Unrolling Parameters

May 19, 2021

We need to "unroll" the Θ and D matrices to pass into MATLAB functions, since they only accept vectors.

e.g. $s_1 = 10$, $s_2 = 10$, $s_3 = 1$

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

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

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

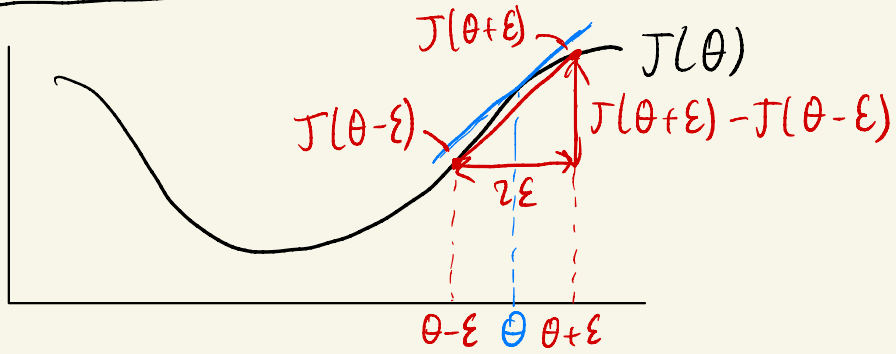
`DVec = [D1(:); D2(:); D3(:)];`

Or:

```
Theta1 = reshape(thetaVec(1:110), 10, 11);
Theta2 = reshape(thetaVec(111:220), 10, 11);
Theta3 = reshape(thetaVec(221:231), 10, 11);
```

Numerical Estimation of Gradients!

e.g.



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

For Parameter Vector θ:

$\theta \in \mathbb{R}^n$ (unrolled)

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

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

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

$$\vdots$$

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

Finally, check that $\frac{\partial}{\partial \theta} J(\theta) \approx$ DVec.

✡ Once you check backprop works, stop computing gradApprox since it is very slow.

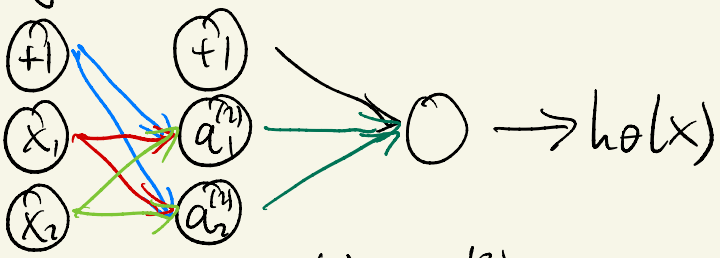
Random Initialization!

What do we initialize Θ as?

↳ Can't set Θ to all zeros (i.e. $\Theta_{ij}^{(l)} \neq 0$ for all i, j, l)

↳ If they are, after each update, parameters corresponding to inputs going into all of the hidden units are identical.

e.g.



↳ $a_1^{(2)} = a_2^{(2)}$

✡ Solve the problem of symmetric weights w/ random initialization → symmetry breaking.

Symmetry Breaking:

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

e.g. → all values of 10×11 matrix are [0, 1]

Theta1 = rand(10, 11) · (2 · INIT_EPSILON) − INIT_EPSILON

Theta2 = rand(1, 11) · (2 · INIT_EPSILON) − INIT_EPSILON

Putting It All Together:

1. Pick a neural network architecture.
   - → # input units = dimension of features $x^{(i)}$
   - → # output units = # of classes
   - → 1 hidden layer, or if multiple, have same # of units in every hidden layer (usually more is better)
2. Randomly initialize weights
3. Implement forward prop to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
4. Implement code to compute cost function $J(\Theta)$
5. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

5. Use gradient checking to compare backprop computed gradient w/ numerical estimate.
   → Then disable gradient checking.
6. Use gradient descent w/ backprop to minimize $J(\theta)$ as a function of parameters θ.