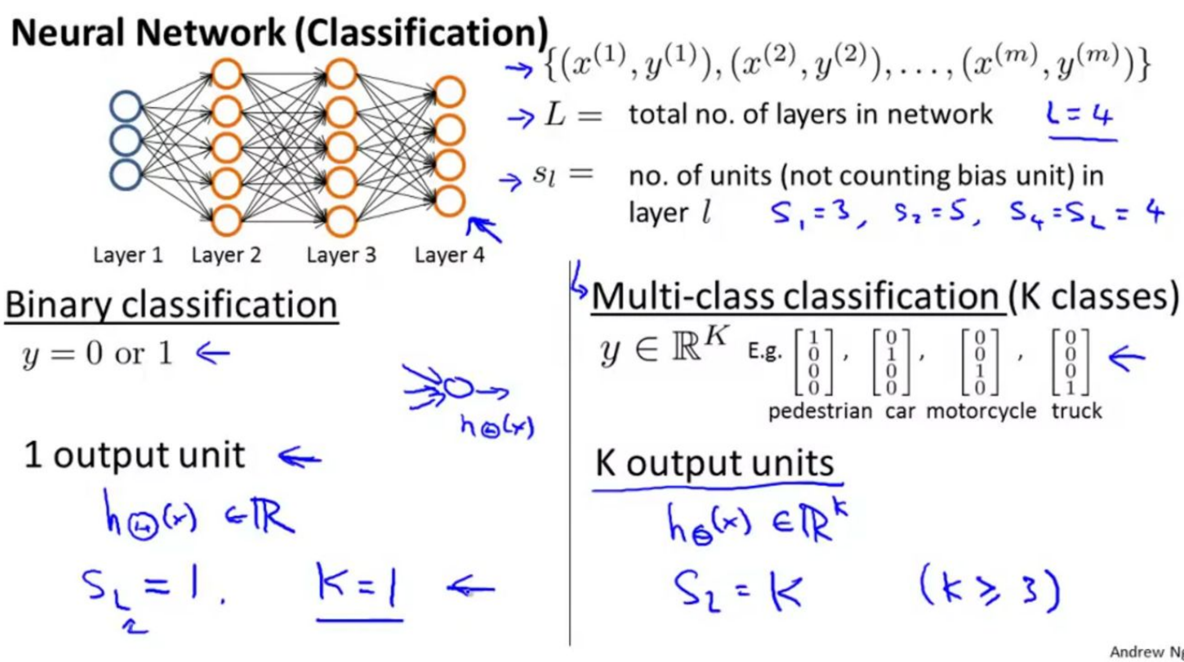
Week : 5

Neural network classification :

There are two types of **neural network**. One is Binary and the other one is Multi-class classification.

