

KAIST Include 동아리 스터디

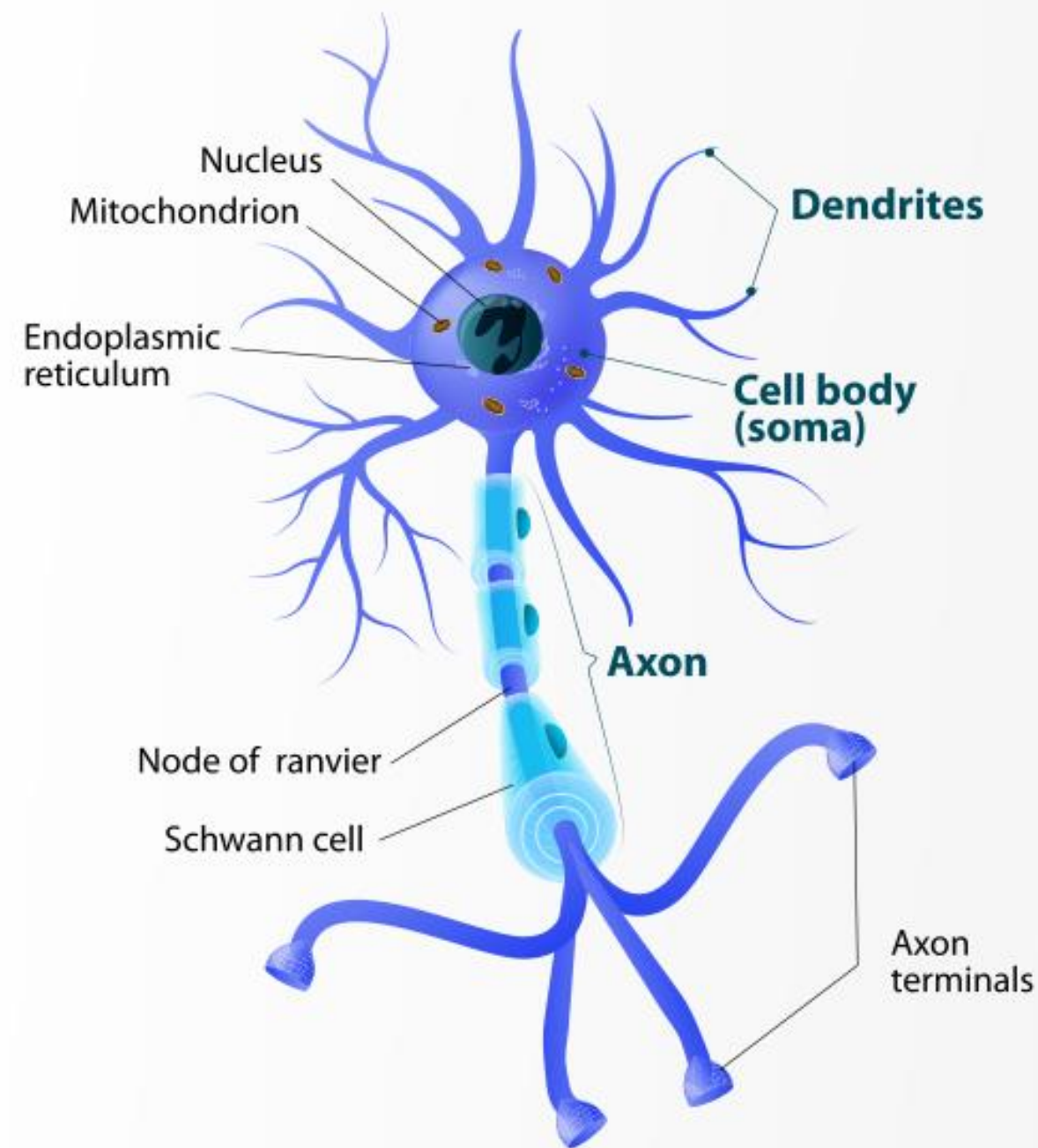
AlphaGo와 AlphaGo Zero를 만들며 익히는 딥러닝 및 강화학습

