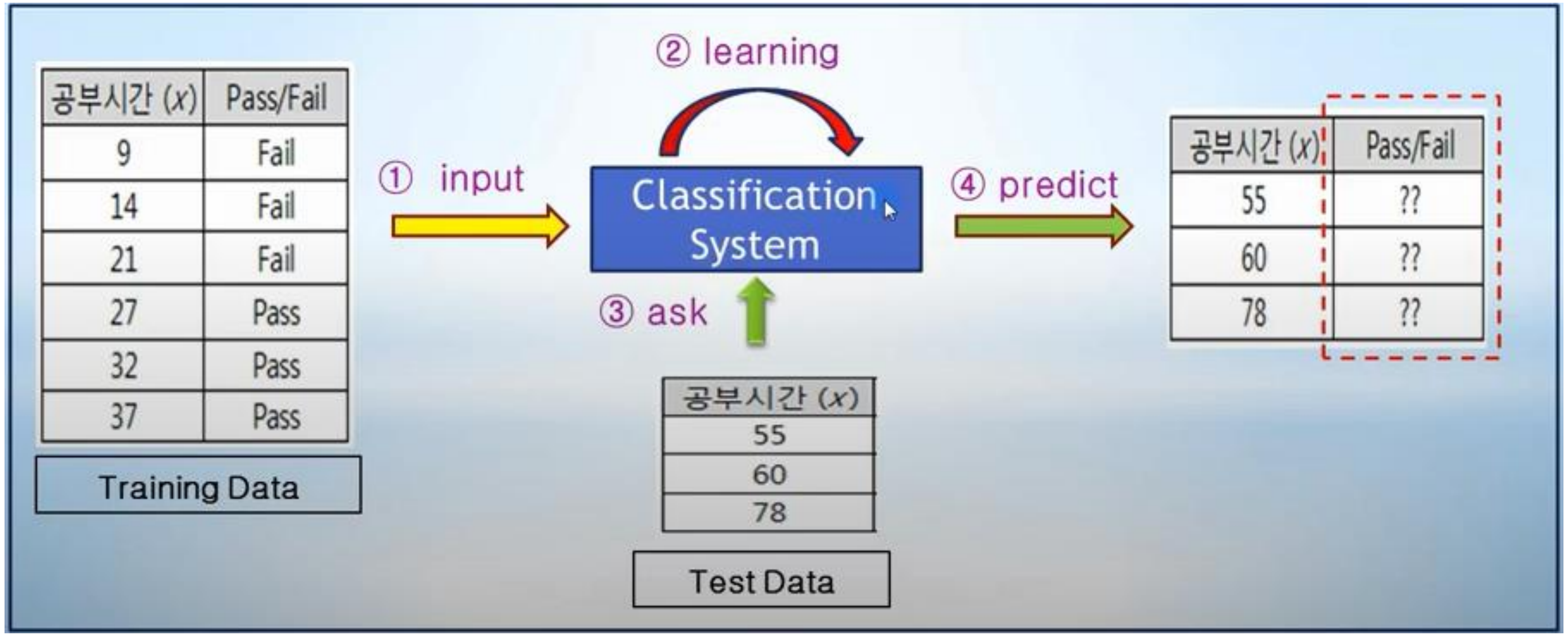
4. 로지스틱 회귀(Logistic Regression)

1. 로지스틱 회귀 **(Logistic Regression)**?
2. 로지스틱 회귀 손실 함수**()**
3. 로지스틱회귀 구현
4. 소프트맥스 회귀(Softmax Regression)

1. 로지스틱 회귀(Logistic Regression)?

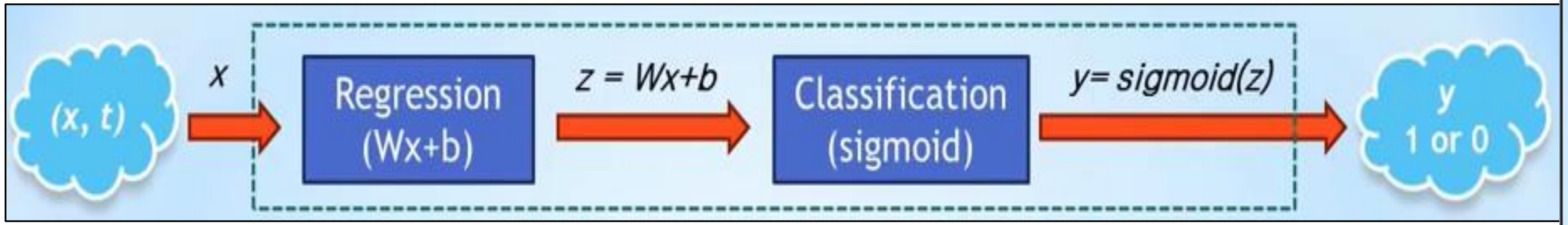
❖ 분류(Classification)

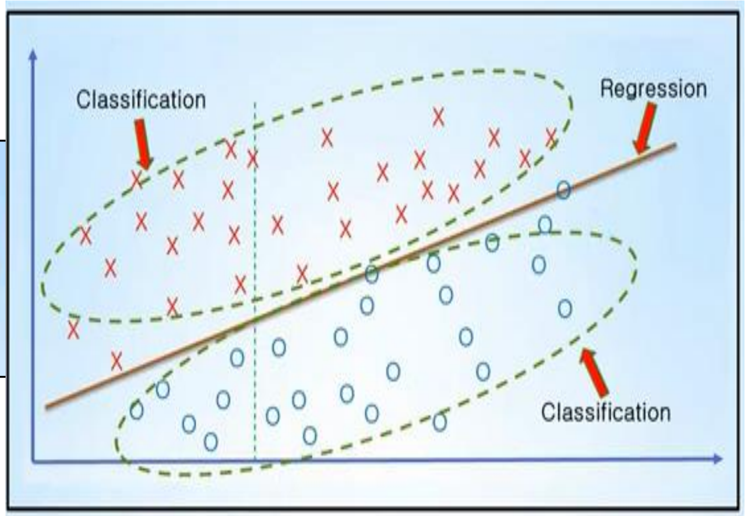
- Training Data 특성과 관계 등을 파악한 후에, 미지의 입력 데이터에 대해서 결과가 어떤 종류의 값으로 분류 될 수 있는지를 예측하는 것
- 예 : 스팸문자 분류[Spam(1) or Ham(0], 암 판별[악성종양(1) or 종양(0)]



