Week 5

Notation

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

L - total no. of layers in network

S_l - no. of units (not counting bias unit) in layer L

Lets consider 2 classification cases:

- Binary classification.

1 output unit
$y = 0$ or 1 $\qquad h_\Theta(x) \in \mathbb{R}$
$S_L = 1$ (only one unit in output layer)
as this will be written as $K=1$

- Multi-class classification (K classes)

K output units
$y \in \mathbb{R}^K$ Eg. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$
pedestrian car motorcycle truck
$h_\Theta(x) \in \mathbb{R}^K$
$S_L = K$ where $K \geq 3$

Cost Function

Logistic regression

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

since θ_0 not considered

Neural network (for K terms)

$h_\Theta(x) \in \mathbb{R}^K \qquad (h_\Theta(x))_i = i\text{th output}$ ⟸ it selects the ith output

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

Scanned with CamScanner

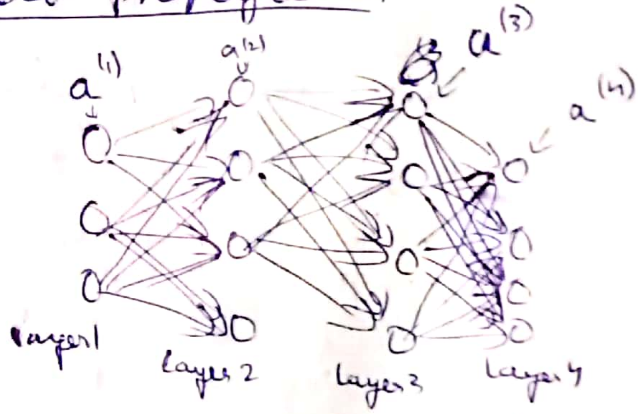
Gradient Computation

The goal is to $\min_{\Theta} J(\Theta)$

For gradient descent or any other advanced optimization algorithms, we need,

- $J(\Theta)$ ← we can use the formula to find this
- $\frac{\partial}{\partial \Theta^{(l)}_{ij}} J(\Theta)$ ← we will learn how to find this

Forward propogation



Given one training example (x, y)

$a^{(1)} = x$

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

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

this is the bias term

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

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

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

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

Scanned with CamScanner

Backpropagation Algorithm

$\delta_j^{(l)}$ - "error" of node j in layer l

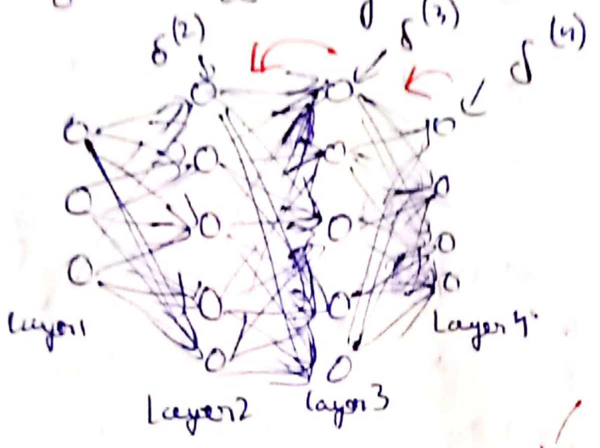
For each output unit (layer $L=4$)

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

This is nothing but prediction $(h_\theta(x))_j$

In vectorized form,

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



differentiation of $g(z^{(3)})$

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

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}) \rightarrow a^{(2)} .* (1-a^{(2)})$$

We find that (through calculus),

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0)$$

Note: We don't find $\delta^{(1)}$ since it is the input layer x

Scanned with CamScanner

The steps for Backpropogation Algorithm

Consider a training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

This is the accumulation factor that'll be used to find $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

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

→ Set $a^{(1)} = x^{(i)}$

→ Perform forward propogation to compute $a^{(l)}$ for $l = 2, 3, \ldots L$

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

→ Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ using backpropogation algorithm

