5주차 - Neural Networks : Learning

<Cost Function and Backpropagation>

1. Cost Function

```
* L = total number of layers in the network
* s_l = number of units (not counting bias unit) in layer l
* K = number of output units/classes
```

인공신경망에서 K클래스의 다범주 분류 비용함수는 아래와 같이 나타낼 수 있다.

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

이것을 multiclass classification에서 one-vs-all 예측법과 헷갈리면 안된다.
one-vs-all은 모든 클래스에 대해서 동시에 비용함수를 구하는 것이 아닌, 각각의 클래스에 대해서 이진 분류를 마친 뒤에 hypothesis를 구한 뒤 가장 확률이 높은 것을 선택하는 것이었다면, 인공신경망(NN)에서는 애초에 전 클래스를 동시에 고려하는 것이다.
-> 참조
https://www.coursera.org/learn/machine-learning/discussions/weeks/5/threads/WmCWI7CPEeaTRBLTy5Qz8A

Note

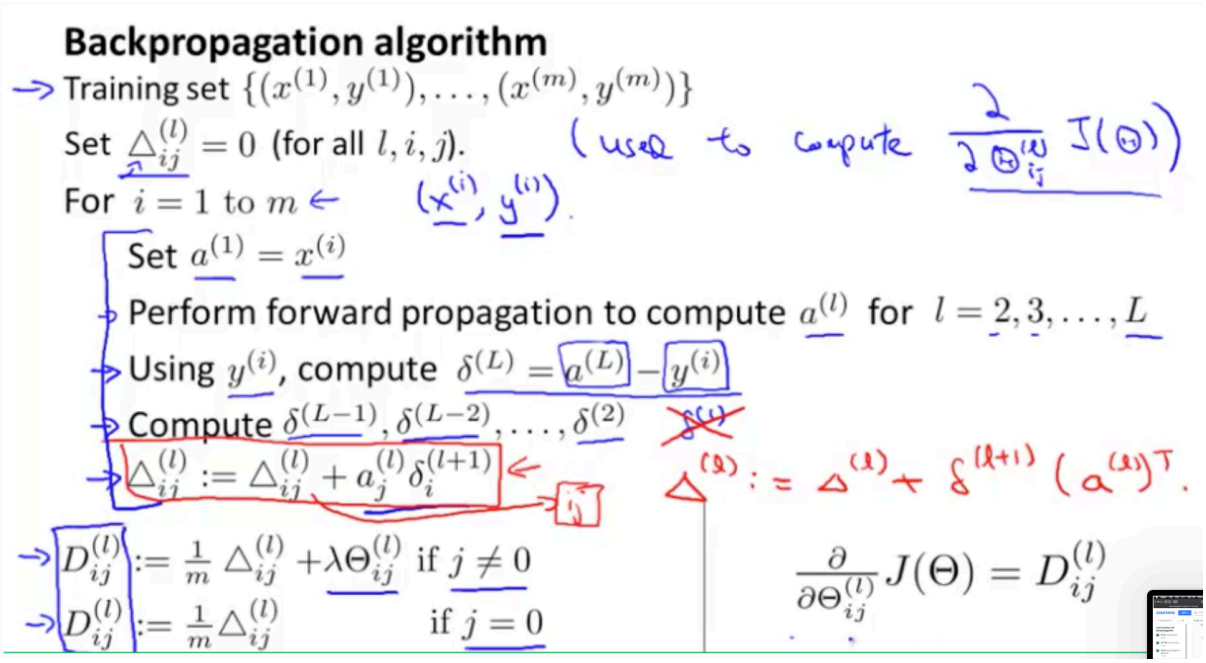
```
* the double sum simply adds up the logistic regression costs calculated
for each cell in the output layer
* the triple sum simply adds up the squares of all the individual Θs in the
entire network.
* the i in the triple sum does not refer to training example i
```

2. Backpropagation Algorithm

역전파 알고리즘은 인공신경망의 비용함수를 최소화하는 알고리즘이다. 즉 다음과 같은 목표를 가진다.

$$\min_\Theta J(\Theta) \quad \frac{\partial}{\partial\Theta_{i,j}^{(l)}}J(\Theta)$$

대략적인 알고리즘은 다음과 같다.



