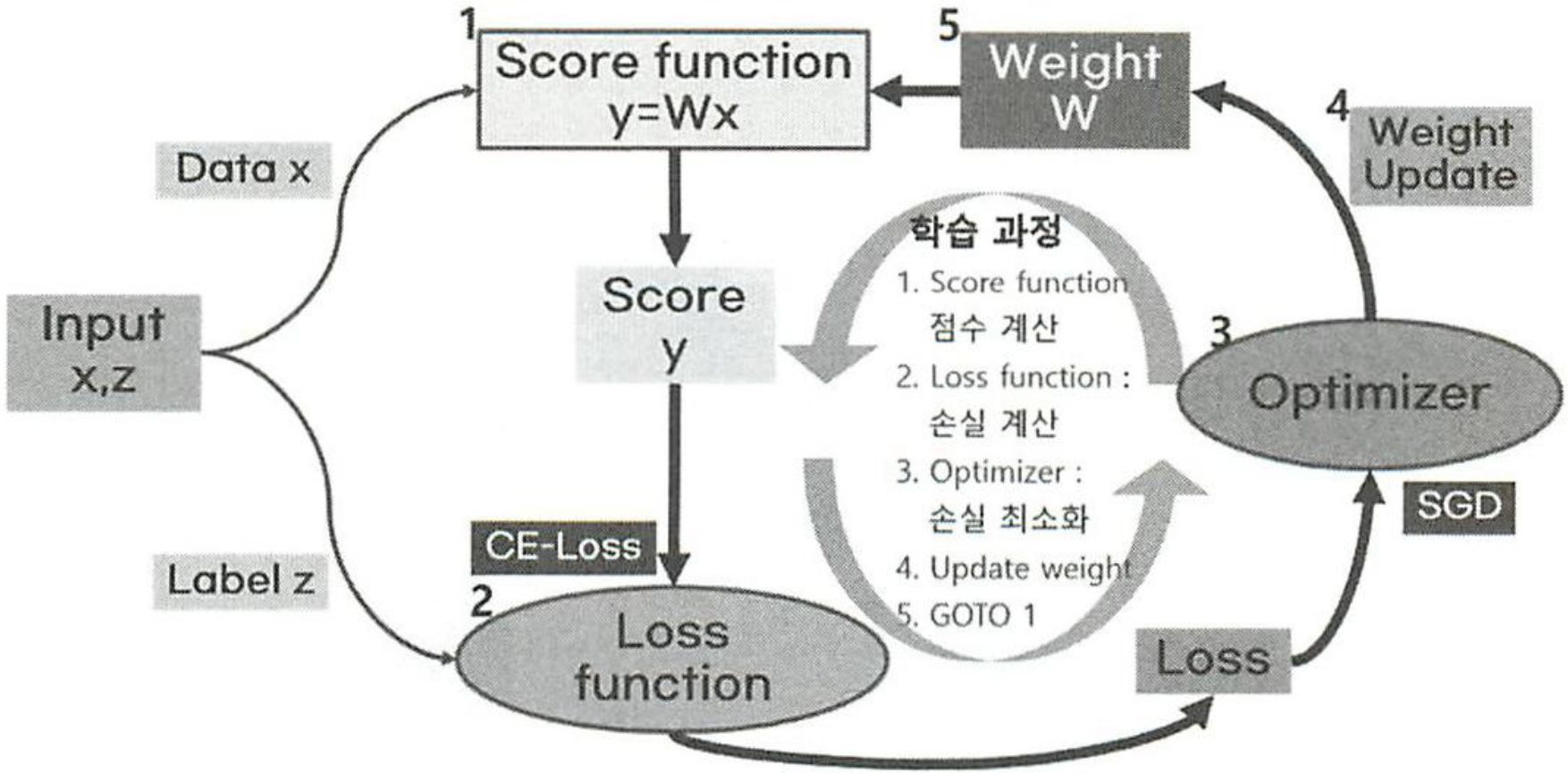
선형모델

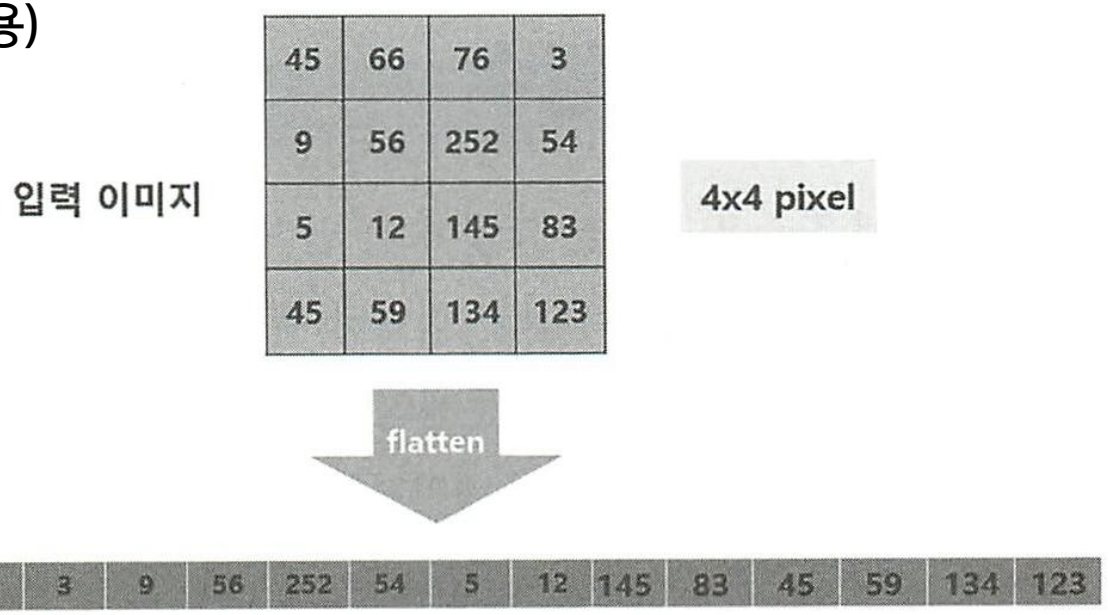
선형모델의 학습 과정


1
Score function
y=Wx
5
Weight
W
4
Weight
Update
Data x
Input
x,z
Score
y
학습 과정
1. Score function
점수 계산
2. Loss function :
손실 계산
3. Optimizer :
손실 최소화
4. Update weight
5. GOTO 1
3
Optimizer
SGD
CE-Loss
Label z
2
Loss
function
Loss

벡터화

- 선형모델에서 입력 데이터는 벡터 형태로 처리
- 2차원 또는 3차원 이미지 데이터를 **1차원 벡터**로 변환하는 과정
- 선형 모델에서는 입력이 반드시 **1차원 벡터**이어야 하므로, 벡터화 필수

