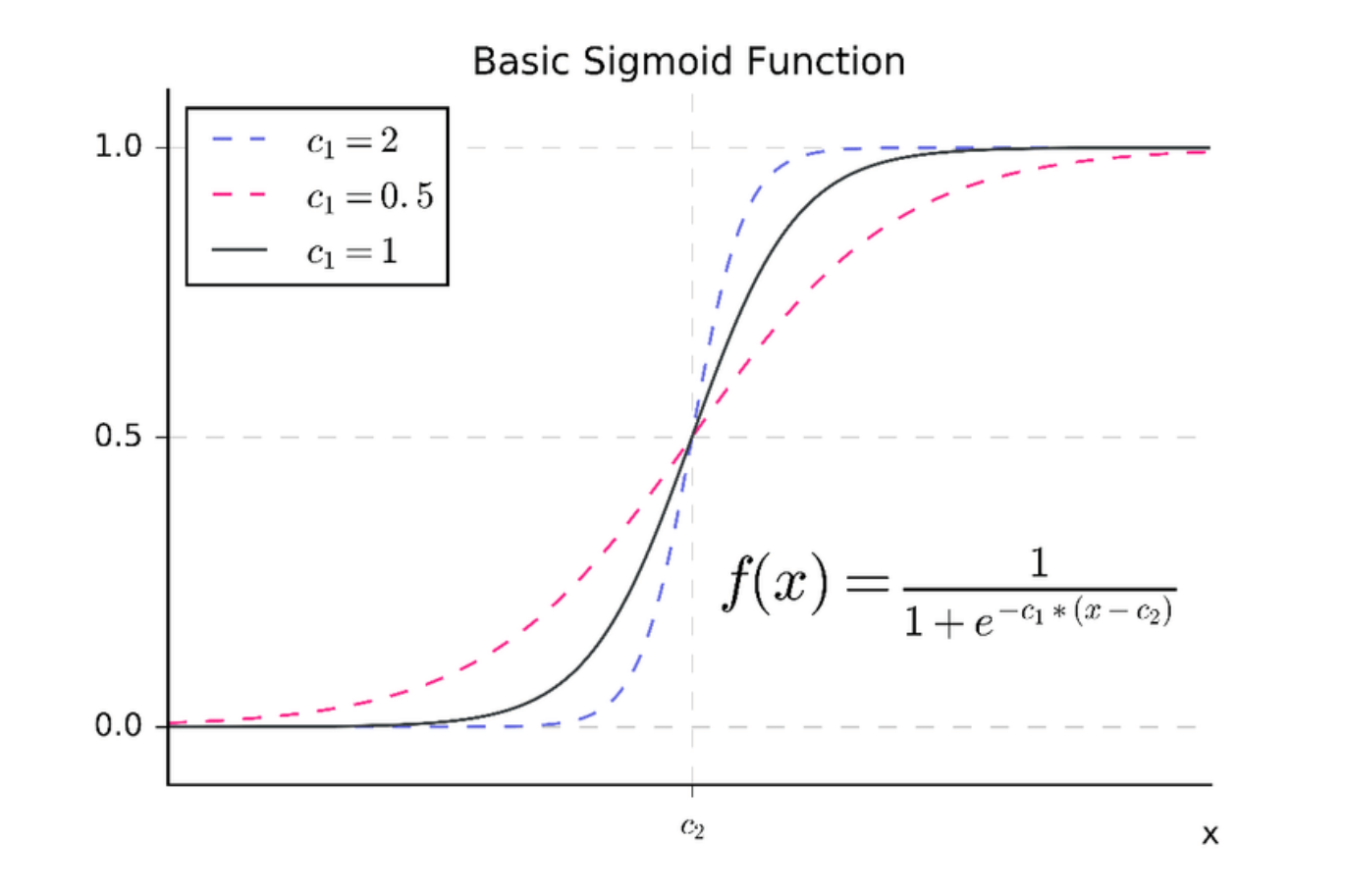
softmax

기존 LG

$$Wx$$

실수 전체를 범위로 가지고 있다.
일차함수의 개형
선형적으로 분포되어 있는 데이터 분류에 적절

기존 로지스틱 회귀

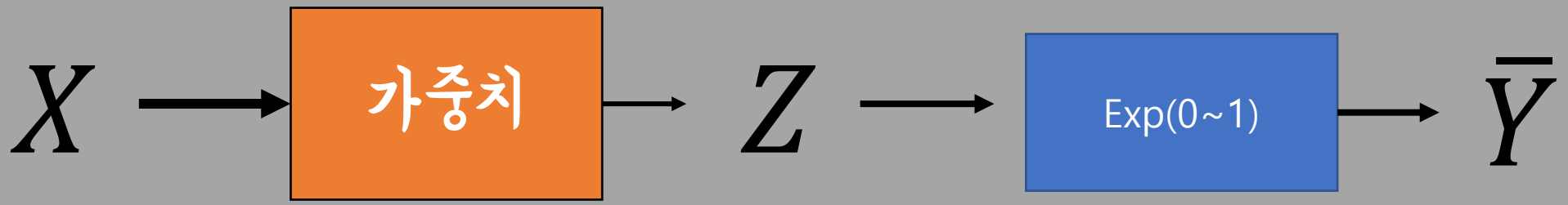


기존 로지스틱 회귀는,
시그모이드 함수의 개형을
수정하여, 데이터의
범주를 나누는
정확한 가중치를
찾아내는 데에 있다.