2 ~ L-1 층까지 델타는 다음과 같은 방법으로 구한다.

$$\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)} \text{ using } \delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$$

로지스틱 회귀와는 다르게 가중치가 3차원이라 식이 매우 복잡하다. 구체적인 흐름은 아래 증명 페이지를 참고할 수 있다. 하지만 결국 위 과정은 비용함수를 최소화 하기 위하여 가중치에 대한 비용함수의 도함수를 구하는 과정일 뿐이며, chain rule을 이용하여 좀 더 직관적으로 그리고 간편하게 도함수를 구하는 것 뿐이다.
chain rule : https://en.wikipedia.org/wiki/Chain_rule
구체적 증명 : Backpropagation : process
첨언하자면, 여기서 big delta는 결국 시그마를 달리 표현한 것과 다름 없다. 로지스틱 회귀에서 시그마(1~m)이 같은 역할을 한다. 결국 각 example에 대하여 비용함수의 도함수를 구하여 모두 더한 뒤 평균을 내는 것이다.

로지스틱 회귀는 theta의 변화가 바로 비용함수에 영향을 미친다. 하지만 인공신경망은 input layer의 theta가 다음 활성화 unit에 영향을 미치고 그 활성화 유닛과 그 층의 theta는 다시 다음 층에 영향을 주는 구조이다. 따라서 비용함수의 도함수를 구하는 과정도 층마다 스텝을 거쳐야한다. 이것이 바로 오차 역전파의 핵심 흐름이 아닐까 싶다.

<Backpropagation in Practice>

1. Unrolling Parameters

로지스틱 회귀와 다르게 인공신경망의 theta와 gradient descent는 vector가 아닌 matrix이다.
따라서 비용함수를 최소화 알고리즘을 사용할 때 matrix를 vector로 변환하여 주어야한다.
-> theta와 gradient의 층마다 matrix의 size가 달라서 하나의 matrix로 만들 수 없기 때문.
이 과정을 'unrolling Parameters'라고 한다.

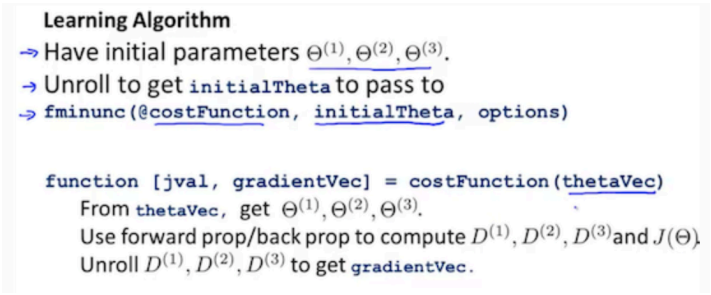
- unroll

```
thetaVector = [ Theta1(:); Theta2(:); Theta3(:); ]
deltaVector = [ D1(:); D2(:); D3(:) ]
```

- roll

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

전체적인 알고리즘 사용법



<Gradient Checking>

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

아주 작은 값(엡실론)에 대한 평균 변화율은 순간 변화율과 거의 같은 값을 가진다.
이를 이용하여 오차역 전파 알고리즘이 얼마나 잘 작동하는 지 알 수 있다.
만약 오차역 전파 알고리즘이 잘 작동한다면 오차역 전파로 구한 gradient와 이 값이 거의 일치할 것이다.

theta가 메트릭스일 경우 unroll 과정을 거쳐서 각 요소에 대하여 적용한다.

$$\frac{\partial}{\partial\Theta_j}J(\Theta) \approx \frac{J(\Theta_1,\dots,\Theta_j+\epsilon,\dots,\Theta_n)-J(\Theta_1,\dots,\Theta_j-\epsilon,\dots,\Theta_n)}{2\epsilon}$$

```
epsilon = 1e-4;
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) += epsilon;
  thetaMinus = theta;
  thetaMinus(i) -= epsilon;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)
end;
```

gradient checking은 학습을 시작하기 전에 check하는 용도로 사용하는 것이다.