  (예: `4×4` 픽셀 이미지를 하나의 벡터로 변환하여 사용)



벡터화 코드

- 이미지를 벡터화할 때, numpy를 사용하는 경우 flatten 또는 reshape을 사용해 벡터화 할 수 있다.

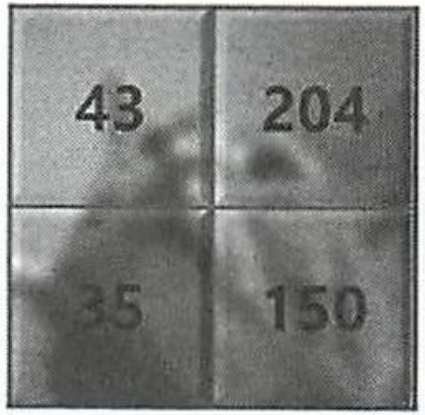
```
# random 함수로 0~255 사이의 임의의 정수를 성분으로 갖는 4×4 행렬을 만든다.
a = np.random.randint(0, 255, (4, 4))
a


# flatten을 사용해 1차원 행렬(벡터)로 만든다.
b = a.flatten()
b
```

```
# reshape을 사용해 행렬 크기를 바꾼다. -1은 자동으로 계산한다는 의미이고 이 경우 16을 적
는 것과 같다. 만약 (2, 8)의 행렬로 바꾸려 한다면 reshape(2, -1) 또는 reshape(2, 8)
둘 다 같은 결과이다.
c = a.reshape(-1)
c
```