❖ Logistic Regression algorithm Flow

- Training Data 특성과 분포를 나타내는 최적의 직선을 찾고(Linear Regression)
- 그 직선을 기준으로 데이터를 위(1) 또는 아래(0) 등으로 분류(Classification) 해주는 알고리즘
- 이러한 Logistic Regression은 Classification 알고리즘 중에서도 정확도가 높은 알고리즘으로 알려져 있어서 Deep Learning에서 기본이 되는 Componet로 사용되고 있음



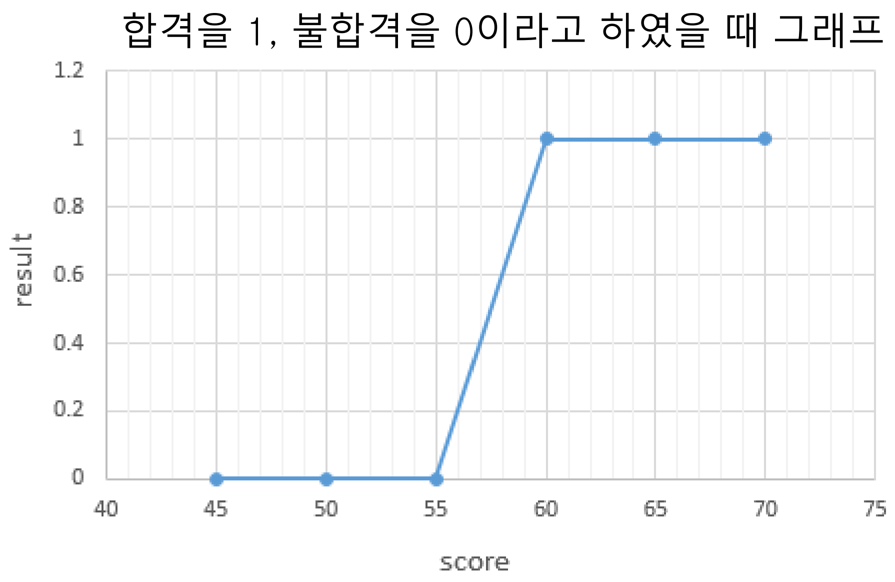


1. 로지스틱 회귀(Logistic Regression)?

❖ 이진 분류(Binary Classification)

- 시험 점수가 합격(1)인지 불합격(0)인지?
- 어떤 메일을 받았을 때 이게 정상 메일(1)인지 스팸 메일(0)인지?
- 이렇게 둘 중 하나를 결정하는 문제

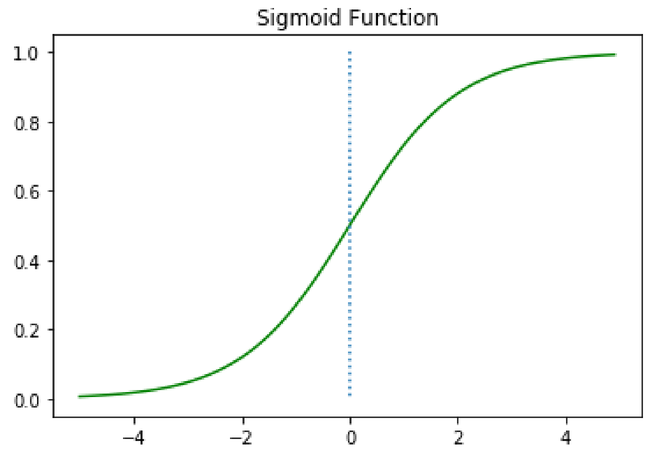
score(x)	**result(y)**
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격



1. 로지스틱 회귀(Logistic Regression)?

❖ 시그모이드 함수(Sigmoid function)

- 출력 값이 1 또는 0인 것만 가져야 하는 분류 시스템에서, 함수 값으로 0~1 값을 가지도록 하는 함수 방정식
- 시그모이드 함수 결과 값을 그래프로 표현하면 S자 형태로 나타남



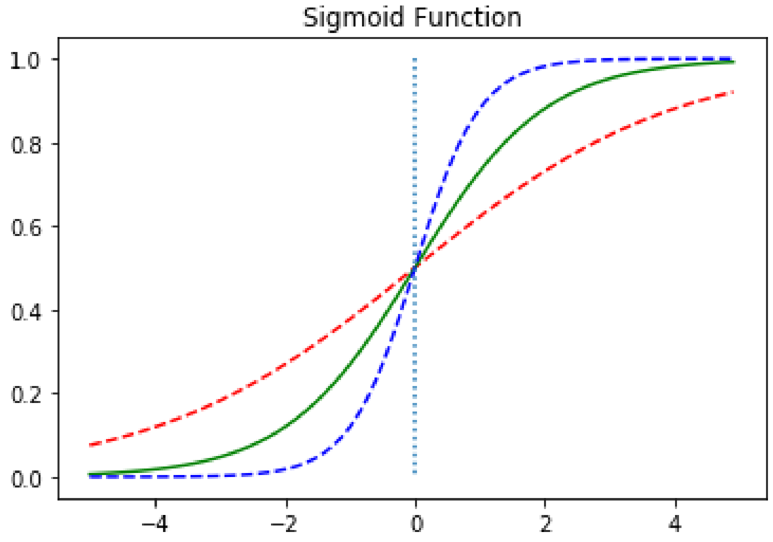
$$H(x) = sigmoid(Wx + b) = \frac{1}{1 + e^{-(Wx+b)}} = \sigma(Wx + b)$$