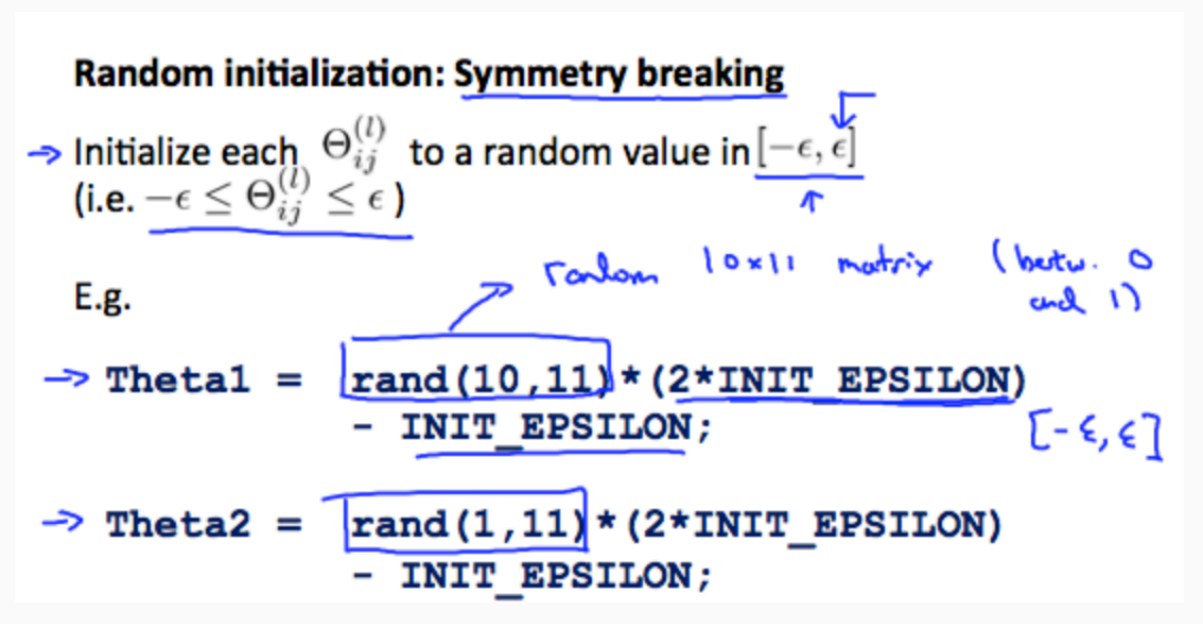
평균 변화율을 구하는 방식은 매우 느리기 때문에 학습 데이터를 가지고 테스트를 해보는 것은 시간이 오래 걸린다. 몇가지 예시로 테스트를 해보고 gradient가 잘 작동하는 것 같으면 테스트를 그만하고 학습을 시작하면 된다.

참고 : 왜 gradient checking이 오차역전파를 증명해줄 수 있는가?
답 : gradient checking에서 쓰는 평균변화율이 엡실론이 작을 경우 순간 변화율로 근사할 수 있기 때문에 기준값이 될 자격이 있다.
https://www.coursera.org/learn/machine-learning/discussions/weeks/5/threads/XFofPueaEeiCnxJgKwvJrA

참고 : 오차역 전파에 무슨 문제가 있길래 gradient checking이 필요한 것인가?
답 : 비용함수나 활성화함수의 구현에 따라서 혹은 단순 코딩 상 버그로 발생할 수 있는 계산 오류를 잡아낼 수 있기 때문.
https://www.coursera.org/learn/machine-learning/discussions/weeks/5/threads/W2WFeIX9Eei3XQ6UbYvWCA

<Random Initialization>

random Initialization은 초기 parameter theta를 랜덤한 값으로 제공하는 것을 말한다.
이름을 들어보면 진화적으로 합리적인 초기값을 제공한다고 생각할 수 있지만 실제로 theta안의 요소 값이 모두 같을 때 각 층의 모든 unit은 같은 활성화 값을 가지게 되고, 계산해보면 쉽게 알 수 있는데 오차역 전파과정에서 delta값도 모든 unit이 같게 된다. 이는 곧 gradient descent를 한번 실행했을 때 theta의 변하는 정도가 각 층 모든 unit이 같다는 것. 이것은 결국 각 층의 unit들이 차별화되지 못 한다는 것이고 불필요한 특징값이 존재한다는 말이 된다. 따라서 random initialization은 이런 문제를 회피할 수 있는 한 방법이다.



```
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;

```

<Putting it together>

```
First, pick a network architecture; choose the layout of your neural
network, including how many hidden units in each layer and how many layers
in total you want to have.
* Number of input units = dimension of features x(i)x(i)
* Number of output units = number of classes
* Number of hidden units per layer = usually more the better (must balance
with cost of computation as it increases with more hidden units)
* Defaults: 1 hidden layer. If you have more than 1 hidden layer, then it
is recommended that you have the same number of units in every hidden
layer.
```

```
Training a Neural Network
1. Randomly initialize the weights
2. Implement forward propagation to get h0(x(i)) for any x(i)
3. Implement the cost function
4. Implement backpropagation to compute partial derivatives
5. Use gradient checking to confirm that your backpropagation works. Then
disable gradient checking.
6. Use gradient descent or a built-in optimization function to minimize the
cost function with the weights in theta.
```

-참고-
인공신경망에서 비용함수는 일반적으로 볼록함수라고 확정할 수 없다. 따라서 단순히 local minimum 을 찾을 수도 있지만 실제 상황에서 좋은 결과를 내는 것을 볼 수 있다.