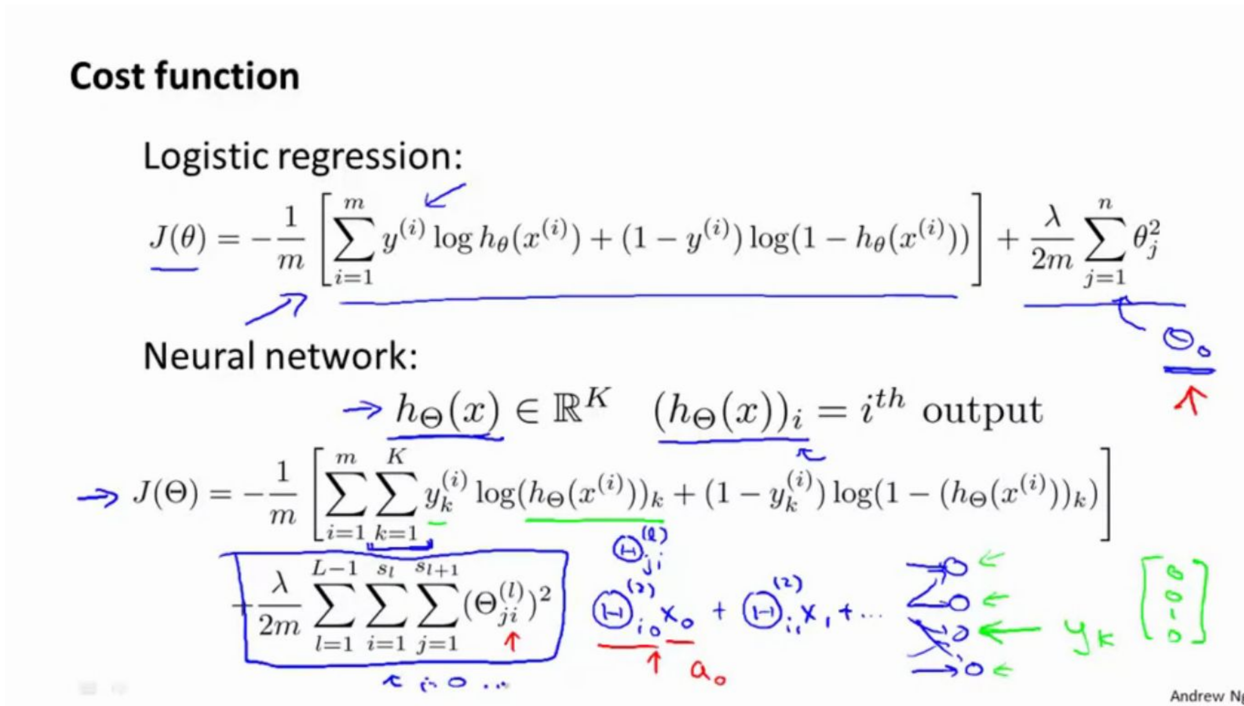
Binary Classification : If the number of output classes is less 3 then it is binary classification. In that case the number of output units is just one.

Multi Class Classification : If there number of output classes is more than 2 then it is Multi class classification. In that case the number of output units is equal to output classes. We need to apply one versus all method to get output.



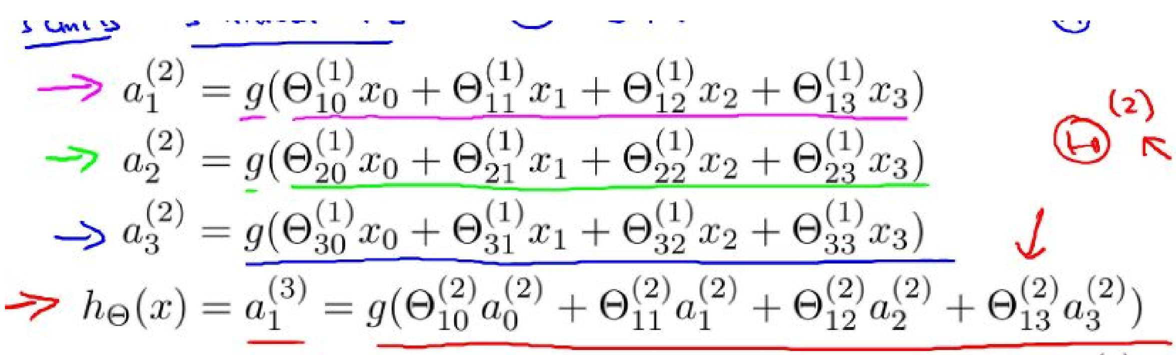
Cost Function :

Cost function needed to be calculated to visualize how much bugg free our algo is.



In the last part of the cost function I had a confusion why they compute summation of the $\Theta_{11}\ \Theta_{12}\ \Theta_{13}\ \Theta_{14} \cdots$