- 선형 회귀와 같이 로지스틱회귀 역시 최적의 W와 b를 찾는 것이 목표.
- 선형 회귀에서는 W가 직선의 기울기, b가 y절편을 의미.
- W와 b가 함수의 그래프에 어떤 영향을 주는지 직접 그래프를 그려서 확인

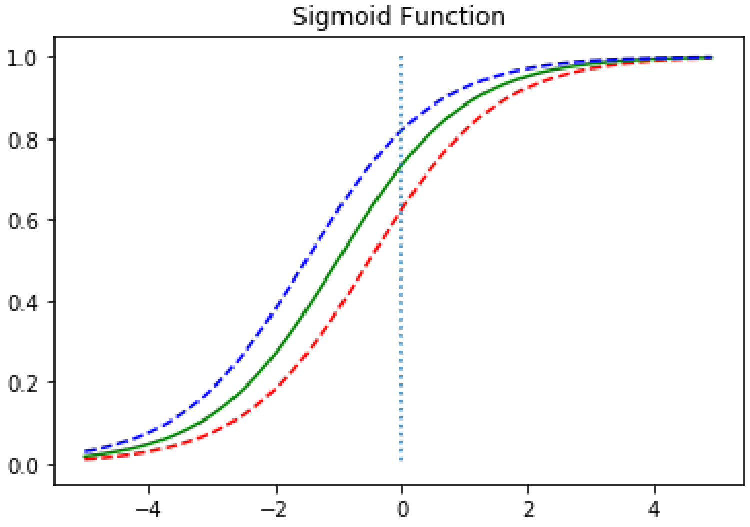
1. 로지스틱 회귀(Logistic Regression)?

❖ W, b의 변화에 따른 그래프

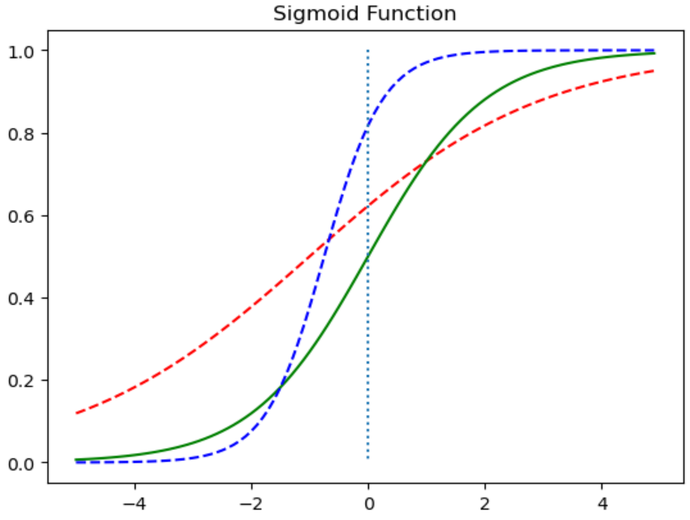
- W값의 변화에 따른 경사도의 변화
- b값의 변화에 따른 좌, 우 이동



W의 값을 변화에 따른 그래프



b의 값을 변화에 따른 그래프

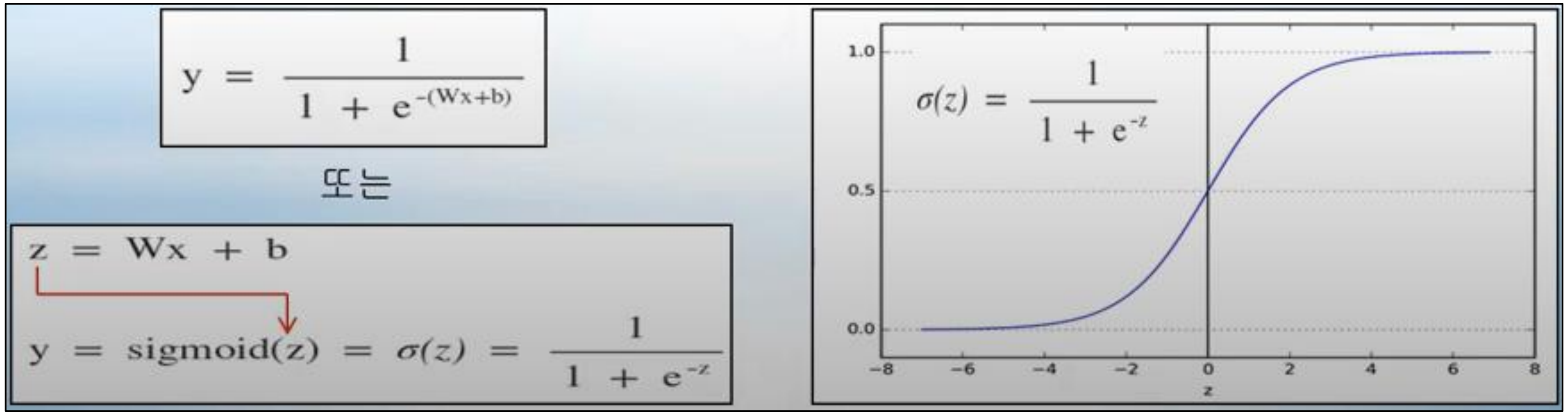


W와b의 값을 변화에 따른 그래프

1. 로지스틱 회귀(Logistic Regression)?