Chris Ohk

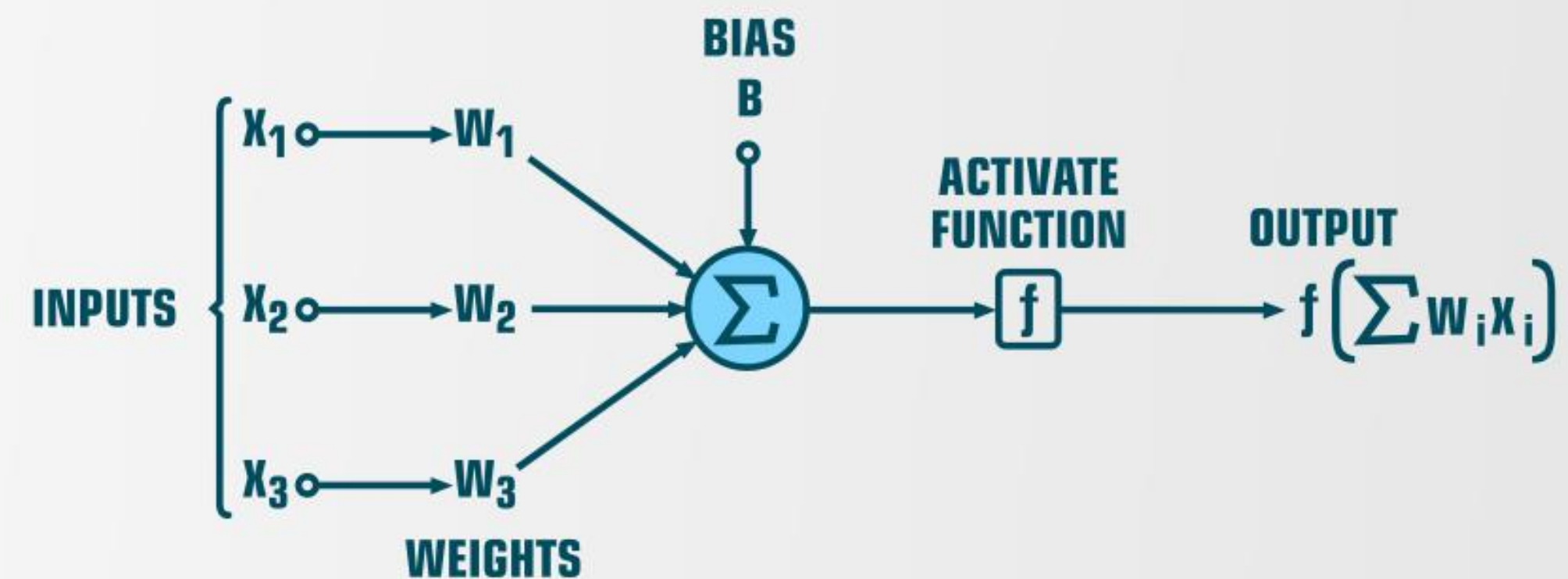
utilForever@gmail.com

- 인공 신경망 (Artificial Neural Network; ANN)