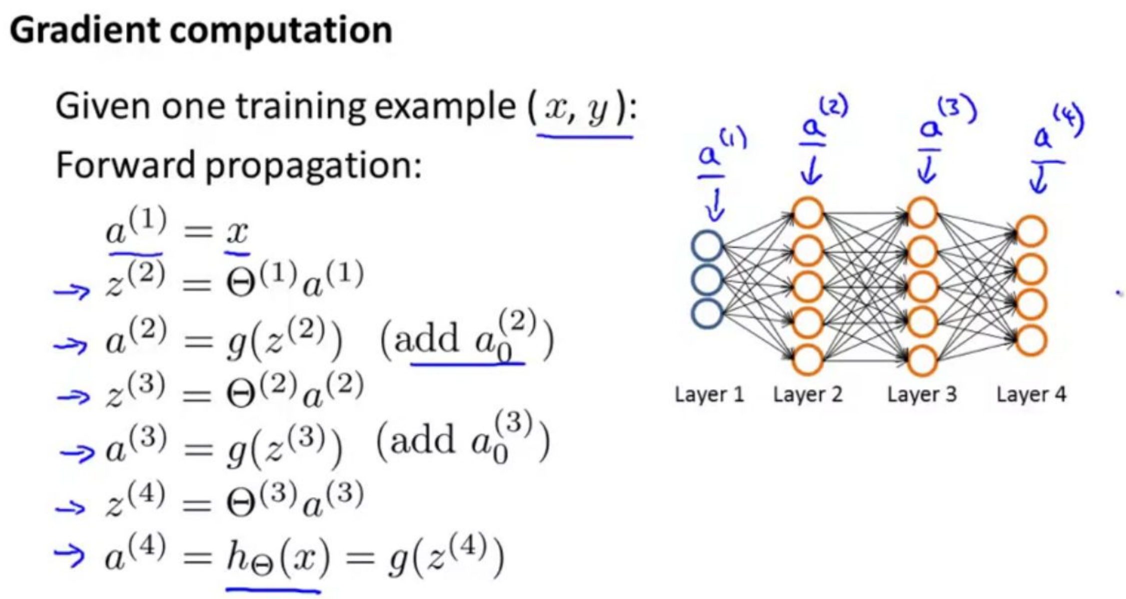
Notice that picture below

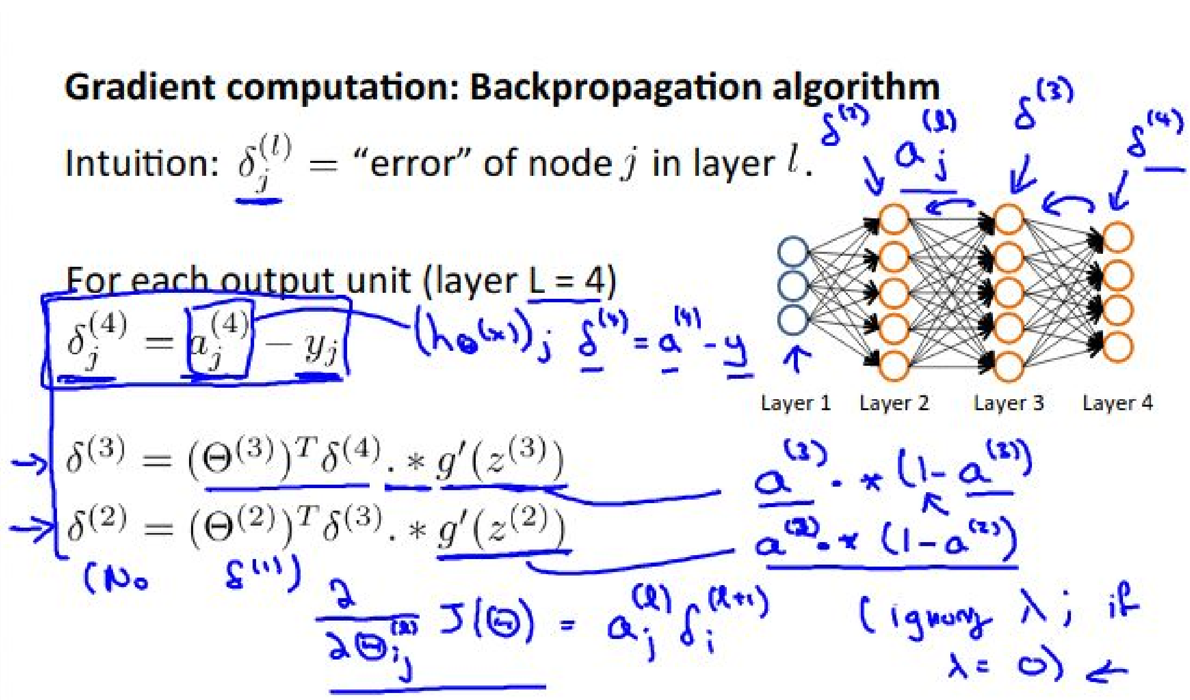


When we calculate for a_1^2 we need $\Theta_{11}\ \Theta_{12}\ \Theta_{13}\ \Theta_{14}$ all together.