❖ 시그모이드 함수를 이용한 분류

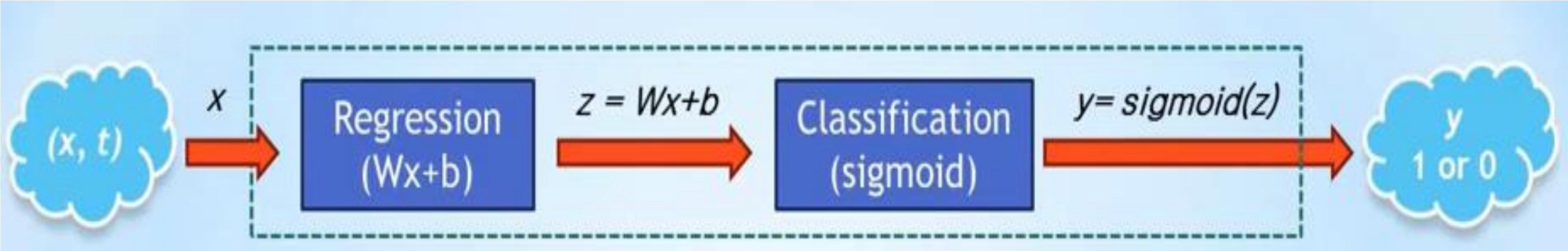
- 시그모이드 함수는 입력값이 한없이 커지면 1에 수렴하고,
- 입력값이 한없이 작아지면 0에 수렴
- **시그모이드 함수의 출력값은 0과 1 사이의 값을 가지는데** 이 특성을 이용하여 분류 작업에 사용
- 예를 들어 임계 값을 0.5라고 정하여 출력값이 0.5 이상이면 1(True), 0.5이하면 0(False)으로 판단
- 이를 확률이라고 생각하면 해당 레이블에 속할 확률이 50%가 넘으면 해당 레이블로 판단하고, 해당 레이블에 속할 확률이 50%보다 낮으면 아니라고 판단



2. 손실함수

❖ 로지스틱 회귀의 손실 함수(Cost Function), W,B

- 분류 시스템(classification) 최종 출력 값 y는 sigmoid 함수에 의해 논리적으로 1 또는 0 값을 가지기 때문에, 연속 값을 갖는 선형회귀 때와는 다른 손실함수 필요함



$$y = \frac{1}{1 + e^{-(Wx+b)}}, \quad t_i = 0 \text{ or } 1$$

손실함수
(cross-entropy)

$$E(W,b) = -\sum_{i=1}^{n} \{t_i \log y_i + (1-t_i)\log(1-y_i)\}$$

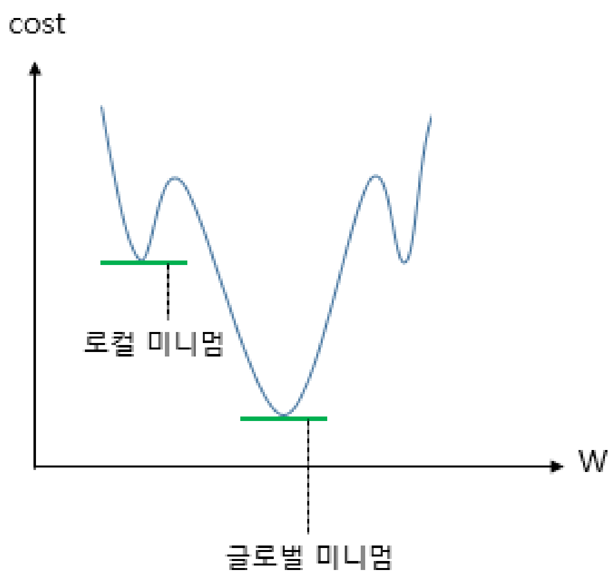
$$W = W - \alpha \frac{\partial E(W,b)}{\partial W}$$

$$b = b - \alpha \frac{\partial E(W,b)}{\partial b}$$

2.비용 함수(Cost function)

❖ 로지스틱 회귀의 손실 함수

- 선형회귀의 손실 함수 평균 제곱 오차(MSE) : $cost(W,b) = \frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} - H(x^{(i)})\right]^2$
- 로지스틱 회귀의 가설 : $H(x) = sigmoid(Wx+b)$'
- 로지스틱 회귀의 비용함수에 MSE 사용 아래 그래프와 같은 결과가 나타남

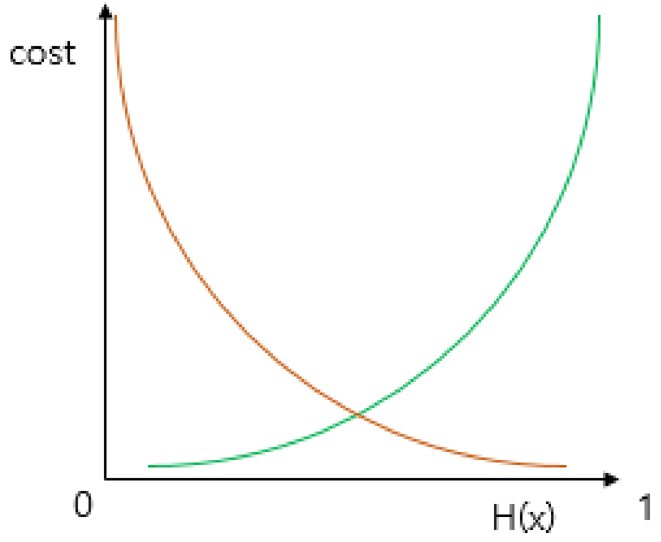


- **경사 하강법(Gradient Descent)**을 사용할 경우의 **로컬 미니멈(Local Minimum)**에 도달 문제가 발생
- 즉, 경사 하강법이 오차가 최소값이 되는 구간에 도착했다고 판단한 그 구간이 **글로벌 미니멈(Global Minimum)** 구간이 아닐 수 있음
- 실제 최소가 되는 구간을 잘못 판단하면 최적의 가중치 w가 아닌 값을 택해 모델의 성능이 더 오르지 않을 수 있음.