또한 종속변수는 0과1로
고정된다.

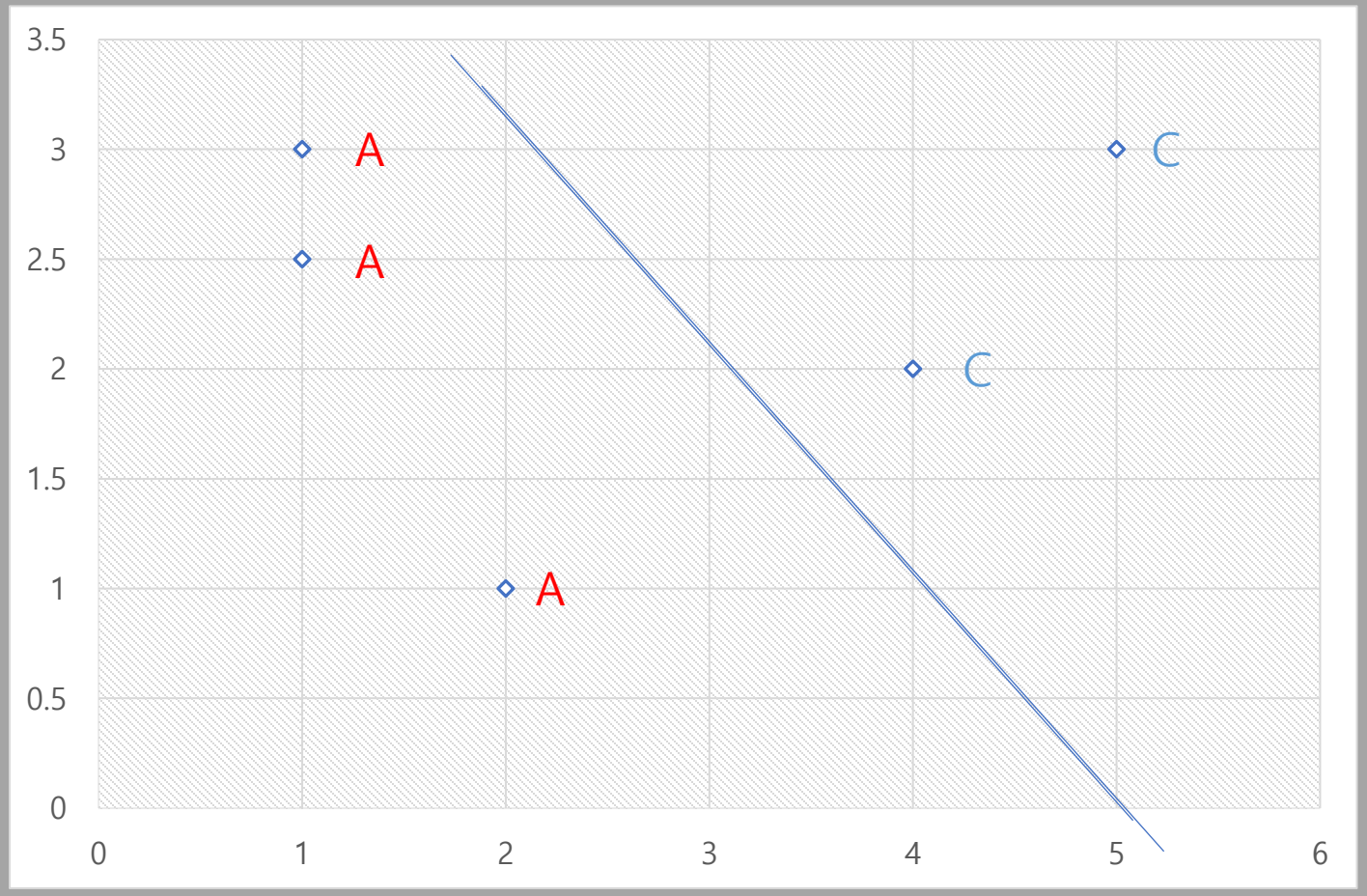
알고리즘 순서

$$\frac{1}{1+e^{-z}}$$



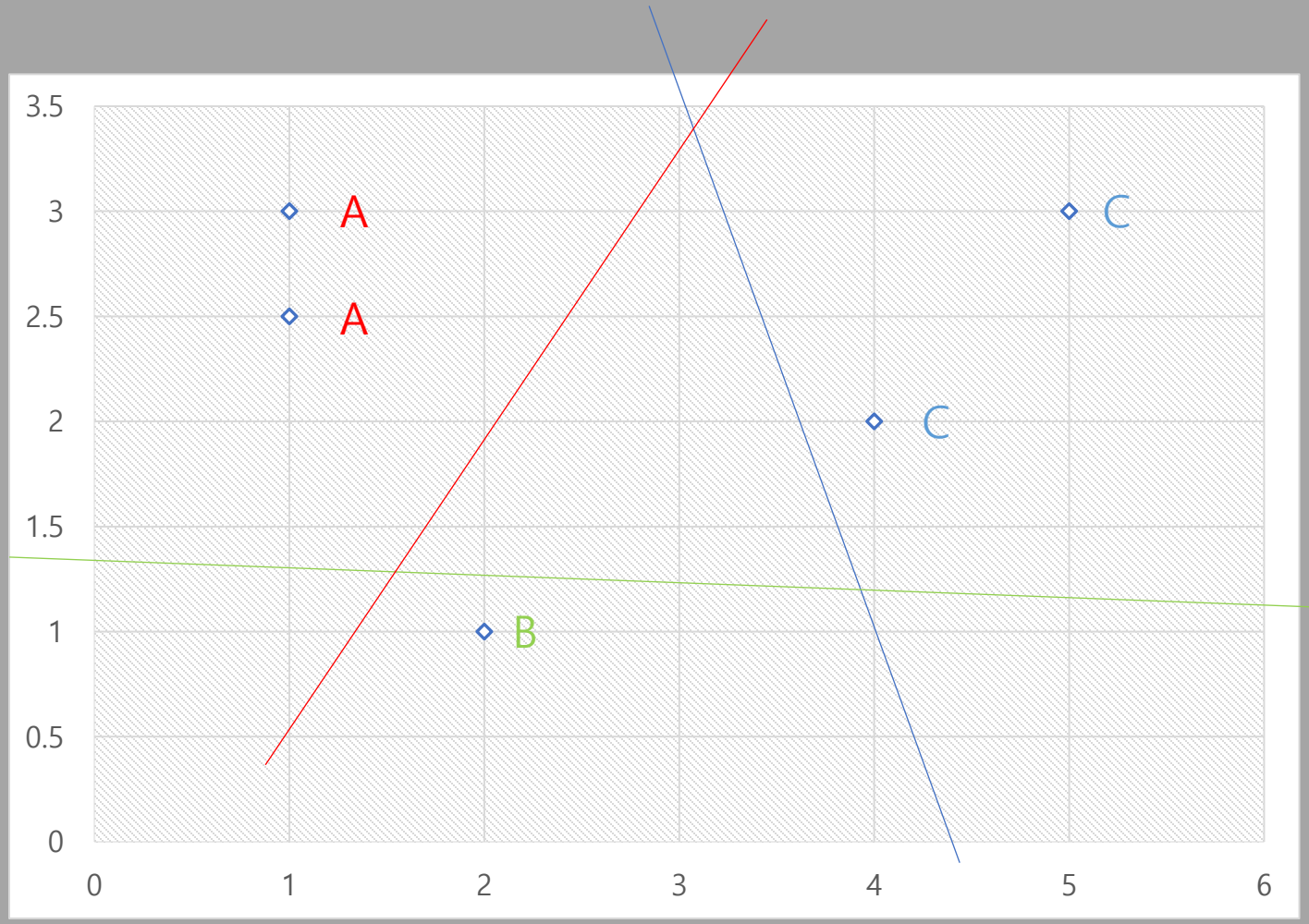
Y:original label
$\bar{Y}$:predict label

기하적인 이해



사실상 로지스틱이 가진 의의는
입력된 feature 값들이 가질
Label을 정확하게 예측하는 데에 있다.

기하적인 이해