Structure of Typical Neuron



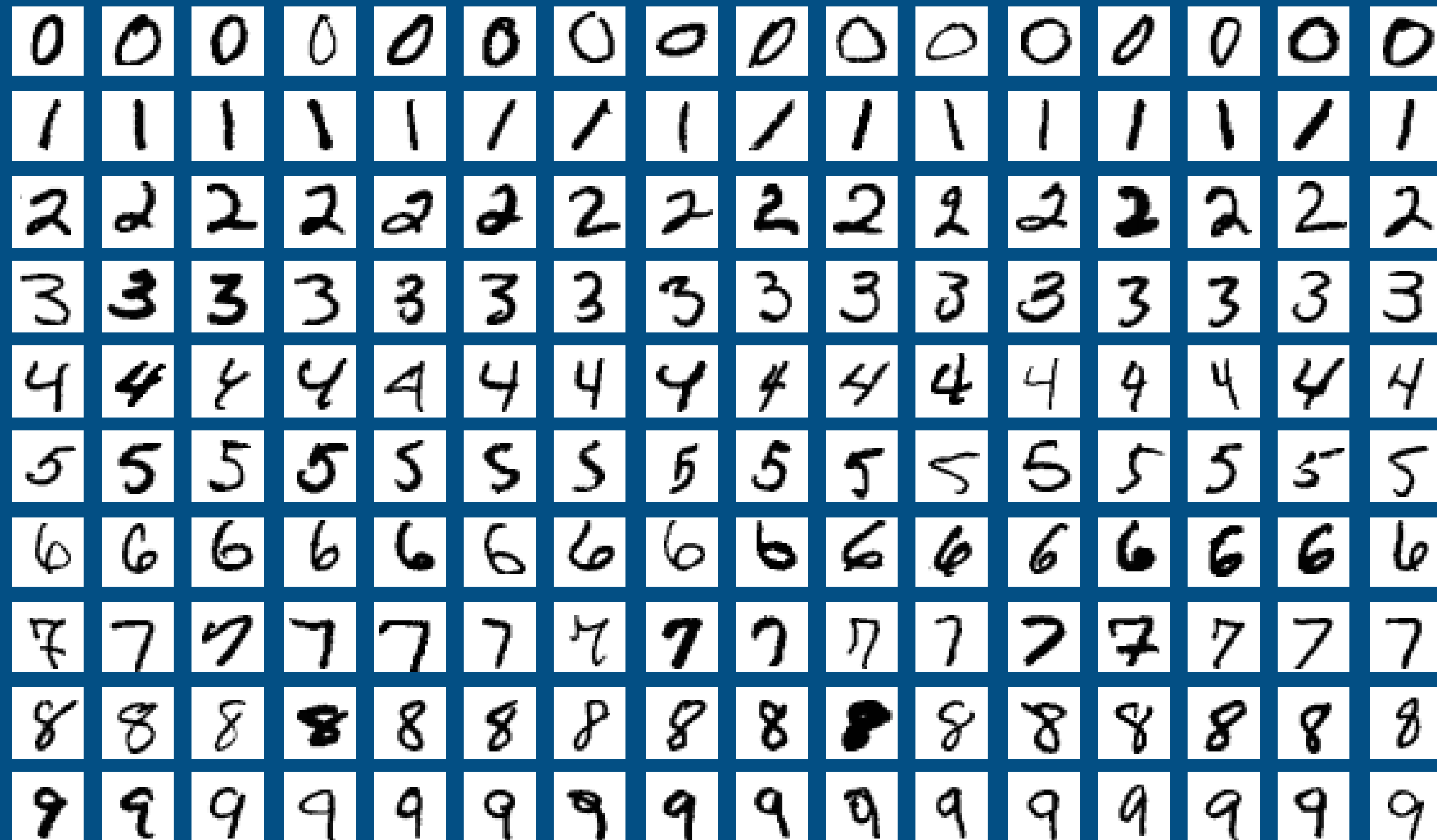
Structure of Artificial Neuron



- numpy
 - 머신러닝 및 수학 컴퓨팅 관련 산업 전반의 Python 표준 라이브러리
 - 설치 방법
 - 1) pip를 사용하는 경우
`pip install numpy`
 - 2) Conda를 사용하는 경우
`conda install numpy`



- MNIST 숫자 손글씨 데이터셋



- MNIST 데이터 처리
 - MNIST 라벨의 원-핫 인코딩

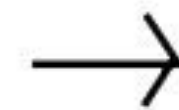
```
import six.moves.cPickle as pickle
import gzip
import numpy as np

def encode_label(j):
    e = np.zeros((10, 1))
    e[j] = 1.0
    return e
```

- 원-핫 인코딩 (One-Hot Encoding)

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50



One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

- MNIST 데이터 처리
 - MNIST 데이터 형태 변형 및 훈련 데이터와 검정 데이터 불러오기
(MNIST 데이터셋의 이미지는 가로/세로 28픽셀) $\rightarrow 28 \times 28 = 784$ 크기의 특징 벡터

```
def shape_data(data):
    features = [np.reshape(x, (784, 1)) for x in data[0]]
    labels = [encode_label(y) for y in data[1]]
    return list(zip(features, labels))

def load_data():
    with gzip.open('mnist.pkl.gz', 'rb') as f:
        data = pickle.Unpickler(f)
        data.encoding = 'latin1'
        train_data, validation_data, test_data = data.load()
    return shape_data(train_data), shape_data(test_data)
```