2.비용 함수(Cost function)

❖ 로지스틱 회귀의 손실 함수

- 시그모이드 함수의 특징은 함수의 출력값이 0과 1사이의 값이라는 점
- 즉, 실제 값이 1일 때 예측 값이 0에 가까워지면 오차가 커져야 하며,
- 실제 값이 0일 때, 예측 값이 1에 가까워지면 오차가 커져야 함.
- 이를 충족하는 로그 함수 사용.
  - y=0.5에 대칭하는 두 개의 로그 함수 그래프



- 실제 값이 1일 때의 그래프를 주황색 선으로 표현
- 실제 값이 0일 때의 그래프를 초록색 선으로 표현
- 실제 값= 1일 경우: 예측 값인 H(x)의 값이 1이면 오차가 0이므로 cost는 0, H(x)가 0으로 수렴하면 cost는 무한대로 발산함.
- 실제값= 0인 경우: 예측 값인 H(x)의 값이 0이면 오차가 0이므로 cost는 0, H(x)가 1으로 수렴하면 cost는 무한대로 발산함.

2.비용 함수(Cost function)

❖ 로지스틱 회귀의 손실 함수

- 오차 값에 대한 로그 식

$$\text{if } y = 1 \rightarrow \text{cost}\,(H(x), y) = -\log(H(x))$$

$$\text{if } y = 0 \rightarrow \text{cost}\,(H(x), y) = -\log(1 - H(x))$$

- 위의 두식을 결합 :

$$\text{cost}\,(H(x), y) = -[ylogH(x) + (1 - y)log(1 - H(x))]$$

- 모든 오차의 평균식 :

$$cost(W) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}logH(x^{(i)}) + (1 - y^{(i)})log(1 - H(x^{(i)}))]$$

- 최적의 가중치 W 업데이트 식 :

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

3. 로지스틱 회귀 구현-1

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)

x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]

x_train = torch.FloatTensor(x_data)
y_train = torch.FloatTensor(y_data)

print(x_train.shape)
print(y_train.shape)

W = torch.zeros((2, 1), requires_grad=True) # 크기는 2 x 1
b = torch.zeros(1, requires_grad=True)
```

```
# optimizer 설정
optimizer = optim.SGD([W, b], lr=1)

nb_epochs = 1000
costs=[]
for epoch in range(nb_epochs + 1):

    # Cost 계산
    hypothesis = torch.sigmoid(x_train.matmul(W) + b)
    cost = -(y_train * torch.log(hypothesis) +
             (1 - y_train) * torch.log(1 - hypothesis)).mean()

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 100번마다 로그 출력
    costs.append(cost.item())
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()
        ))
```

3. 로지스틱 회귀 구현-2. nn.Modual 사용

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]
x_train = torch.FloatTensor(x_data)
y_train = torch.FloatTensor(y_data)

model = nn.Sequential(
  nn.Linear(2, 1), # input_dim = 2, output_dim = 1
  nn.Sigmoid() # 출력은 시그모이드 함수를 거침
)

model(x_train)

# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=1)

nb_epochs = 1000
```

```
for epoch in range(nb_epochs + 1):
  # H(x) 계산
  hypothesis = model(x_train)

  # cost 계산
  cost = F.binary_cross_entropy(hypothesis, y_train)

  # cost로 H(x) 개선
  optimizer.zero_grad()
  cost.backward()
  optimizer.step()

  # 20번마다 로그 출력
  if epoch % 10 == 0:
    prediction = hypothesis >= torch.FloatTensor([0.5]) # 예측값이 0.5를 넘으면 True로 간주
    correct_prediction = prediction.float() == y_train # 실제값과 일치하는 경우만 True로 간주
    accuracy = correct_prediction.sum().item() / len(correct_prediction) # 정확도를 계산
    print('Epoch {:4d}/{} Cost: {:.6f} Accuracy {:2.2f}%'.format( # 각 에포크마다 정확도를 출력
      epoch, nb_epochs, cost.item(), accuracy * 100,
    ))
```

3. 로지스틱 회귀 구현-3. 클래로 파이토치 모델 구현

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)

x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]
x_train = torch.FloatTensor(x_data)
y_train = torch.FloatTensor(y_data)

class BinaryClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        return self.sigmoid(self.linear(x))

model = BinaryClassifier()
```

```
# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=1)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = model(x_train)

    # cost 계산
    cost = F.binary_cross_entropy(hypothesis, y_train)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 20번마다 로그 출력
    if epoch % 10 == 0:
        prediction = hypothesis >= torch.FloatTensor([0.5]) # 예측값이 0.5를 넘으면 True로 간주
        correct_prediction = prediction.float() == y_train # 실제값과 일치하는 경우만 True로 간주
        accuracy = correct_prediction.sum().item() / len(correct_prediction) # 정확도를 계산
        print('Epoch {:4d}/{} Cost: {:.6f} Accuracy {:2.2f}%'.format( # 각 에포크마다 정확도를 출력
            epoch, nb_epochs, cost.item(), accuracy * 100,
        ))
```

4. 소프트맥스 회귀(Softmax Regression)

❖ 소프트맥스 회귀(Softmax Regression)

- 3개 이상의 선택지로부터 1개를 선택하는 문제인 다중 클래스 분류(Multi-Class classification)를 풀기 위한 방법

❖ 원-핫 인코딩(One-hot encoding)

- 범주형 데이터를 처리할 때 레이블을 표현하는 방법
- 선택해야 하는 선택지의 개수만큼의 차원을 가지면서, 각 선택지의 인덱스에 해당하는 원소에는 1, 나머지 원소는 0의 값을 가지도록 하는 표현 방법
- 예 : 강아지는 0번 인덱스, 고양이는 1번 인덱스, 냉장고는 2번 인덱스를 부여