이런 식으로 각각 범주의 개수만큼 Hyper plane(구분선)이 생기고, 이에 대한 가중치를 feed back 한다고 치자.

그러면 아래와 같이 세가지의 분류에 대한 확률을 구하는 모델이 있어야 할 것이다

A classification

B classification

C classification

행렬 (matrix)

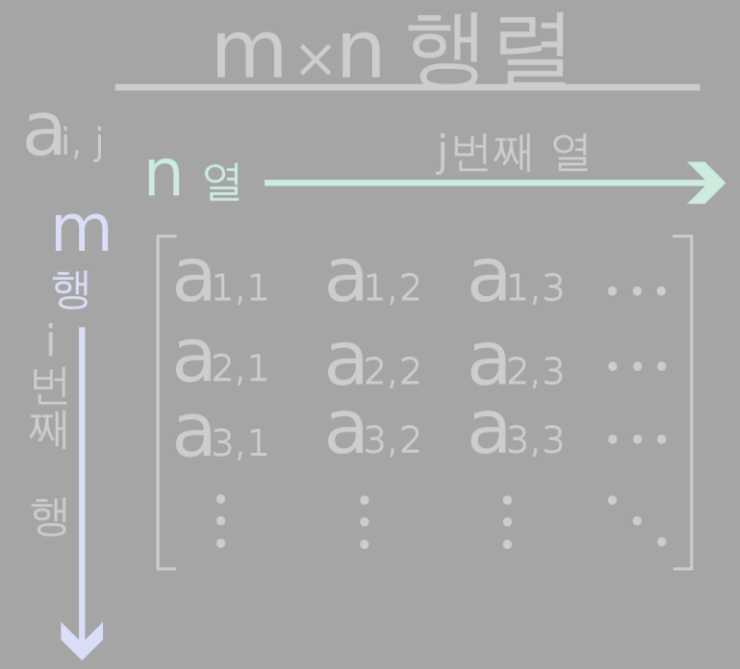
1개 이상의 수 또는 다항식을 직사각형 모양으로 배열한 것이다.