Gradient Computation :

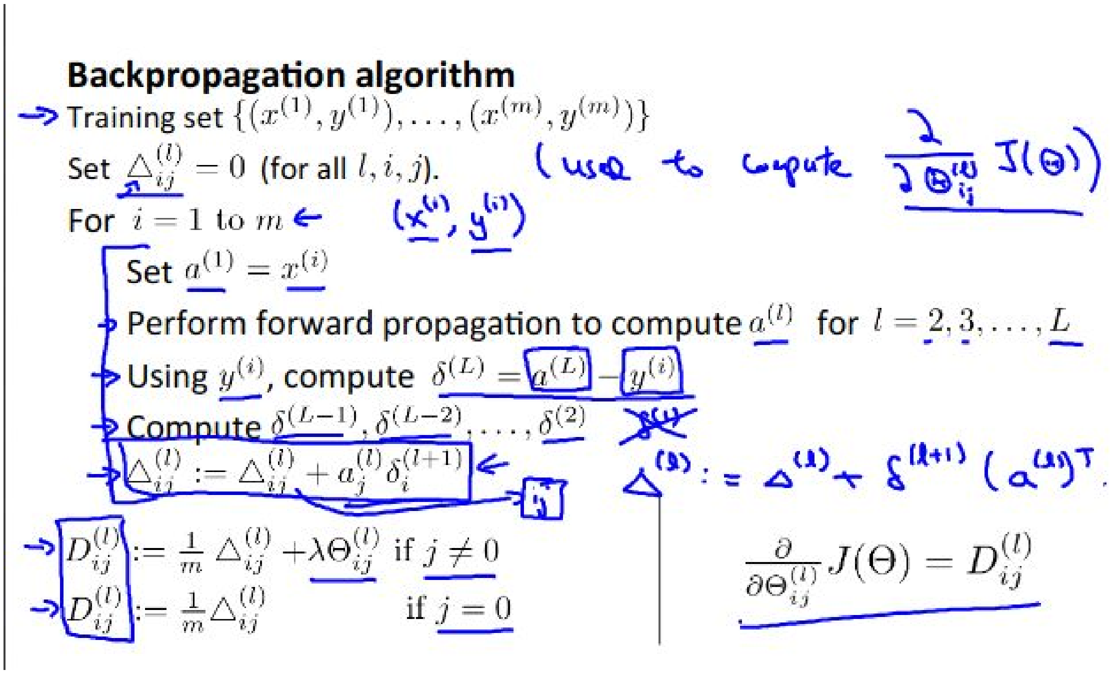
To get the theta which will provide min cost we need to calculate gradient computation.
At first we will do forward gradient computation and then backward computation.