선형 분류기 - Score 함수

$$f(x, W) = Wx + b = y$$



Input image

Class: cat

z: Label

flatten

score function

x → **f** → $Wx + b$

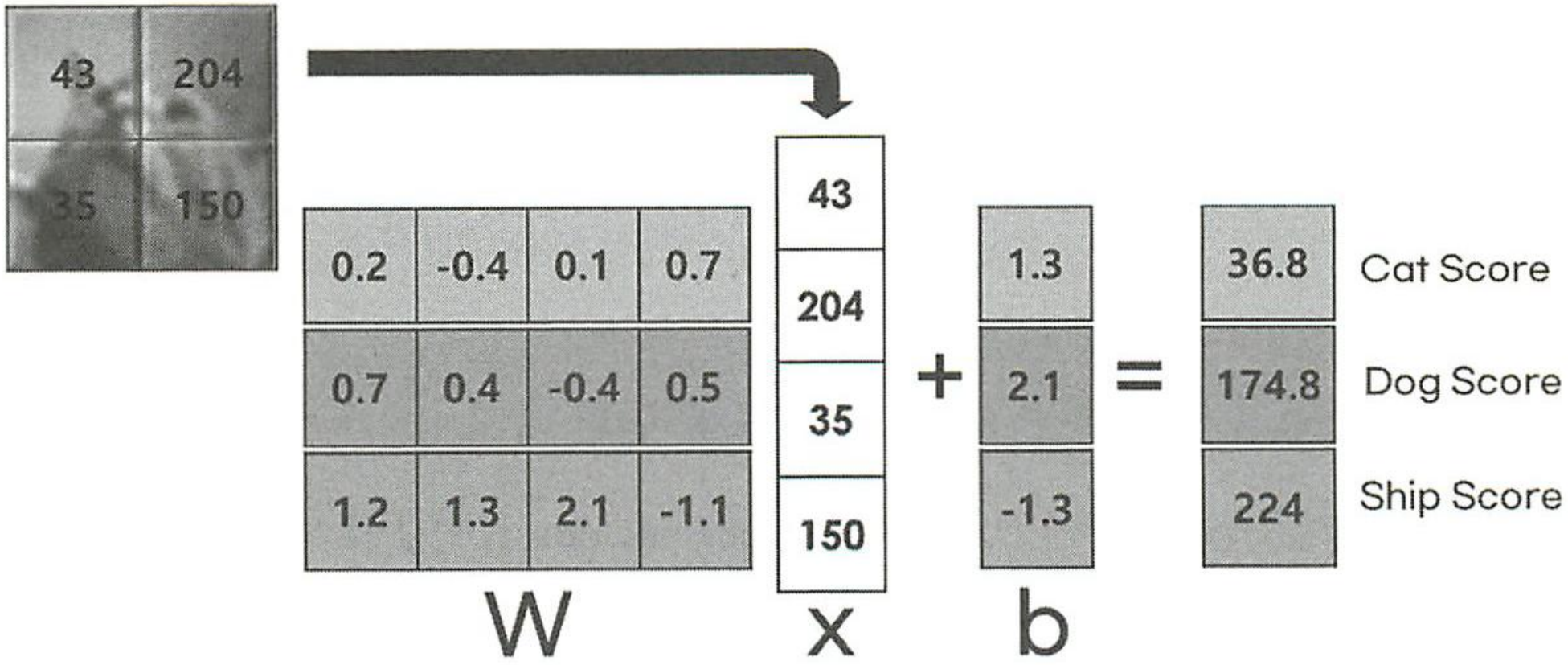
score

36.8

W: weight