행: 행렬의 가로줄

열: 행렬의 세로줄

성분: 행렬 안에 배열된 구성원

$m \times n$행렬: m개의 행과 n개의 열로 이루어진 행렬



행렬 문제

몇행 몇열?

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

행렬의 종류

주대각선 : 행렬의 왼쪽 위에서 오른쪽 아래를 가르는 선

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

영행렬 : 모든 성분이 0인 행렬

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

전치행렬 : 주대각선을 기준으로 행렬을 뒤집은 행렬

A

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

여러가지 행렬

대칭 행렬 : 원래 행렬과 전치 행렬이 같은 행렬

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{pmatrix}$$

정사각형 행렬: 행, 열의 개수가 같은 행렬

$$\begin{pmatrix} 1 & 2 & 4 \\ 8 & 3 & 9 \\ 5 & 6 & 7 \end{pmatrix}$$

단위 행렬: 모든 대각 성분이 1이고, 그 외의 선분은 0인 정사각형 행렬

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

행렬의 연산

1. 행렬의 덧셈:
   행렬의 각 성분들끼리 더해준다.

$$\begin{pmatrix} 1 & 2 & 4 \\ 8 & 3 & 9 \\ 5 & 6 & 7 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 4 \\ 3 & 2 & 5 \\ 6 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 7 \\ 7 & 7 & 11 \\ 13 & 15 & 11 \end{pmatrix}$$

행렬의 스칼라배(상수배)

곱해지는 수를 분배법칙 하는 것 처럼 곱해준다.

$$2 \times \begin{pmatrix} 1 & 3 \\ 5 & 9 \end{pmatrix} = \begin{pmatrix} (2 \times 1) & (2 \times 3) \\ (2 \times 5) & (2 \times 9) \end{pmatrix}$$

$$4 \times \begin{pmatrix} 1 & 2 & 4 \\ 5 & 6 & 11 \\ 3 & 12 & 7 \end{pmatrix} = ?$$

Dot product(내적)

$$\begin{matrix}1 & 2 & 3 \\ 4 & 5 & 6\end{matrix} \times \begin{matrix}1 & 4 \\ 2 & 5 \\ 3 & 6\end{matrix} = \begin{pmatrix}14 & \\ 32 & \end{pmatrix}$$

1*1 + 2*2 + 3*3 = 14

행: 가로 (= 옷 거는 행거로 외우면 쉬움)

