**Logistic Regression**

H(X) = WX 의 가설은 {100, 200, -10} 등 다양한 Y를 도출한다.

Logistic Regression을 사용하면, 우리의 가설의 결과를 0~1 까지의 숫자로 나타내 준다.

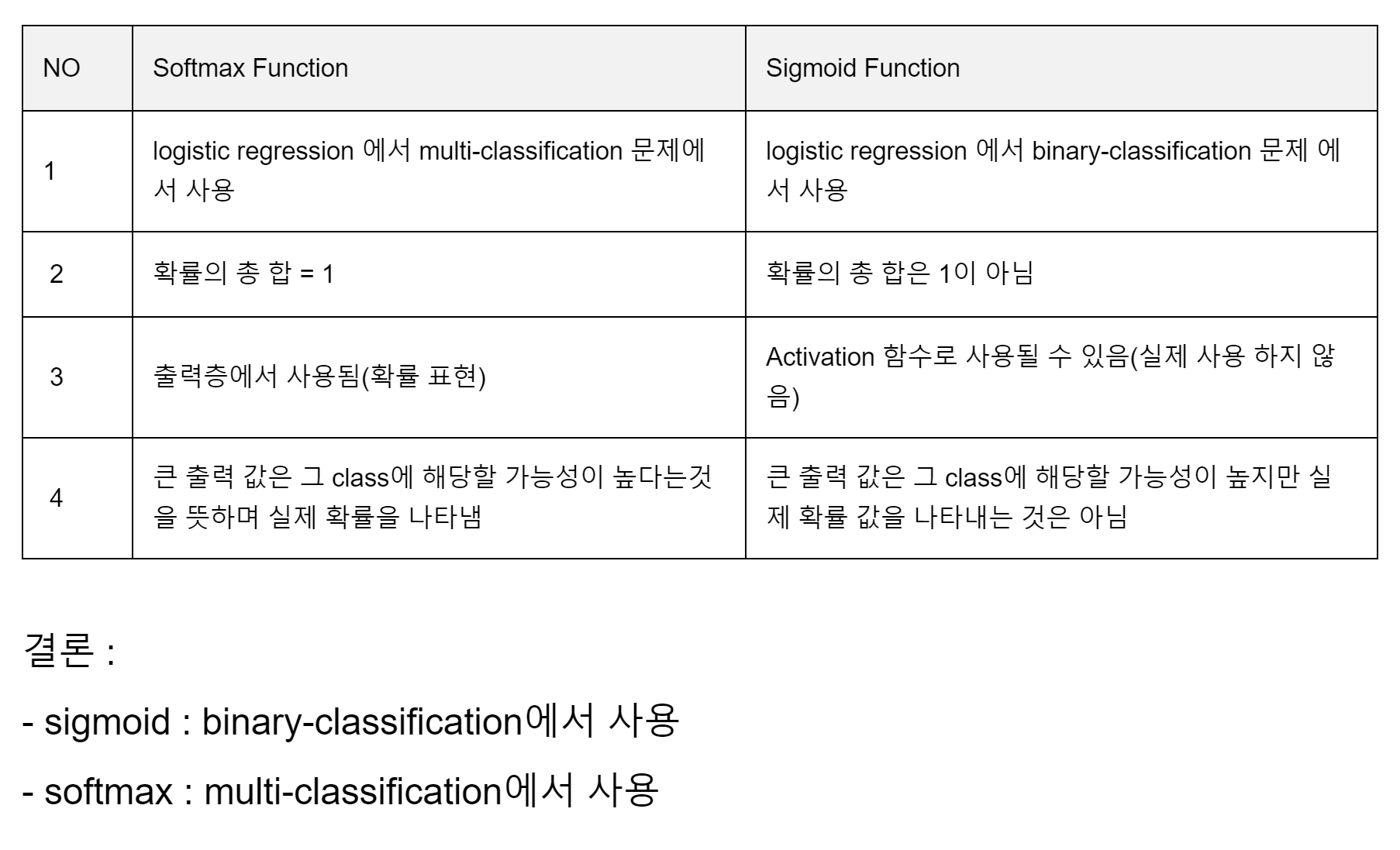
하지만 Binary Classfication의 형식이기 때문에 Multi Classfication을 사용하고자 할 때는 부적합하다.