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

← In matrix form $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

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

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

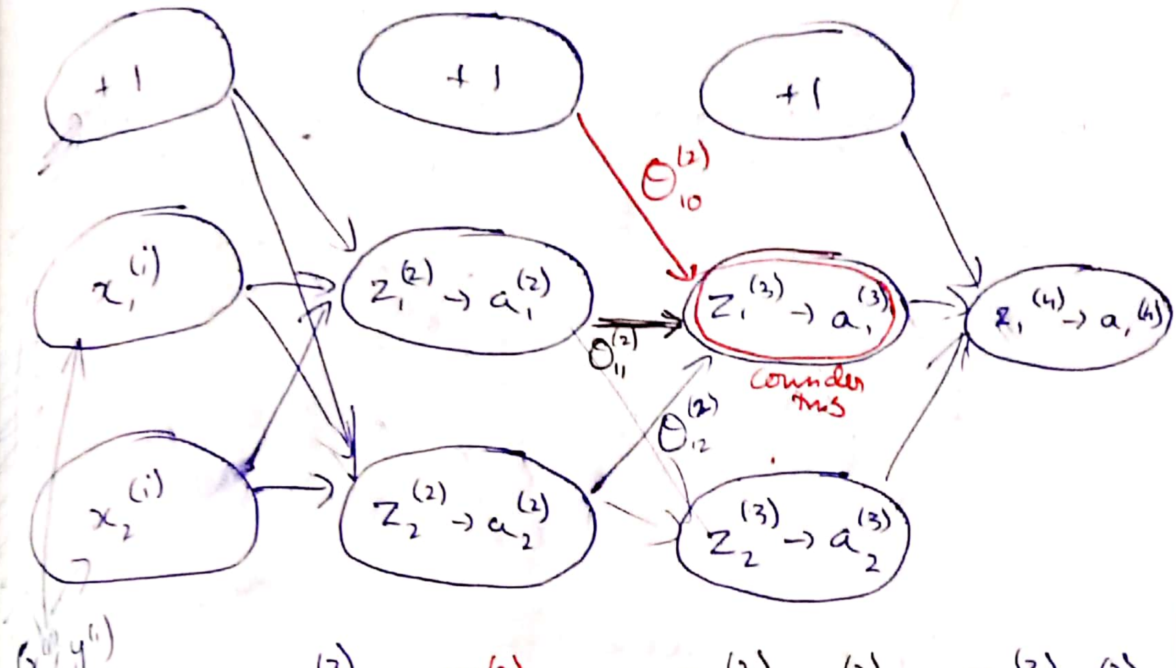
~~Accumulator~~ Gradient terms accumulator

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

Note: Backpropogation is neural-network terminology for minimizing our cost function

Scanned with CamScanner

Intuition of backpropagation



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

Backpropagation is kind of opposite of this

consider cost function,

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

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

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

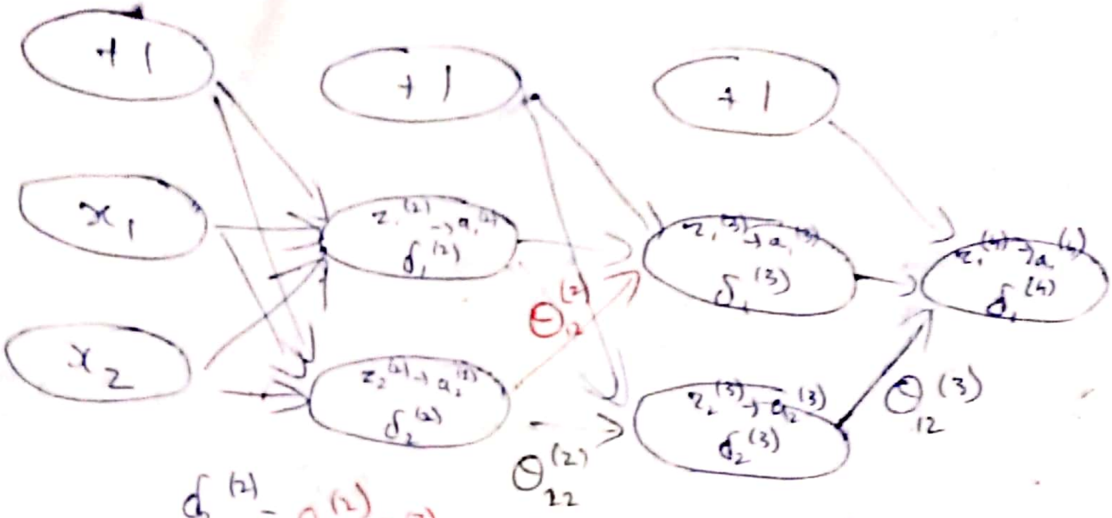
$$\text{cost}(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log h_\theta(x^{(i)})$$