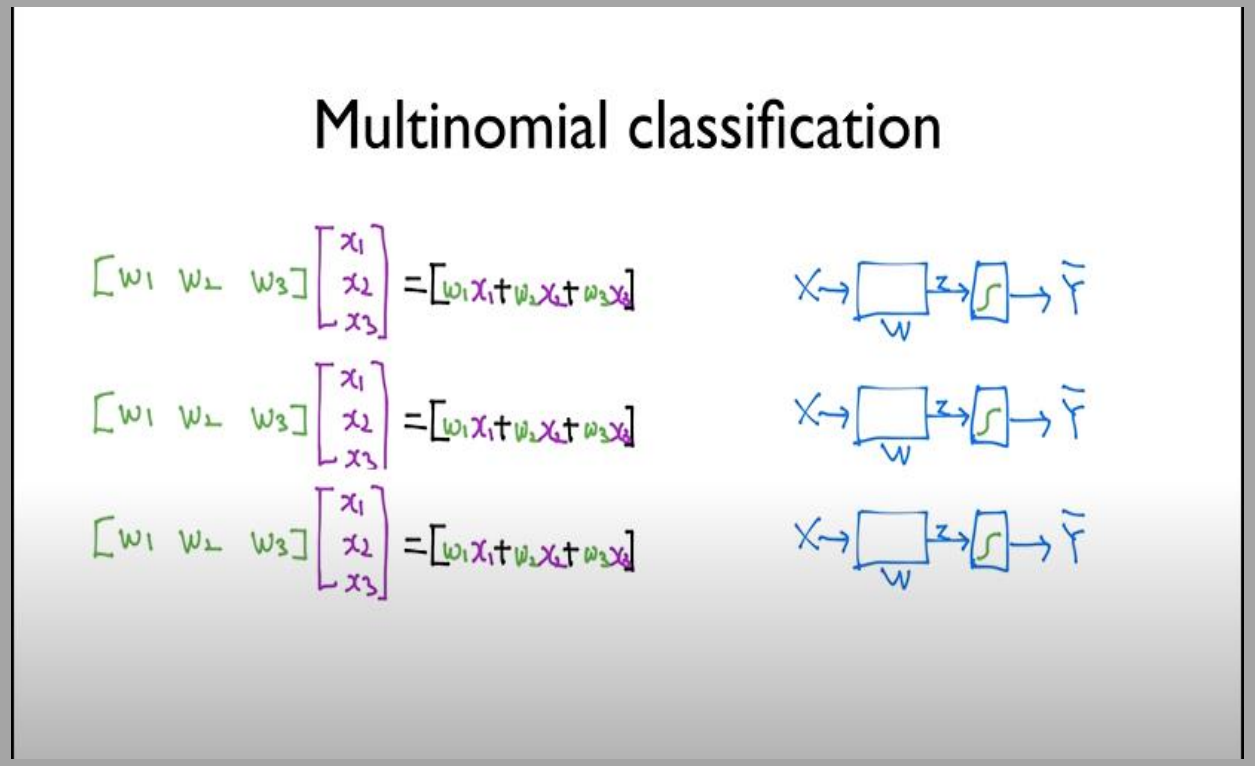
중요한 것

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 11 & 22 \\ 33 & 44 \\ 55 & 66 \end{pmatrix} = \begin{pmatrix} 11\times 1 + 33\times 2 + 55\times ? & \\ & \end{pmatrix}$$

이렇게 행렬의 곱셈은 MxN행렬과 NxR행렬 사이에서만 행렬의 곱을 할 수 있다는 것을 알 수 있다.

그래서 어떻게 3차원을 다 받나요?

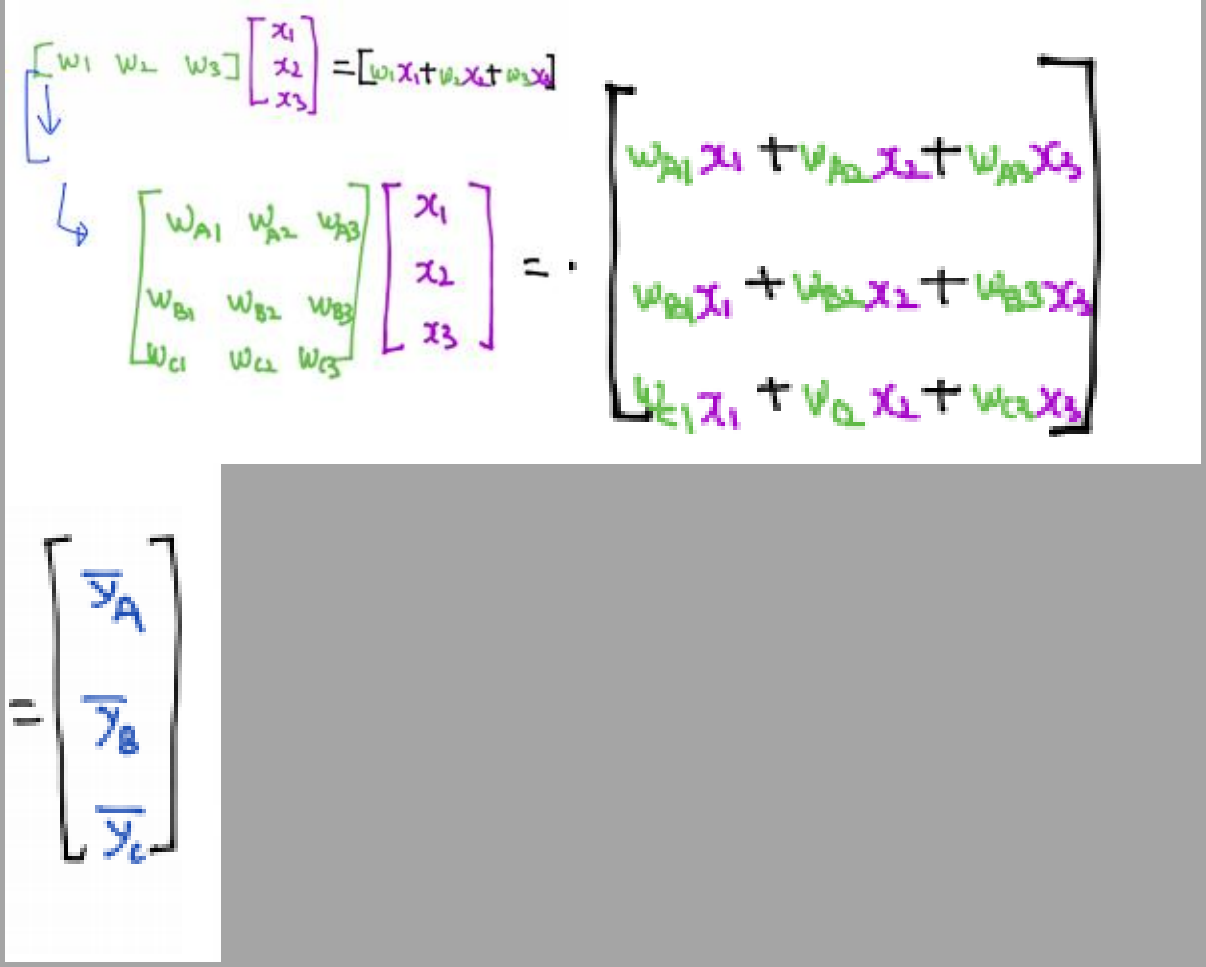
```python
def readData(filename):
    dataMatrix = np.loadtxt(filename)
    np.random.shuffle(dataMatrix)
    X = dataMatrix[:, 1:]
    y = dataMatrix[:, 0].astype(int)
    y = y.reshape((-1, 1))
    y -= 1

    return X, y
```



성킴 교수님 softmax 실습 슬라이드 발췌.

