Implementation Note: Unrolling Parameters

Learning Algorithm

Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

Unroll to get initial θ to pass to

fminunc (@costFunction, initialθ, options)

function [jval, gradientVec] = costFunction (thetaVec)

From thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

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

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

Gradient Checking

- can approximate the derivative of cost f with:

$$\frac{\partial}{\partial \theta} J(\theta) \approx \frac{J(\theta-\epsilon) - J(\theta+\epsilon)}{2\epsilon}$$

Random Initialization

- 모든 Θ 가중치를 0으로 초기화하는 것은 신경망에 부적합
- Symmetry breaking

Putting It Together

Training a neural network

0. Pick a network architecture (뉴런 간 연결 패턴)
   - no. of input units, hidden layers, output units
   Dimension of features $x^{(i)}$ (usually the more the better) no. of classes

1. Randomly initialize weights Θ

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

5. Use gradient checking

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