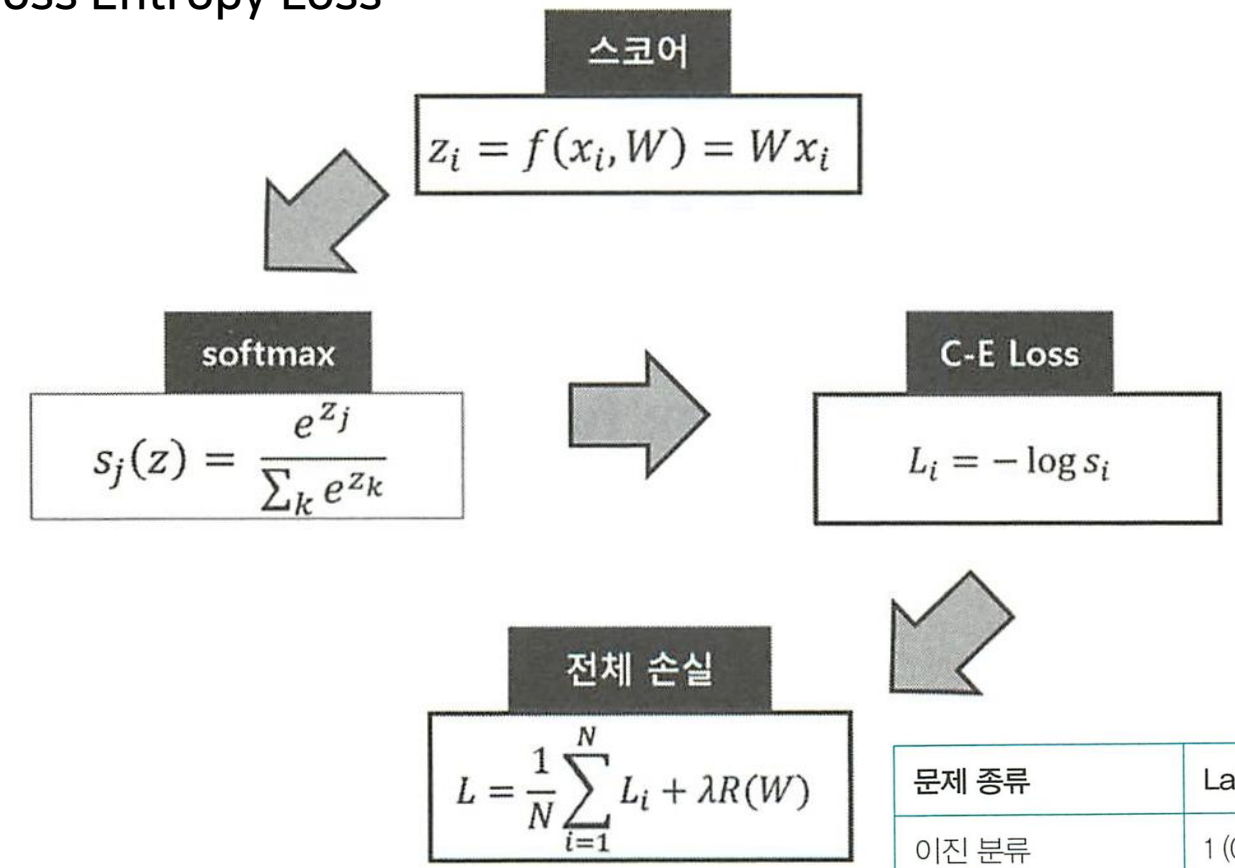
b: bias

선형 분류기 - Score 함수의 병렬처리



Sofemax 분류기

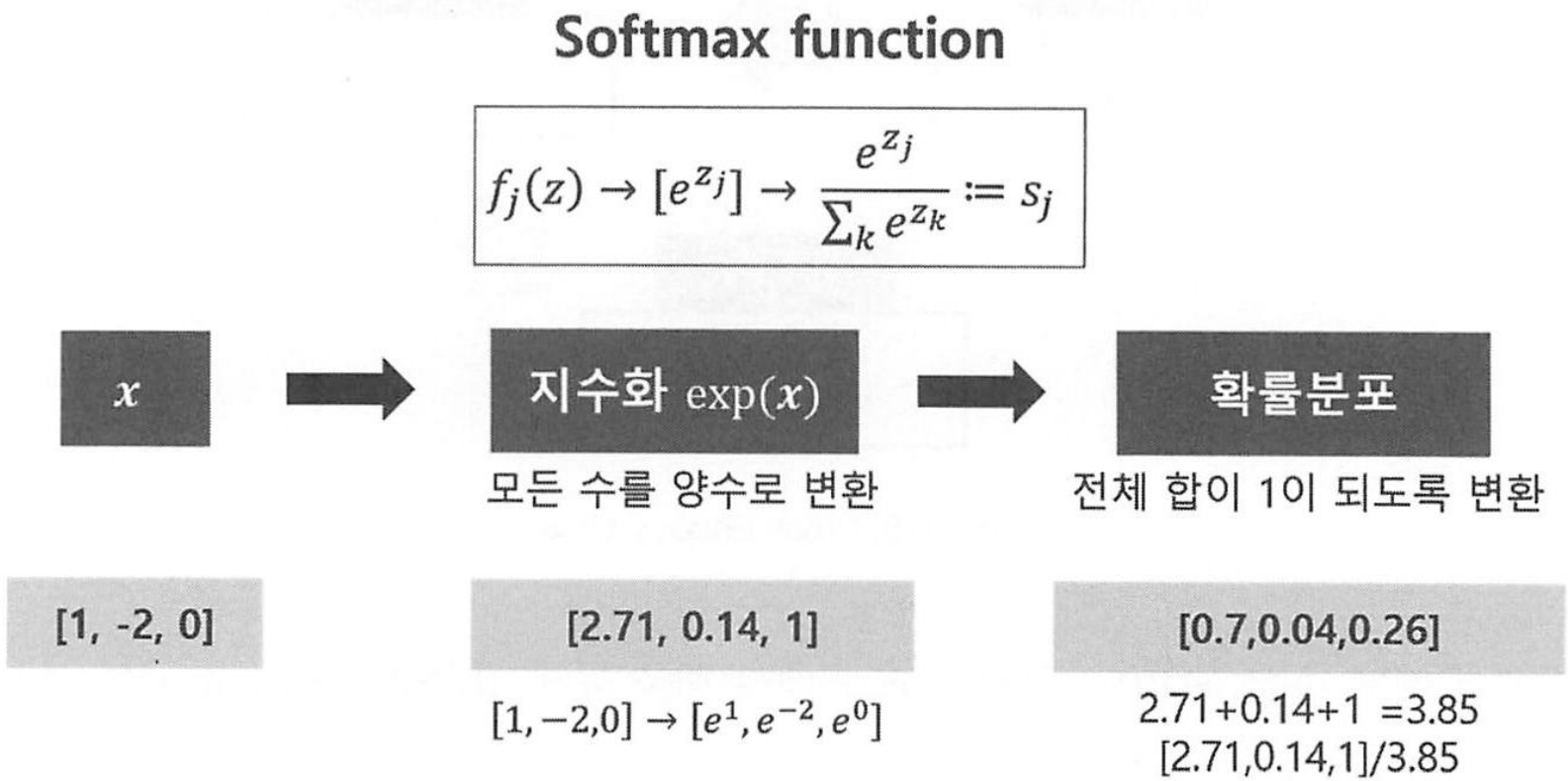
- Cross Entropy Loss



문제 종류	Label 차원(형태)	activation	Loss 함수
이진 분류	1 (0 또는 1)	sigmoid	binary_crossentropy
다중 분류	n (One-hot 벡터)	softmax	categorical_crossentropy
다중 분류	n (숫자 0, 1, 2, …)	softmax	sparse_categorical_crossentropy
회귀	1 (실수)	–	mse / mae