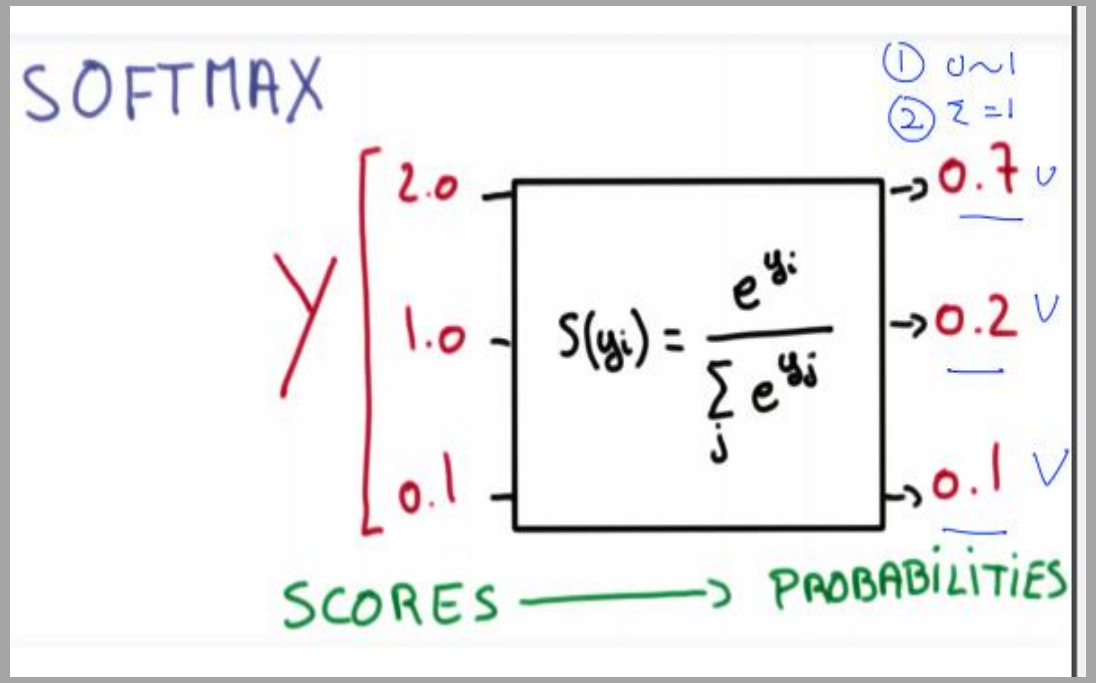
- 3차원이기 때문에, 3개의 Feature가 입력된다.
- 또한 가중치 역시 3개의 범주로 나눠야 하기 때문에, 3개가 생긴다.
- 가중치와 feature가 연산되어 나온 z가 시그모이드로 들어간다.

..



- 합성곱을 하게 될 경우,
- A,B,C에 대한 각각의 가중치가 나오게 된다.
- 이 Z_a, Z_b, Z_c의 값들을 A,B,C일 확률을 구하는 시그모이드함수에 넣으면 된다.
- 다만, 각각의 확률값을 더하는 형태이기 때문에, 정규화가 되지 않는다.

소프트맥스



- 그리하여,
  일반화 된 시그모이드
  (소프트맥스)를 사용하면
  이와 같은 문제를 해결 할 수 있다.

여기보면 분명 0~1값을 가져야 하는데 주제넘게 2가 나온 것을 볼 수 있다.

일반화 과정

```python
def softmaxEquation(self, scores):
    scores -= np.max(scores)
    prob = (np.exp(scores).T / np.sum(np.exp(scores), axis=1)).T
    return prob
```

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1 * e^{z}}{(1 + e^{-z}) * e^{z}} = \frac{e^{z}}{e^{z} + 1}$$