```
강아지 = [1, 0, 0]
고양이 = [0, 1, 0]
냉장고 = [0, 0, 1]
```

- 원-핫 인코딩으로 표현된 벡터를 **원-핫 벡터(one-hot vector)**

4. 소프트맥스 회귀(Softmax Regression)

❖ 원-핫 벡터의 무작위성

- 다중 클래스 분류 문제가 각 클래스 간의 관계가 균등하다는 점에서 원-핫 벡터는 이러한 점을 적절한 표현
- Banana(1), Tomato(2), Apple(3)라는 3개의 클래스가 존재하는 문제에 숫자 레이블을 사용할 경우

$$Loss\ function = \frac{1}{n}\sum_{i}^{n}\left(y_i - \hat{y_i}\right)^2$$

  - Tomato가 Banana로 인식된 경우 : $(2-1)^2 = 1$
  - Apple이 Banana로 인식된 경우 : $(3-1)^2 = 4$

Banan가 Apple보다 Tomato에 가깝다는 정보를 나타냄
=> 의미 있는 정보인가?

- 원-핫 벡터 사용인 경우

$$((1,0,0)-(0,1,0))^2 = (1-0)^2 + (0-1)^2 + (0-0)^2 = 2$$

$$((1,0,0)-(0,0,1))^2 = (1-0)^2 + (0-0)^2 + (0-1)^2 = 2$$

원-핫 벡터는 각 클래스 차이 균등함
=> 원-핫 벡터의 무작위성

4. 소프트맥스 회귀(Softmax Regression)

❖ 다중 클래스 분류(Multi-class Classification)

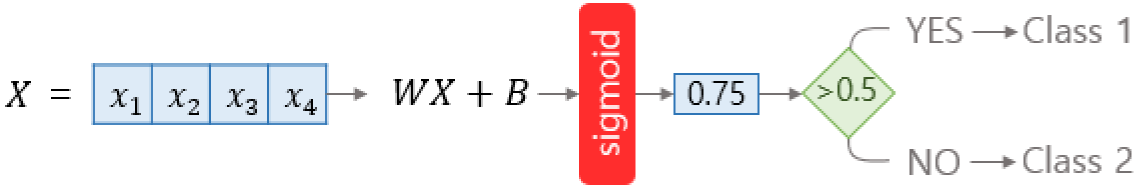
- 로지스틱 회귀 : 2개의 선택지 중에서 1개를 고르는 이진 분류(Binary Classification)에 사용
- 소프트맥스 회귀: 3개 이상의 선택지 중에서 1개를 고르는 다중 클래스 분류(Multi-Class Classification)에 사용
- 예: 꽃받침 길이, 꽃받침 넓이, 꽃잎 길이, 꽃잎 넓이라는 4개의 특성(feature)로부터 setosa, versicolor, virginica라는 3개의 붓꽃 품종 중 어떤 품종인지를 예측하는 문제로 전형적인 다중 클래스 분류 문제

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	PetalWidthCm(x_4)	Species(y)
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
5.8	2.6	4.0	1.2	versicolor
6.7	3.0	5.2	2.3	virginica
5.6	2.8	4.9	2.0	virginica

4. 소프트맥스 회귀(Softmax Regression)

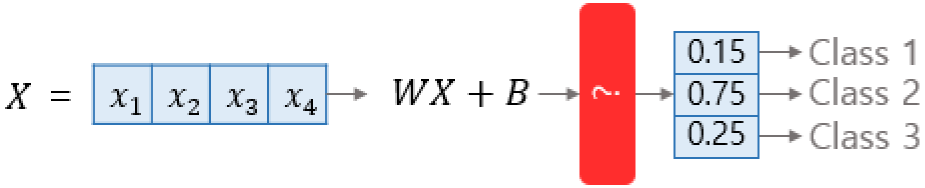
❖ 다중 클래스 분류(Multi-class Classification)

- 로지스틱 회귀



가설 : $H(X) = sigmoid(WX + B)$

- 소프트맥스 회귀



가설 : $H(X) = softmax(WX + B)$

4. 소프트맥스 회귀(Softmax Regression)

❖ 소프트맥스 함수(Softmax function)

▪ 소프트맥스 함수의 이해

- k차원의 벡터에서 i번째 원소를 P_i, i번째 클래스가 정답일 확률을 P_i로 일 때 소프트맥스 함수는 P_i를 다음과 같이 정의

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad for\ i = 1, 2, \ldots k$$

- k=3이므로 3차원 벡터 $z = [z_{1,} z_{2,}, z_{3,}]$의 입력을 받으면 소프트맥스 함수는 아래와 같은 출력을 리턴

$$softmax(z) = [\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}] = [p_1, p_2, p_3] = \hat{y} = \text{예측값}$$

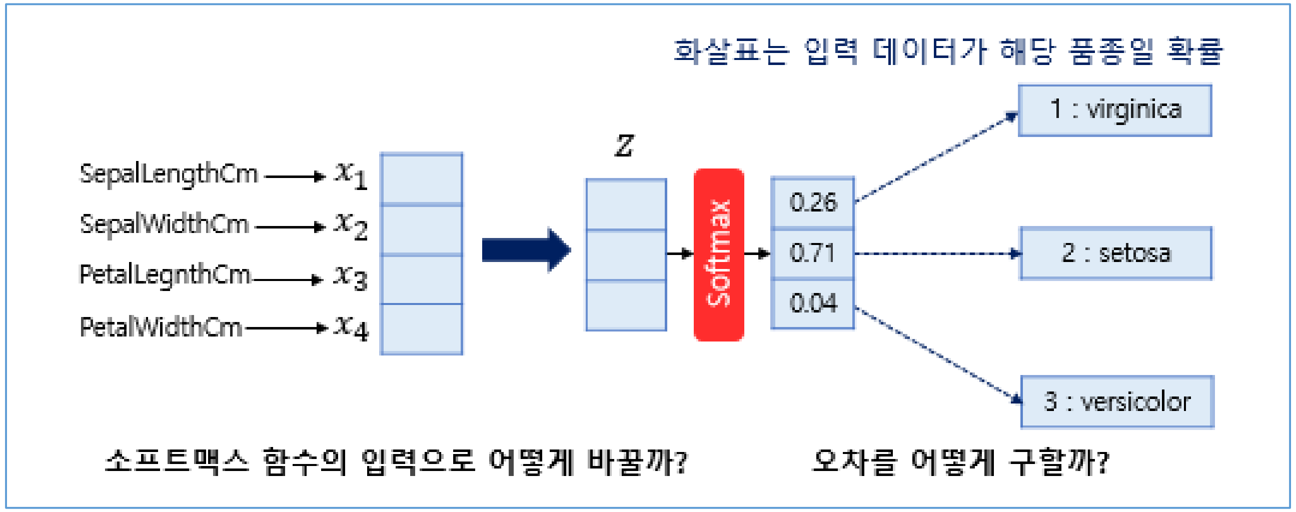
- p_1, p_2, p_3 각각은 1번 클래스가 정답일 확률, 2번 클래스가 정답일 확률, 3번 클래스가 정답일 확률을 나타내며, 각각 0과 1사이의 값으로 총 합은 1이 됨

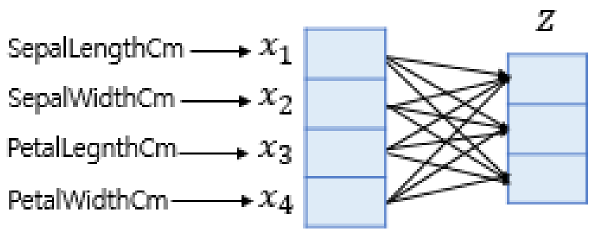
$$softmax(z) = [\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}] = [p_1, p_2, p_3] = [p_{virginica}, p_{setosa}, p_{versicolor}]$$

