Random Initialization

Initial Value of Θ

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

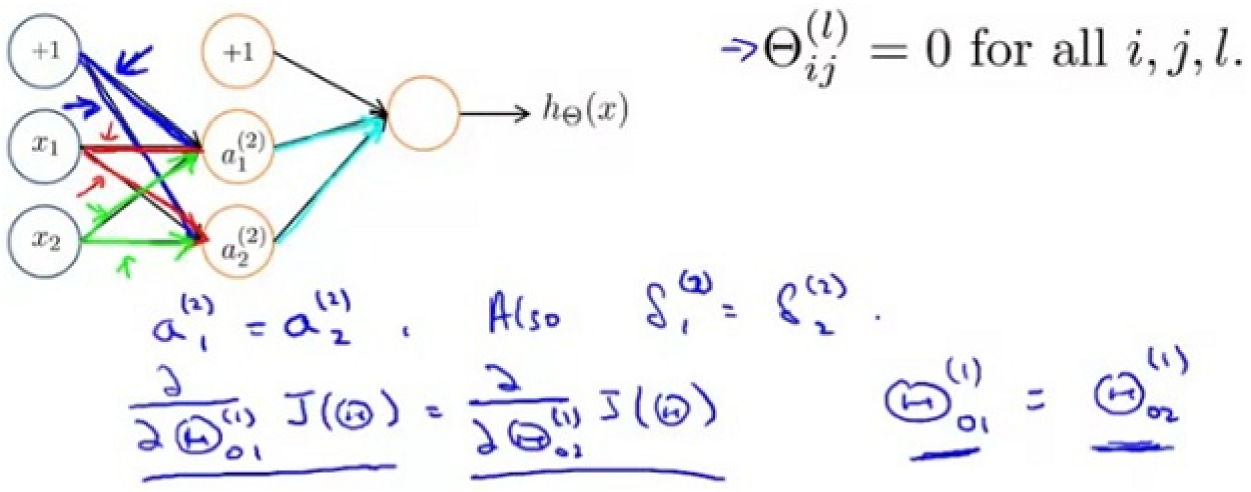
```
optTheta = fminunc(@costFunction, initialTheta, options)
```

Consider gradient descent

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

So, what can we set the initial value of theta to? Is it possible to set the initial value of theta to the vector of all zeros? Whereas this worked well when we were using logistic regression, initializing all of our parameters to zero actually does not work when we are trading on our own network.

If we start them on zero (which does work for linear regression) then the algorithm fails - **all activation values for each layer are the same**. Concretely after each update, parameters corresponding to inputs going to each of two hidden units are identical.



+1
x_1
x_2
+1
$a_1^{(2)}$
$a_2^{(2)}$
$h_\Theta(x)$
$\Theta_{ij}^{(l)} = 0$ for all i, j, l.
$a_1^{(2)} = a_2^{(2)}$. Also $\delta_1^{(2)} = \delta_2^{(2)}$.
$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$
$\Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

In order to get around this problem, the way we initialize the parameters of a neural network therefore is with random initialization. Concretely, the problem that we saw on previously is something called **the problem of symmetric weights**, that's the weights are being the same. So the random initialization is how we perform **symmetry breaking**.

Random initialization: Simmetry breaking

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

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

E.g.

```
Theta1 = rand(10, 11) * (2 * INIT_EPSILON) - INIT_EPSILON;
Theta2 = rand(1, 11) * (2 * INIT_EPSILON) - INIT_EPSILON;
```

rand(10, 11) is a random 10 x 11 matrix (between 0 and 1)

Video Question: Consider this procedure for initializing the parameters of a neural network:

Pick a random number `r = rand(1, 1) * (2 * INIT_EPSILON) - INIT_EPSILON` ; Set $\Theta^{(l)}_{ij} = r$ for all i, j, l.

Does this work?

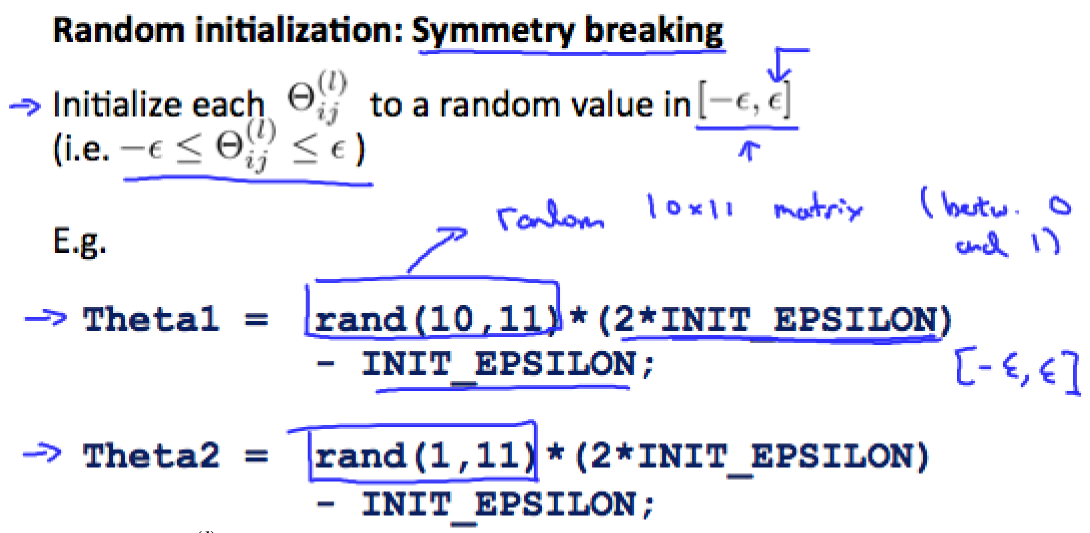
- Yes, because the parameters are chosen randomly.
- Yes, unless we are unlucky and get r = 0 (up to numerical precision).
- Maybe, depending on the training set inputs x(i).

> No, because this fails to break symmetry.

To summarize, to create a neural network what we should do is randomly initialize the weights to small values close to zero, between $-\epsilon$ and ϵ. And then we implement back propagation algorithm, do gradient checking, and use the gradient descent or one of the advanced optimization algorithms to try to minimize $J(\theta)$ as a function of the parameters theta starting from just randomly chosen initial value for the parameters, and by doing symmetry breaking, hopefully gradient descent or the advanced optimization algorithms will be able to find a good value of theta.

Summary

Initializing all theta weights to zero does not work with neural networks. When we backpropagate, all nodes will update to the same value repeatedly. Instead we can randomly initialize our weights for our Θ matrices using the following method:



Random initialization: Symmetry breaking

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

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;
```

Hence, we initialize each $\Theta^{(l)}_{ij}$ to a random value between $[-\epsilon, \epsilon]$. Using the above formula guarantees that we get the desired bound. The same procedure applies to all the $\Theta's$. Below is some working code you could use to experiment.

```
If the dimensions of Theta1 is 10x11, Theta2 is 10x11 and Theta3 is 1x11.

Theta1 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
Theta2 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
Theta3 = rand(1,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
```

rand(x,y) is just a function in octave that will initialize a matrix of random real numbers between 0 and 1.

(Note: the epsilon used above is unrelated to the epsilon from Gradient Checking)