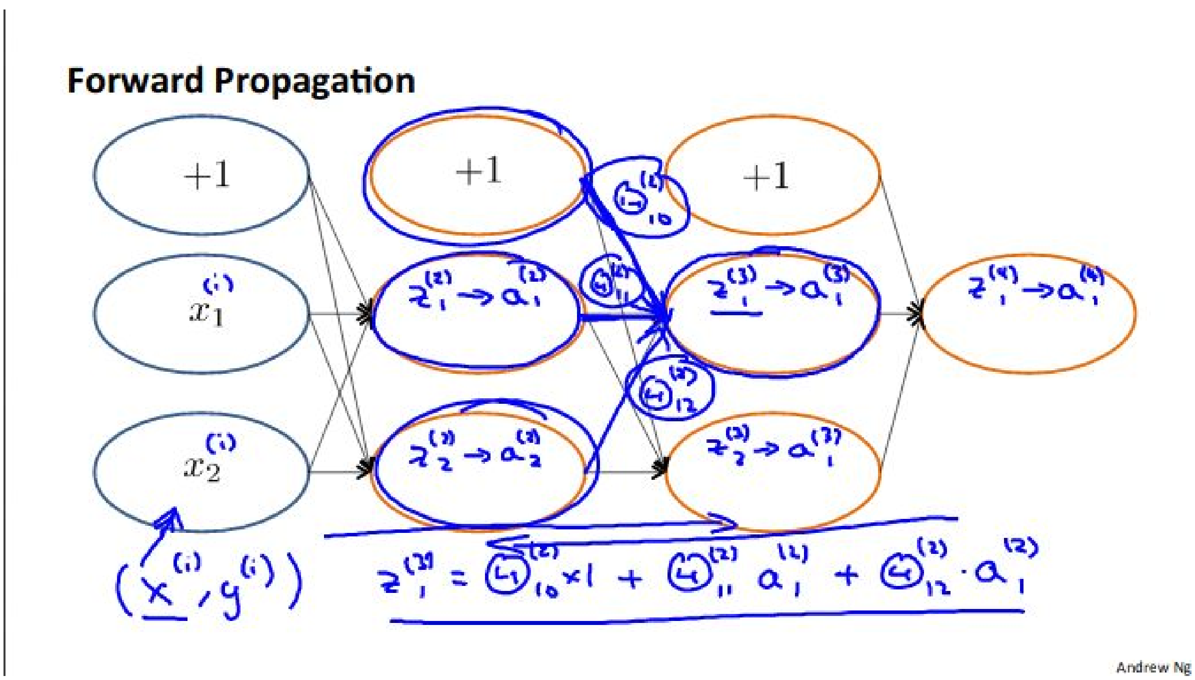


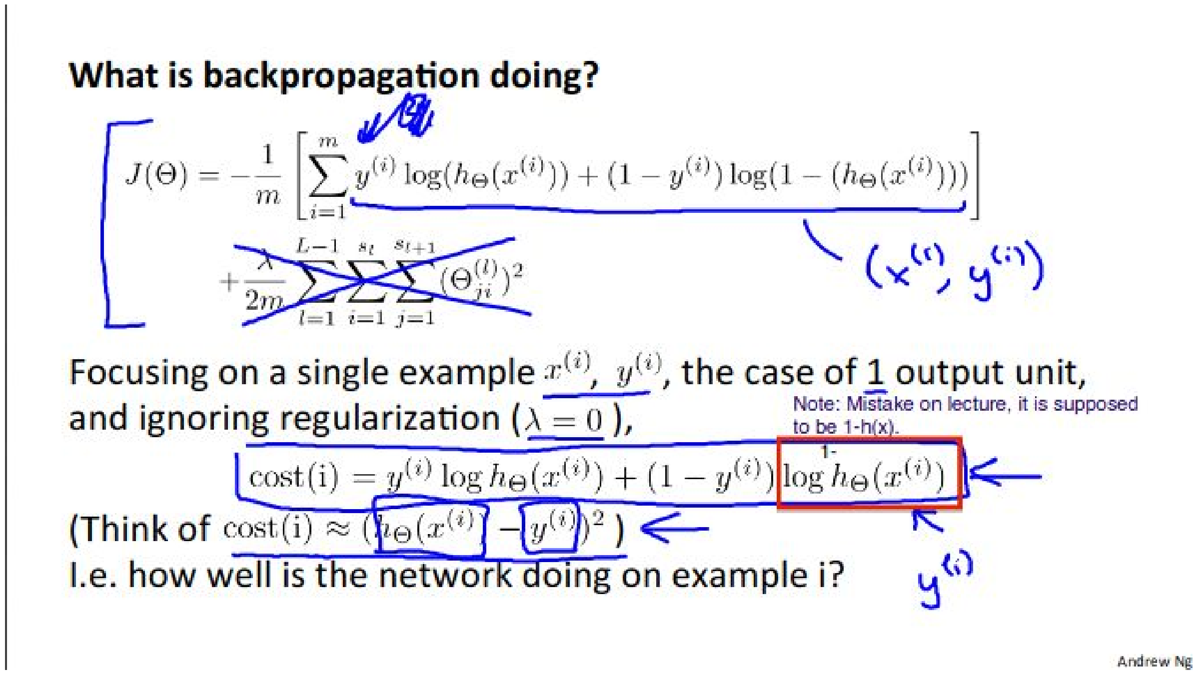


How g'(z3) = a3(1-a3) is explained here.