4. 소프트맥스 회귀(Softmax Regression)

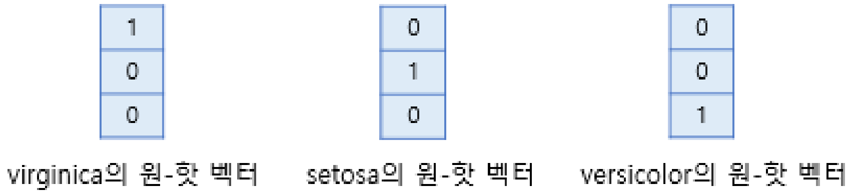
❖ 소프트맥스 함수(Softmax function)

- 그림을 통한 이해





4차원 벡터를 입력으로 3차원 벡터로 변환

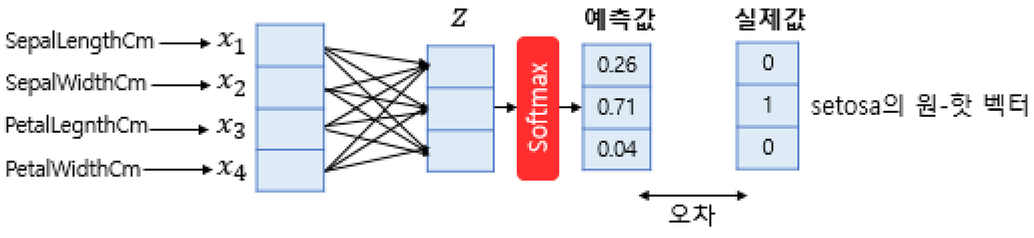


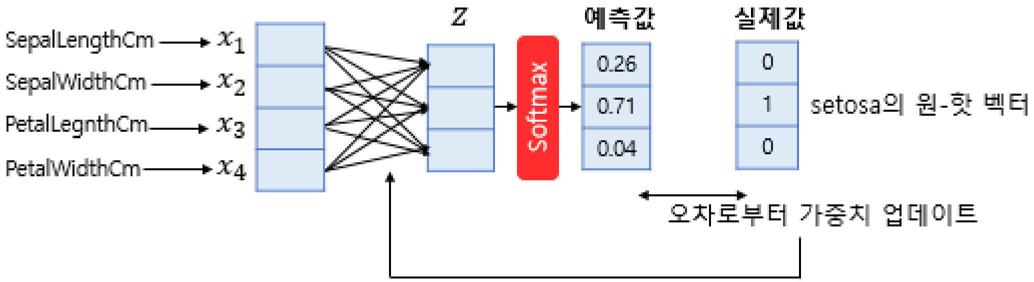
실제값의 정수 인코딩은 1, 2, 3을 원-핫 인코딩 수행결과

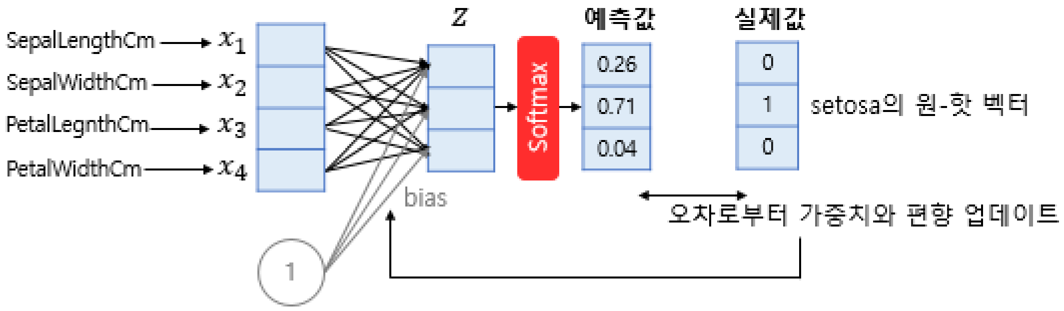
4. 소프트맥스 회귀(Softmax Regression)

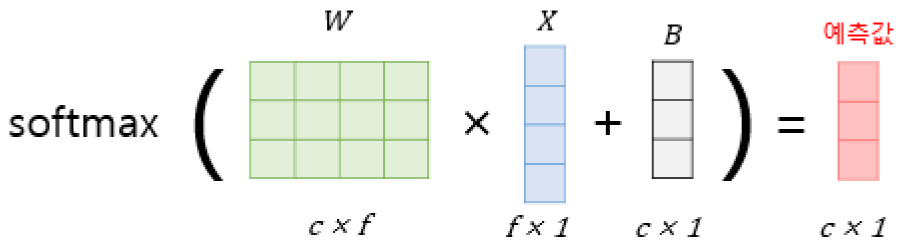
❖ 소프트맥스 함수(Softmax function)

- 그림을 통한 이해









여기서 f는 특성의 수이며 c는 클래스의 개수에 해당됩니다.

[소프트맥스 회귀에서 예측값을 구하는 과정을 벡터와 행렬 연산으로 표현]

4. 소프트맥스 회귀(Softmax Regression)

❖ 붓꽃 품종 분류하기 행렬 연산으로 이해하기

- 전체 샘플의 개수가 5개, 특성이 4개, 선택지(정답) 3개

$$X = \begin{pmatrix} 5.1 & 3.5 & 1.4 & 0.2 \\ 4.9 & 3.0 & 1.4 & 0.2 \\ 5.8 & 2.6 & 4.0 & 1.2 \\ 6.7 & 3.0 & 5.2 & 2.3 \\ 5.6 & 2.8 & 4.9 & 2.0 \end{pmatrix} \Rightarrow X = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \\ x_{51} & x_{52} & x_{53} & x_{54} \end{pmatrix}$$

입력데이터행렬(5X4)

$$\hat{Y} = \begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \\ y_{41} & y_{42} & y_{43} \\ y_{51} & y_{52} & y_{53} \end{pmatrix}$$

예측 값 행렬(5X3)

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix}$$

가중치(W) 행렬(5X3)

$$B = \begin{pmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \end{pmatrix}$$

편향(b) 행렬(5X3)

$$\hat{Y} = softmax(XW + B)$$

$$\begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \\ y_{41} & y_{42} & y_{43} \\ y_{51} & y_{52} & y_{53} \end{pmatrix} = softmax\left(\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \\ x_{51} & x_{52} & x_{53} & x_{54} \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} + \begin{pmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \end{pmatrix}\right)$$

4. 소프트맥스 회귀(Softmax Regression)

❖ 비용함수(Cost function)

- 크로스 엔트로피 함수

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

y는 실제값을 나타내며, k는 클래스의 개수. yi는 실제값 원-핫 벡터의 i번째 인덱스를 의미하며, pj는 샘플 데이터가 j번째 클래스일 확률을 나타냄

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

n개 전체 데이터에 대한 평균을 구한 최종 비용함수

- **이진 분류에서의 크로스 엔트로피 함수**

$$cost(W) = -(y \, logH(X) + (1-y) \, log(1 - H(X)))$$

$$cost(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_j^{(i)} \, log(p_j^{(i)}) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}log(p^{(i)}) + (1-y^{(i)})log(1-p^{(i)})]$$

4. 소프트맥스 회귀(Softmax Regression)

❖ 비용함수 구현- 로우-레벨