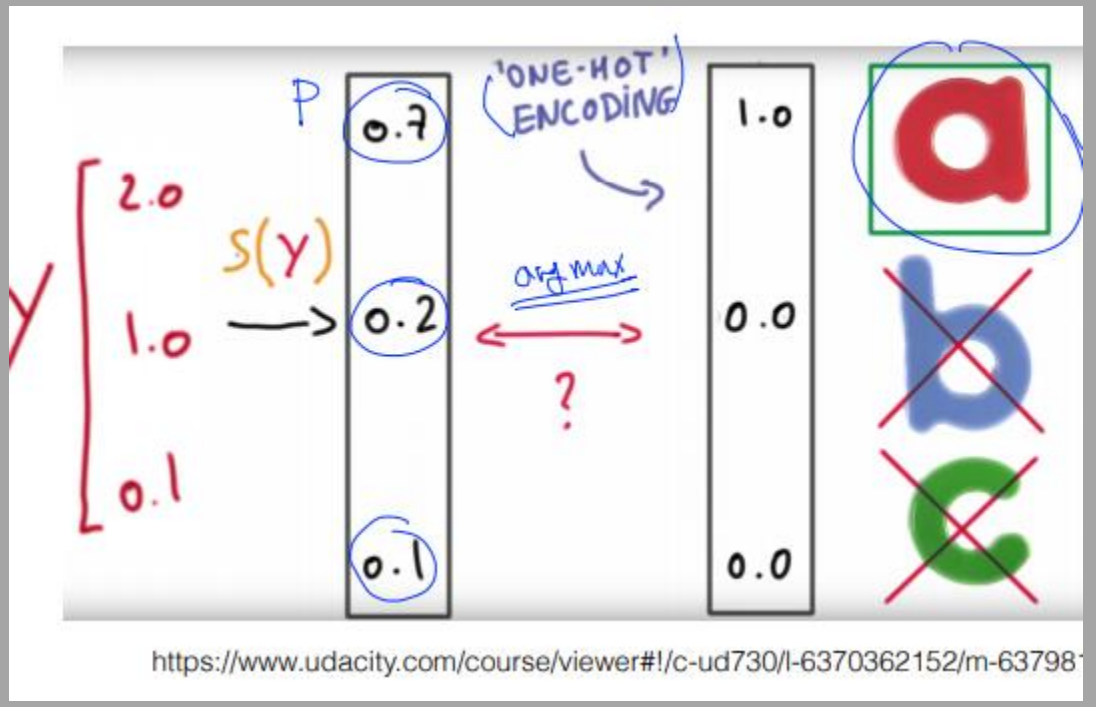
$$\sigma(z_j) = \frac{e^{zj}}{\sum_{i=1}^{K} e^{z_i}}$$

하는 이유..?

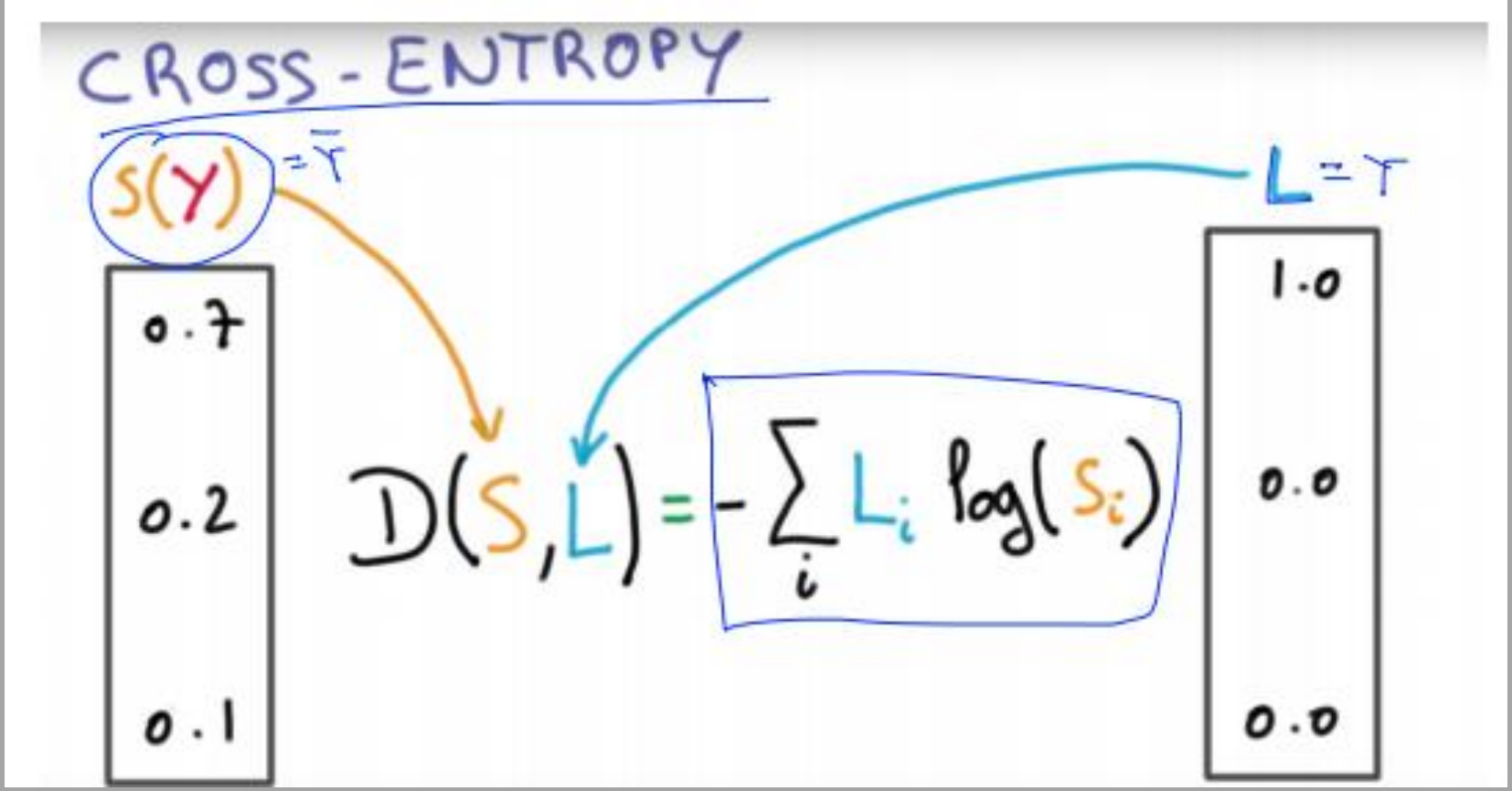
- 위에서 말했듯이, 그냥 시그모이드에 행렬곱한 벡터를 때려박으면 0과 1사이의 값이 나오지 않는다.
- 그렇게 되면, 우선 이 예측값들에 분수를 씌워서 0과 1사이로 맞춰주는 것이 선행되는데,
- 그렇게 되면 loss를 측정하기 위해 log를 씌우게 될 경우, 값이 음수로 튀는 기현상이 일어난다, 이것을 막기 위해 e^z를 곱하여 정규화한 식이 위와 같은 형태이다.