We can think of this as $\text{cost}(i) \approx (h_\theta(x^{(i)}) - y^{(i)})^2$

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

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

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



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)} \qquad \delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$$

Unrolling parameters

Taking matrices and turning them into vector $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ $B = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ \vdots \\ 2 \end{bmatrix}$

~~and vice versa~~

Example

$s_1 = 10, \quad s_2 = 10, \quad s_3 = 1$

$\Theta^{(1)} \in \mathbb{R}^{10\times 11}, \quad \Theta^{(2)} \in \mathbb{R}^{10\times 11}, \quad \Theta^{(3)} \in \mathbb{R}^{1\times 11}$

$D^{(1)} \in \mathbb{R}^{10\times 11}, \quad D^{(2)} \in \mathbb{R}^{10\times 11}, \quad D^{(3)} \in \mathbb{R}^{1\times 11}$

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];
Theta1 = reshape(thetaVec(1:110), 10, 11);
Theta2 = reshape(thetaVec(111:220), 10, 11);
Theta3 = reshape(thetaVec(221:231), 1, 11);
```

Scanned with CamScanner

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

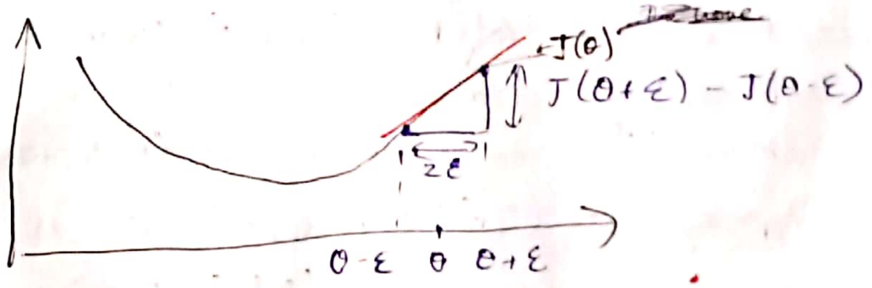
Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ & $J(\Theta)$

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

Gradient Checking

To make sure forward prop/ back prop is working properly.

Numerical estimation of gradients



We need to find the derivate to get the slope of the line. We use the above method to approximate the derivative..

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

Use a small ~~epsilo~~ ε (epsilon) to get a more accurate derivative approximation (around 10^{-4})

Parameter vector θ

$\theta \in \mathbb{R}^n$ (θ is "unrolled" version of $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$)

$\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \dots, \theta_n)}{2\epsilon}$$

$\vdots$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n+\epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n-\epsilon)}{2\epsilon}$$

On octave,