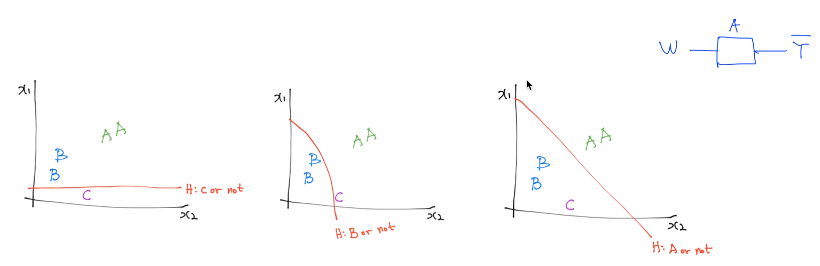
때문에 Multi class를 위해서, H(X)를 어딘가의 함수에 인수로 넣고   
0과 1사이의 값(Y-hat)을 도출해내는 방정식을 생성할 것이다.

즉, H(X) = Z , g(Z)로 하여금 Y값을 0과 1사이의 값으로 변화시킨다

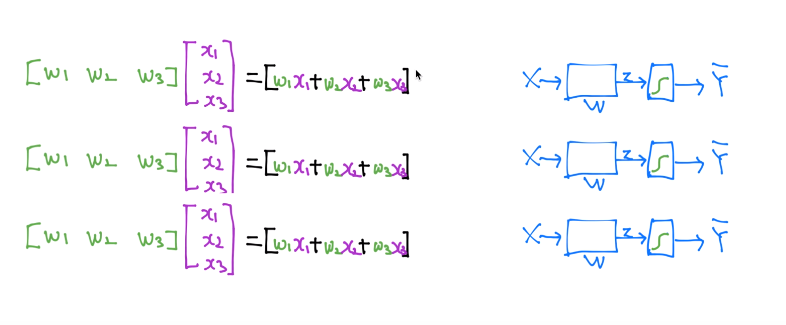
이때 사용하는 함수가 g(Z) = 1 / 1+e^2 (Logistic / Sigmoid Function) 이다.

결론적으로 우리가 구하는 Softmax function은 다중 분류를 위한 Logistic function 이며, H(X) = g(H(X)) 가 된다.



위와 같이 3개의 hypothesis를 통하여 1. C인지 아닌지, 2. B인지 아닌지, 3. A인지 아닌지 판별하게 된다.  
(3개의 Logistic Function)

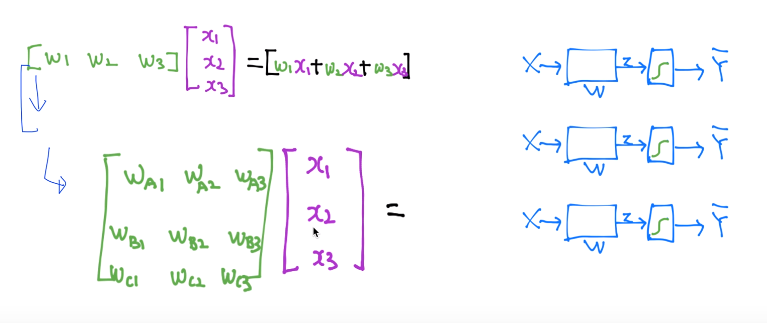
즉,



위와 같은 Matrix 형식을 3번 반복하면 A인지 B인지 C인지 판별할 수 있다는 의미이다.

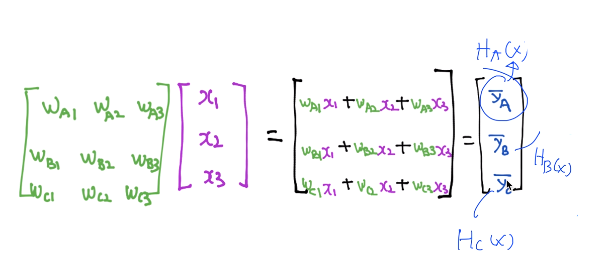
하지만 이와 같은 방법은 비효율적이며 복잡하다.

때문에 이를 하나의 행렬로 모아서 계산한다.



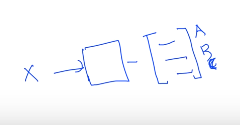
이 Matrix의 계산 결과는 아래와 같다.

하나의 Matrix 곱으로 A, B, C를 판별해내는 3개의 hypothesis를 나타낼 수 있다.  
(Softmax Function)



위와 같은 Matrix 계산을 통하면, X(test data)를 입력하였을 때 [[A], [B], [C]] 의 (3, 1)=Vector 가 도출된다.