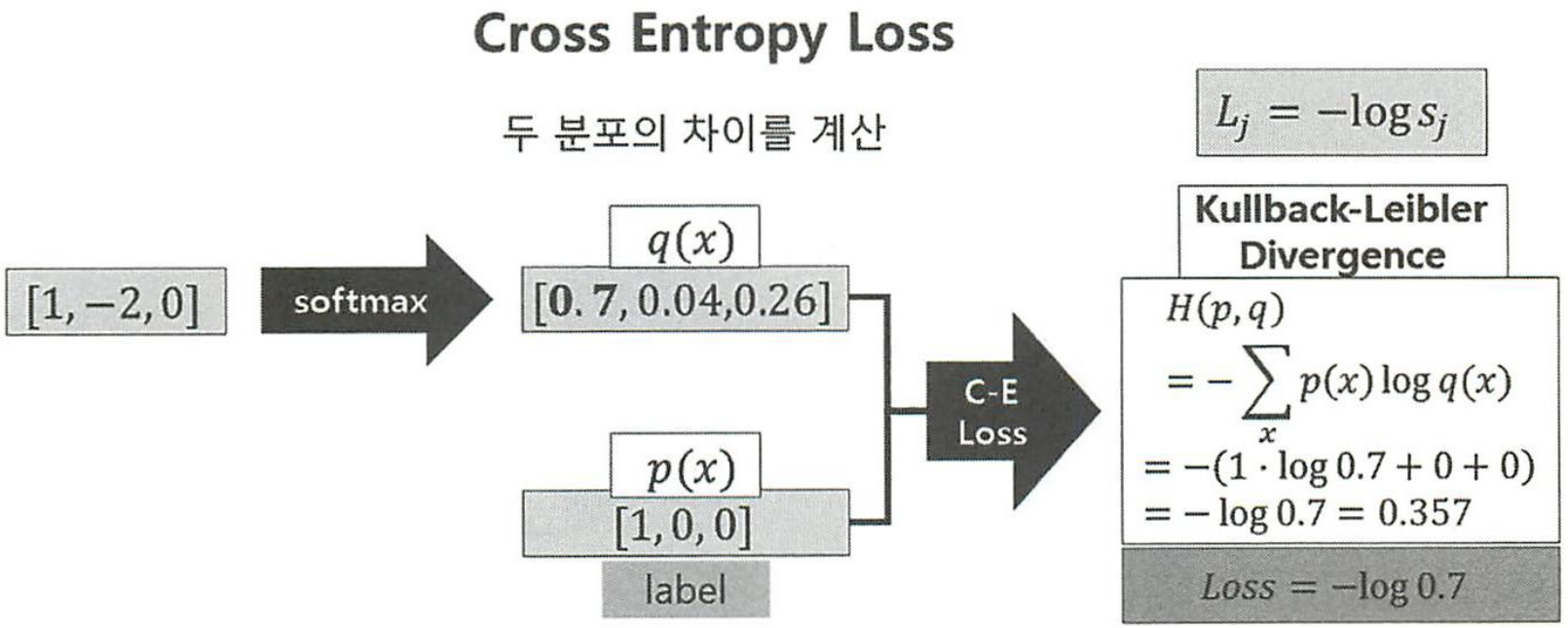
Sofemax 과정

- Cross Entropy Loss



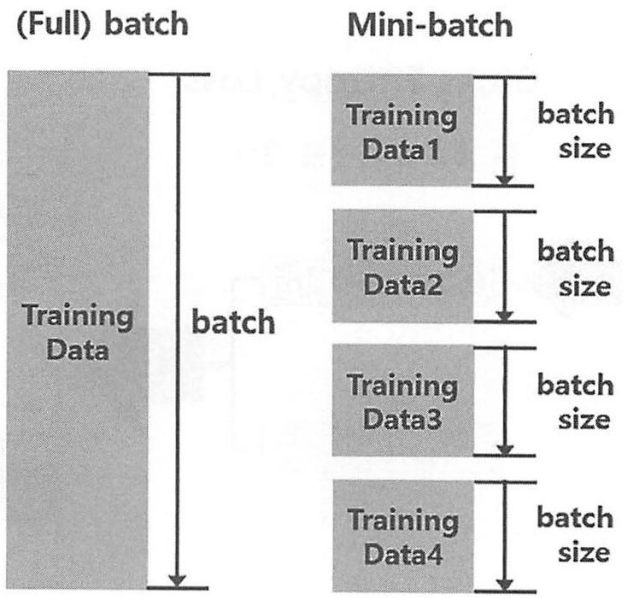
Cross Entropy Loss 과정

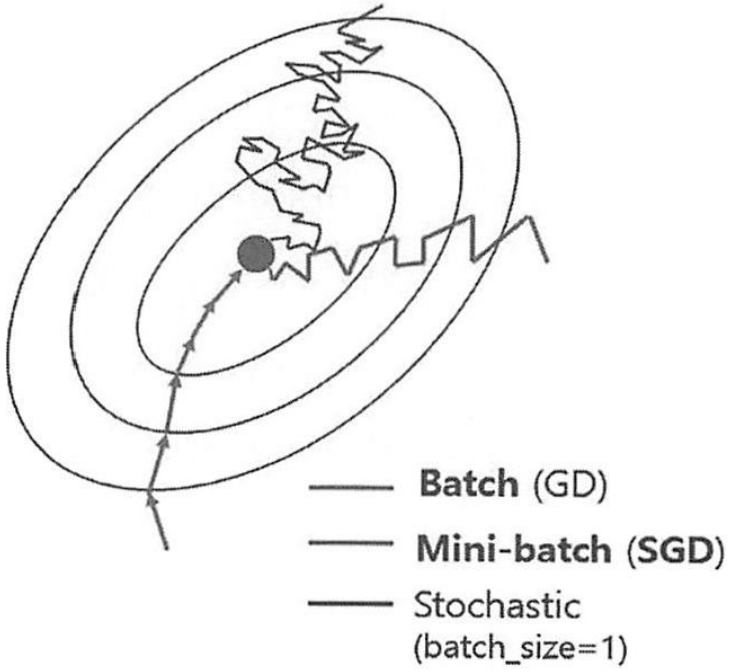
- 하나의 확률 분포와 다른 확률 분포 사이의 엔트로피 변화를 계산
- 결과적으로 손실을 계산하고자 하는 class 위치의 softmax 값 Sj에서 마이너스 로그를 취한 값



최적화 : SGD

- 최적화는 기본적으로 경사하강법을 이용함.
- 전체 학습용 데이터에 한꺼번에 경사 하강법을 적용하면 계산량이 너무 많음
- 계산 비용의 효율성을 고려해 학습 단위 (batch_size)별로 경사 하강법을 적용함.





Mnist 실습

1 기본 라이브러리 불러오기

```
import numpy as np
import pandas as pd
```

2 데이터셋 불러오기

```
from tensorflow.keras.datasets.mnist import load_data
(train_x, train_y), (test_x, test_y) = load_data()
```

2 – 1 데이터 확인하기

```
train_x.shape, train_y.shape    # train 데이터 크기 확인

test_x.shape, test_y.shape    # test 데이터 크기 확인
```

Mnist 실습

2 – 2 이미지 확인하기

```
from PIL import Image
img = train_x[0]

import matplotlib.pyplot as plt
img1 = Image.fromarray(img, mode = 'L')
plt.imshow(img1)

train_y[0]    # 첫번째 데이터 확인
```

Mnist 실습

3 데이터 전처리

3 – 1 입력 형태 변환: 3차원 → 2차원

데이터를 2차원 형태로 변환: 입력 데이터가 선형모델에서는 벡터 형태

```
train_x1 = train_x.reshape(60000, -1)
test_x1 = test_x.reshape(10000, -1)
```