```
import torch
import torch.nn.functional as F

torch.manual_seed(1)

z = torch.FloatTensor([1, 2, 3])
hypothesis = F.softmax(z, dim=0)
print(hypothesis)

hypothesis.sum()

z = torch.rand(3, 5, requires_grad=True)
hypothesis = F.softmax(z, dim=1)
print(hypothesis)

y = torch.randint(5, (3,)).long()
print(y)
```

```
# 모든 원소가 0의 값을 가진 3 × 5 텐서 생성
y_one_hot = torch.zeros_like(hypothesis)
y_one_hot.scatter_(1, y.unsqueeze(1), 1)

print(y.unsqueeze(1))

print(y_one_hot)

cost = (y_one_hot * -torch.log(hypothesis)).sum(dim=1).mean()
print(cost)
```

4. 소프트맥스 회귀(Softmax Regression)

❖ 비용함수 구현- (하이_레벨)

```
# Low level
torch.log(F.softmax(z, dim=1))

# High level
F.log_softmax(z, dim=1)

# Low level
# 첫번째 수식
(y_one_hot * -torch.log(F.softmax(z, dim=1))).sum(dim=1).mean()

# 두번째 수식
(y_one_hot * - F.log_softmax(z, dim=1)).sum(dim=1).mean()
```

```
# High level
# 세번째 수식
F.nll_loss(F.log_softmax(z, dim=1), y)

# 네번째 수식
F.cross_entropy(z, y)
```

소프트맥스 회귀 구현하기
MNIST 데이터 분류하기 실습