원-핫 인코딩

- 쉽게 말해 가장 확률이 높은 범주만 1로 만들고 나머지는 0으로 만들어서, 좀 더 명시적으로 결과 예측값을 만들어 내는 것이다.



Cost func



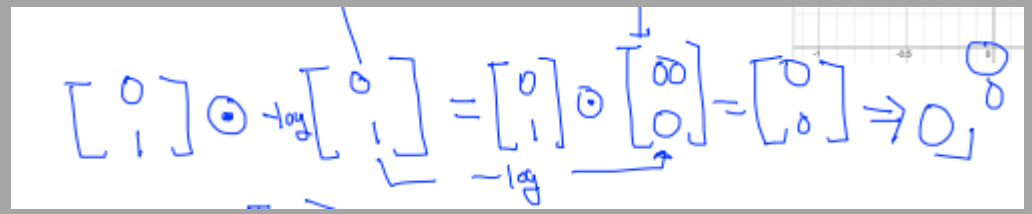
Cost func

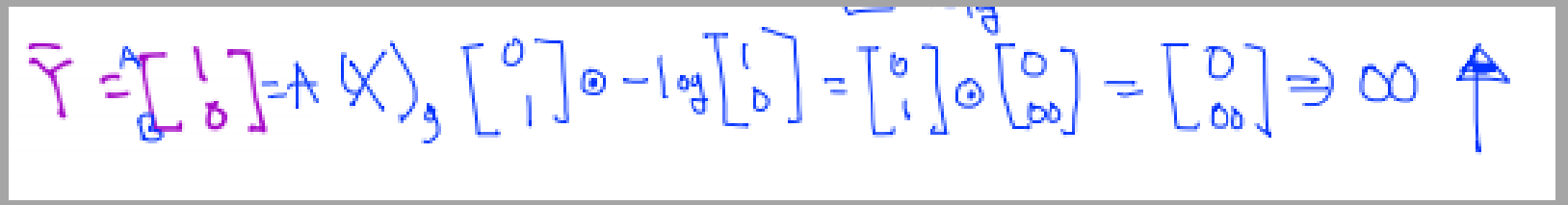
S: 확률값
L: 예측값(one-hot encoding)

```
def computeLoss(self, x, yMatrix):
    numOfSamples = x.shape[0]
    scores = np.dot(x, self.wt)
    prob = self.softmaxEquation(scores)

    loss = -np.log(np.max(prob)) * yMatrix
    regLoss = (1/2)*self.regStrength*np.sum(self.wt*self.wt)
    totalLoss = (np.sum(loss) / numOfSamples) + regLoss
    grad = ((-1 / numOfSamples) * np.dot(x.T, (yMatrix - prob))) + (self.regStrength * self.wt)
    return totalLoss, grad
```

$$D(S,L) = -\sum_i L_i \log(\overline{y}_i) = \sum_i L_i \times -\log(\overline{y}_i)$$

$$\begin{matrix} 0 \\ 1 \end{matrix} B$$





그리하여…

$$cost(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_j^{(i)} \log(p_j^{(i)})$$

```
for e in range(self.epochs): # epoch 반복
    trainLoss = self.SGDWithMomentum(xTrain, yTrainEnc)
    testLoss, dw = self.computeLoss(xTest, yTestEnc)
    trainAcc.append(self.meanAccuracy(xTrain, yTrain))
    testAcc.append(self.meanAccuracy(xTest, yTest))

    trainLosses.append(trainLoss)
    testLosses.append(testLoss)
```

softmax

- 그리하여 k개의 분류를 하여야 할 때에, k차원의 벡터를 입력하여 가중치를 구하는 모델을 만들고
- 또한 그 확률의 합을 1로 만들고자 한 것이 softmax이다.