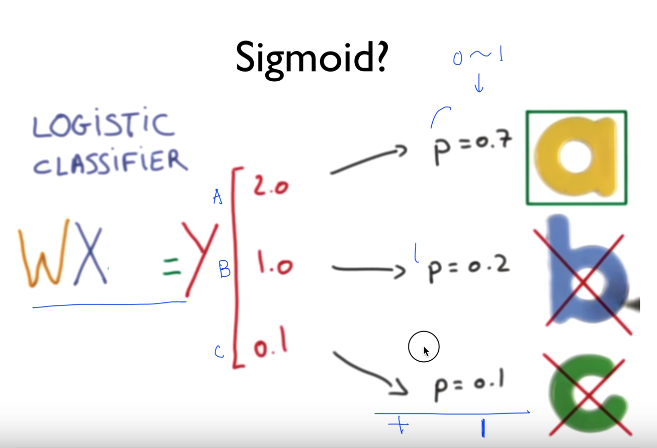
A가 될 확률, B가 될 확률, C가 될 확률



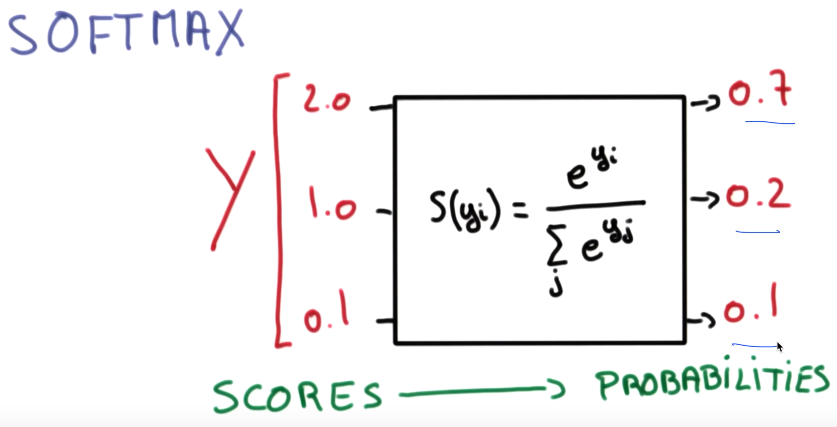
하지만 아직까지 우리의 Logistic Regression 모델은 다양한 수를 도출해 낸다. (2.0, 1.0, 0.1 ..)

우리는 0~1 사이의 값으로 Y값을 도출하고 싶고, 도출된 Y값을 합쳤을 때 1이 나오도록 하고 싶다.

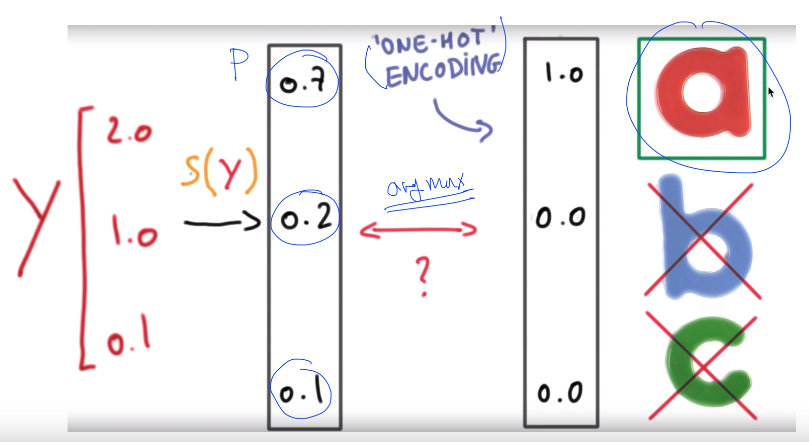
떄문에 도출된 Y값의 모음(Vector)에 각각 Sigmoid 함수를 적용시켜서 이를 만들어 내야한다.



하지만, Multi Classification 의 경우에는 각각 Sigmoid를 적용 시키지 않고, Softmax 함수를 사용하여 이를 한번에 해결한다.



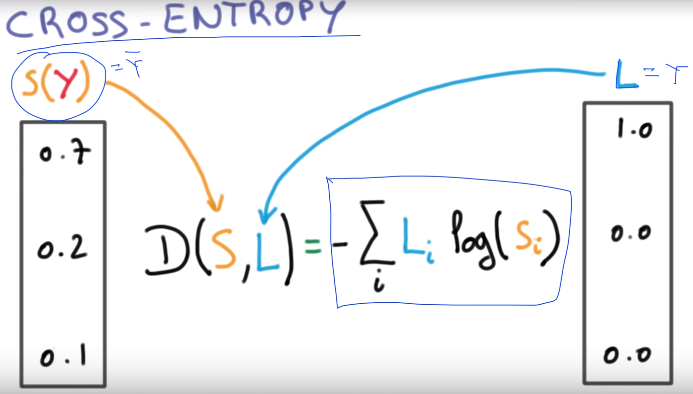
도출된 Lable 의 Vector 를 그대로 표현할 수도 있지만, ONE-HOT ENCODING 기법(각주: 기법이다 기법, 기법임을명시하자)을 활용하여 가장 HOT한(우리가 주목하고 관심을 두고있는) 값만 1로 나타내고 나머지는 0으로 나타내는 ENCODING을 거쳐서 A를 도출해낸다. (argmax 메소드 사용)



예측한 값(H(X))과 실제 값(Y)의 차이를 줄이기 위한 함수, Cost Function!