3 – 2 데이터 값의 크기 조절 : 0~1 사이 값으로 변환

```
train_x2 = train_x1/255
test_x2 = test_x1/255
```

Mnist 실습

4 모델 설정

4 – 1 모델 설정용 라이브러리 불러오기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

4 – 2 모델 설정

```
md = Sequential()
md.add(Dense(10, activation = 'softmax', input_shape = (28*28,)))
md.summary()    # 모델 요약
```

Mnist 실습

5 모델 학습 진행

5 – 1 모델 compile: 손실 함수, 최적화 함수, 측정 함수 설정

```
md.compile(loss = 'sparse _ categorical _ crossentropy', optimizer = 'sgd',
metrics = 'acc')
```

5 – 2 모델 학습: 학습 횟수, batch_size, 검증용 데이터 설정

```
hist = md.fit(train _ x2, train _ y, epochs = 30, batch _ size = 64, validation _
split = 0.2)
```

Mnist 실습

학습결과 분석 : 학습 곡선 그리기

```
plt.figure(figsize=(10,8))
plt.plot(epoch, acc, 'b', label='Training accuracy')
plt.plot(epoch, val_acc, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Mnist 실습

6 테스트용 데이터 평가

```
md.evaluate(test_x2, test_y)
```

7 가중치 저장

```
weight = md.get_weights()
weight
```

Mnist 실습

Model Loss 시각화

```
plt.plot(hist.history['loss'], label='loss')
plt.plot(hist.history['val_loss'], label='val_loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```