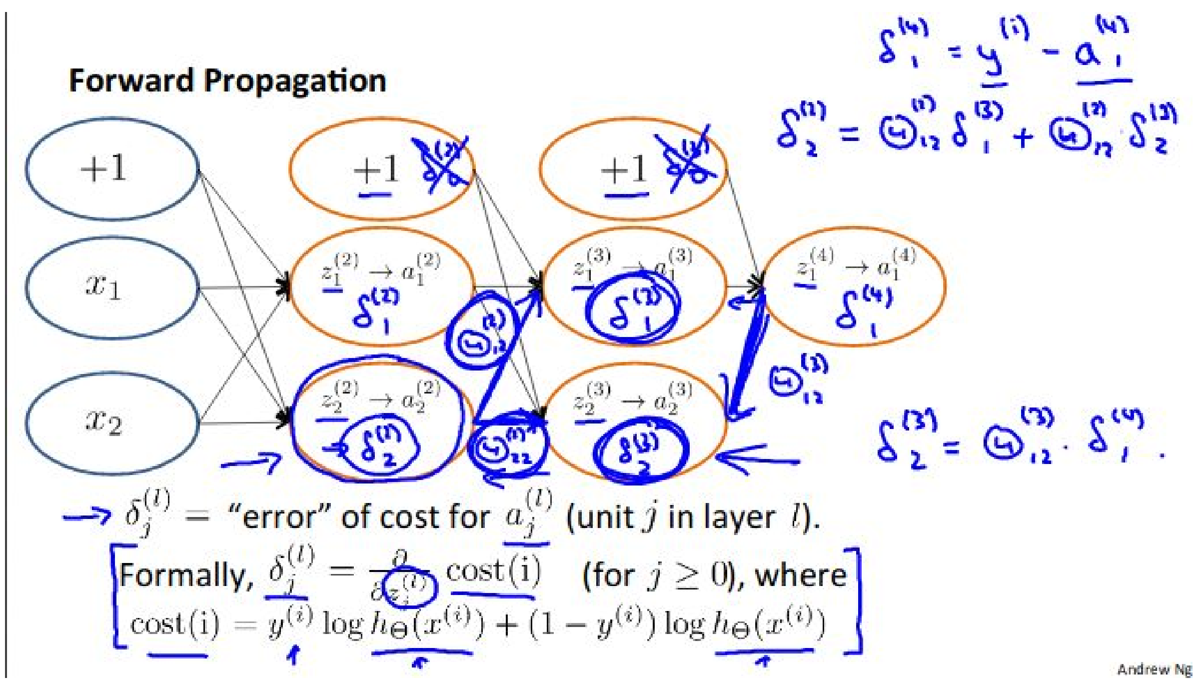
For m number of training sets we can develop a backpropagation algorithm :







If we cut the regularize part of the cost function of the neural network and imagine that we have only one training set, then we can compare it with that simple cost function.



Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
   ...
optTheta = fminunc(@costFunction, initialTheta, options)
```

$\mathbb{R}^{n+1}$ $\mathbb{R}^{n+1}$ (vectors)

Neural Network (L=4):

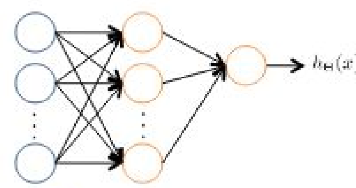
→ $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (`Theta1`, `Theta2`, `Theta3`)

→ $D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (`D1`, `D2`, `D3`)

"Unroll" into vectors

Andrew Ng

Example



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

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

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

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