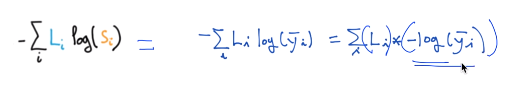
* Cross-Entropy는 위와 마찬가지로 오차를 줄이는 함수이다.

D(예측한 값, 실제 값) = => /

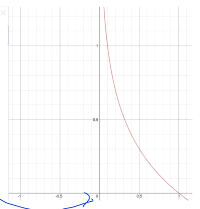


Cross-Entropy cost function

1. 마이너스를 안쪽으로 옮겨서 -log를 만들어준다.
2. 여기서 Li와 -log(Si)는 element의 곱이다.
3. Si(예측한 값)은 Logistic Regression 이후 Softmax 까지 처리된 상태이므로 0~1의 값이다.



1. 즉, 우리의 관심영역은 X축 기준의 0~1사이의 값이다.



Example, L(실제값) 이 [ [0], [1] ] 이 주어졌을 때,

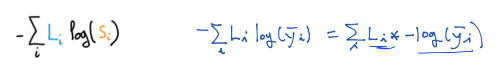
Y1 = [ [0], [1] ] 와 Y2 = [ [1], [0] ] 라는 가설 두개가 있다.

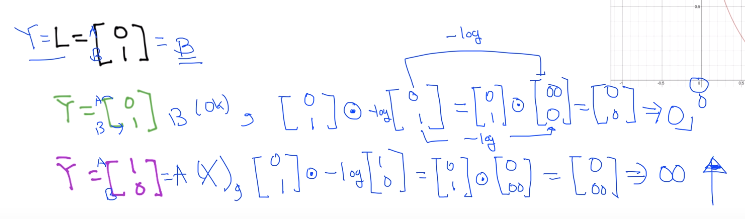
이때 각각의 Cost를 구하면 아래와 같다.

Y1의 경우 / Li와 -log를 취해준 Si를 element 곱 해주면 [ [0], [0] ] 이 나오는데 바깥에 시그마가 있으므로 각각의 element를 더해주면 cost = 0이 도출된다

Y2의 경우 / 위와 동일한 방법으로 계산해주면 cost = 무한대 가 도출된다.

즉 가설이 부적합한 경우 cost가 높고, 가설이 적합한 경우(맞는 경우) cost가 줄어든다.  
이를 활용하여(cost를 줄여가는 방식) 적합한 모델을 찾아가는 방법이 Cost Function (Cross-Entropy Function)이다





그때 회의중에 나온 궁금증 중에

float32 에 관한거랑

axis=1 에 관한게 있었는데

지금 정리하면서 그 답을 찾은거 같아서요

그때 이야기 나눈것 처럼 float32는 32bit, float64는 64bit 까지 저장할 수 있는 공간으로 구별되는게 맞구요

이거에 따른 차이점은 메모리 점유에 따른 속도 향상 <-> 부동 소수점 표현에 따른 정밀도 향상의 장단점이 있는거 같아요

axis=1은 우리가 데이터로 제공하는 matrix의 골격?(묶음 구조)를 어디에 관심을 두고 계산을 진행할지 정하는데 필요한 옵션이랍니다,

# axis의 개수는 먼저 행렬의 rank의 값과 동일하고. axis를 카운트하는 방식은 배열에서 가장 바깥쪽 덩이를 시작으로 0부터 카운트하는게 기본 방식이라 합니다

예를들어서

[ [0, 1, 2]

[0, 1, 3]

[0, 1, 4] ]

의 2차원 matrix가 있으면

# axis: 0 => 가장 바깥쪽 묶음 기준(행)

# axis: 1 => 가장 바깥쪽 묶음 바로 안쪽 기준(열)

# 이런식으로 가장 외측(axis: 0)기준 안쪽 그룹은 +1씩 Count 됩니다

# 추가적으로 axis: -1 => 가장 안쪽에 있는 묶음 기준을 부르는 다른 표현이라고 하네요