```
epsilon = 1e-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;
```

from backprop

We need to check if gradApprox $\approx$ DVec

Implementation:

- → Implement backprop to compute DVec
- → Implement numerical gradient check to get gradApprox
- → Make sure they give similar values
- → Turn off gradient checking. Using backprop for learning.

Note: Be sure to turn off gradient checking code after verifying if the NN is working properly, since it makes the code run slowly, when learning

Scanned with CamScanner

Random initialization

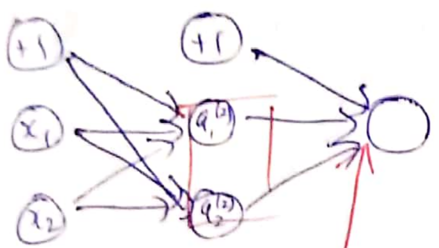
For gradient descent and advanced optimization methods, we need to set an initial value of Θ

optTheta = fminunc(@costFunction, initialTheta, options)

Consider gradient descent, can we set initialTheta = zeros(n, 1)? — All zeros

No. This won't work in neural ntwork.

Zero initialization



$\Rightarrow \Theta_{ij}^{(l)} = 0$ for all i, j, l

$a_1^{(2)} = a_2^{(2)}$ & $\delta_1^{(2)} = \delta_2^{(2)}$

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

So even after each update, $a_1^{(2)} = a_2^{(2)}$ will remain, and the process repeats

So the neural network only sees one value and doesn't learn

So we uses random initialisation to perform symmetry breaking

Random initialisation:

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

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

E.g.

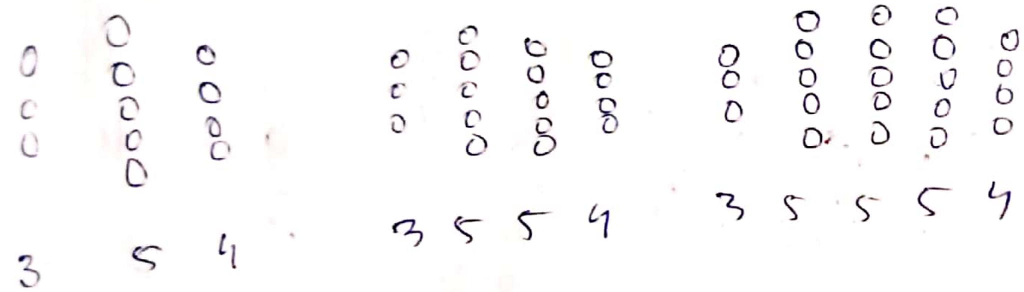
Random 10x11 matrix with value btwn 0 & 1

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

← You'll get a value between $-\epsilon$ & ϵ

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

Picking a Neural Network Architecture



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

→ No. of output units: Number of classes

Make sure it is like $y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \ldots$

→ Reasonable default is 1 ~~hidden~~ hidden layer

→ Or if no. of hidden layers >1, have same no. of hidden units in every layer (usually the more units, the better).

training a neural network

1. Randomly initialize weights
2. Implement forward propogation 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

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

Scanned with CamScanner

for $i = 1:m$ {

Perform forward prop & back prop using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ & delta terms $\delta^{(l)}$ for $l=2,\ldots L$)

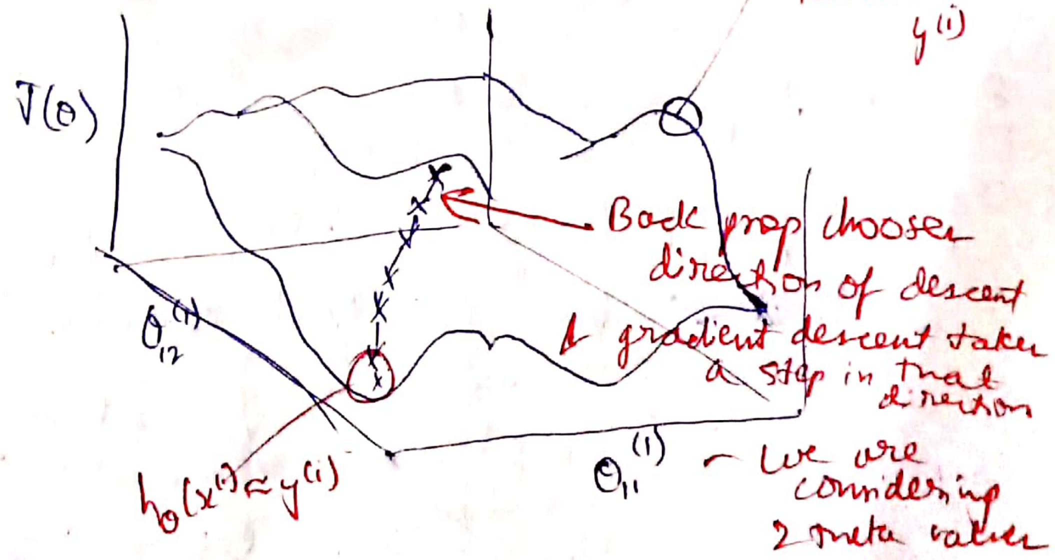
$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

}

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

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using back prop vs. using numeral estimate of gradient of $J(\Theta)$
   Then disable gradient checking code

6. Use gradient descent or advanced optimization method with back prop to try to minimize $J(\Theta)$ as a function of parameters Θ



Scanned with CamScanner