```
→ 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);
```

Andrew Ng

```
octave:10>
octave:10> theta1 = zeros(10,10);
octave:11> theta2=2*ones(10,20) ;
octave:12> theta3=3*ones(10,1);
octave:13> theta=[theta1(:);theta2(:);theta3(:)];
octave:14> reshape(theta(1:100),10,10);
octave:15> reshape(theta(101:300),10,20);
octave:16> reshape(theta(301:310),10,1);
octave:17>
```

Learning Algorithm

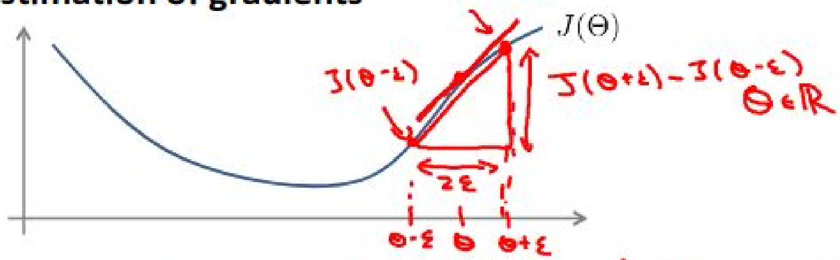
- Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

```
function [jval, gradientVec] = costFunction(thetaVec)
```

- From `thetaVec`, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$. reshape
- Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ $J(\Theta)$
  and $D^{(1)}, D^{(2)}, D^{(3)}$
  Unroll to get `gradientVec`.

Andrew Ng

Numerical estimation of gradients



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

$\varepsilon = 10^{-4} \leftarrow$

$$\cancel{\frac{J(\theta+\varepsilon) - J(\theta)}{\varepsilon}}$$

Implement: `gradApprox = (J(theta + EPSILON) - J(theta - EPSILON)) /(2*EPSILON)`

Andrew Ng

Parameter vector θ

- $\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)
- $\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$

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

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

$$\vdots$$

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

Andrew Ng

```
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;
```

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \varepsilon \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \theta_i - \varepsilon$$

$\frac{\partial}{\partial \theta_i} J(\theta)$.

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

From back prop.

Andrew Ng

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 backprop code for learning.

$\delta^{(4)}, \delta^{(3)}, \delta^{(2)} \to$ DVec

Important:

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

Andrew Ng

Initial value of Θ

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

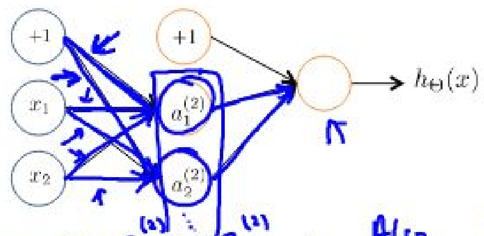
```
optTheta = fminunc(@costFunction,
           initialTheta, options)
```

Consider gradient descent

Set `initialTheta = zeros(n,1)` ?

Andrew Ng

Zero initialization



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

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

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

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

$a_1^{(2)} = a_2^{(2)}$

Andrew Ng

Random initialization: Symmetry breaking

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

E.g. random 10×11 matrix (betw. 0 and 1)

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

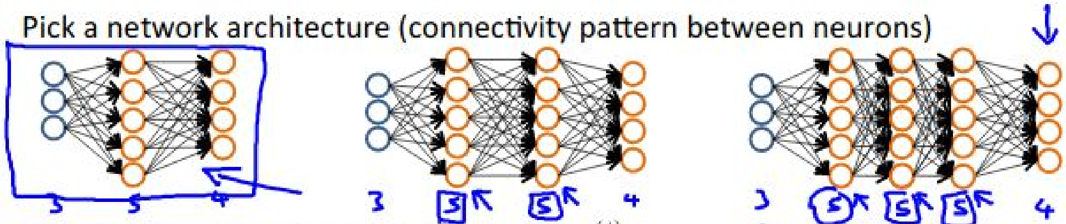
$[-\epsilon, \epsilon]$

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

Andrew Ng

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



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

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

$y \in \{1, 2, 3, \ldots, 10\}$ $\quad \cancel{y = 5}$

$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ or $\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ $\quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

Andrew Ng

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

for i = 1:m { $\quad (x^{(1)}, y^{(1)}) \quad (x^{(2)}, y^{(2)}), \ldots \quad (x^{(m)}, y^{(m)})$

- Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
  (Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).

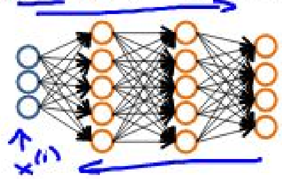
$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

....

}

...

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



Andrew Ng

Training a neural network

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 descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

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

$J(\Theta)$ – non-convex.