turn off gradient check after confirmation! (otherwise slow)

Random Initialization:

How to choose initialTheta? → used all zeros for multiclass regression...

⇒ Can't use all 0's for N.N.! → set $a_1^{(2)} = a_2^{(2)}$, $\delta_1^{(2)} = \delta_2^{(2)}$, $\Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

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

↳ Theta1 = rand(10,11) * (2*ε) - ε;

b/n 0 & 1

Theta2 = rand(1,11) * (2*ε) - ε;

⇒ each $\Theta_{ij}^{(l)}$ must be random!

(break symmetry)

Putting it All Together:

① Pick N.N. Architecture (# layers, # units in each layer)

=> # input units = # of features ($x^{(i)}$)

=> # output units = # of classes (K)

=> reasonable default -> 1 hidden layer

↳ if >1 hidden layer, same # hidden units in every layer

=> more hidden units usually better, but ↑comp. $

② Randomly initialize weights

③ FP to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$

④ Compute cost function $J(\Theta)$

⑤ BP to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

for i = 1:m (each $(x^{(i)}, y^{(i)})$)

FP & BP -> get $a^{(l)}$ & $\delta^{(l)}$ for $l = 2 \dots L$

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

...

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

end

⑥ Use gradient checking to compare P.D. terms from BP & numerical solns.

↳ then, disable gradient checking

⑦ Use gradient descent, fminunc, etc. w/ BP to get minimum of $J(\Theta)$ w/ optimal Θ params.

* Note: $J(\Theta)$ not convex -> could get stuck @ local min

↳ still pretty good.