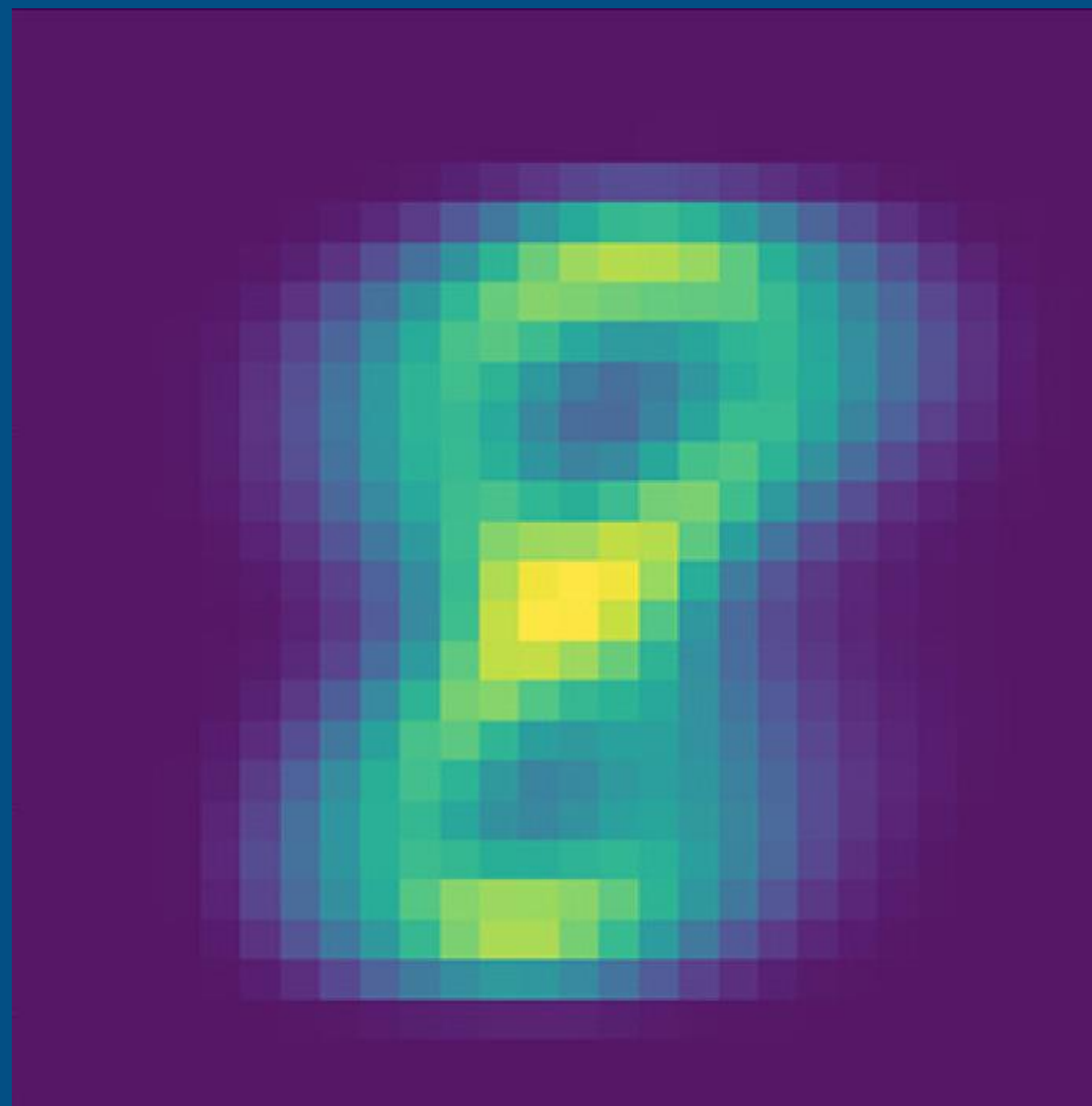
- MNIST 데이터 처리
 - 동일한 수를 나타내는 이미지의 평균값 구하기
(손글씨 숫자를 인식하는 방법의 중요한 열쇠 중 하나 = 패턴 인식)

```
import numpy as np
from dlgo.nn.load_mnist import load_data
from dlgo.nn.layers import sigmoid_double

def average_digit(data, digit):
    filtered_data = [x[0] for x in data if np.argmax(x[1]) == digit]
    filtered_array = np.asarray(filtered_data)
    return np.average(filtered_array, axis=0)

train, test = load_data()
avg_eight = average_digit(train, 8)
```


- 훈련 데이터셋에서의 평균 8



- MNIST 데이터 처리
 - 훈련 데이터셋에서 8인 이미지의 평균을 구하고 출력하기
(avg_eight : 파라미터 → 신경망에서는 이를 가중치(Weight)로 사용할 수 있다.)

```
from matplotlib import pyplot as plt

img = (np.reshape(avg_eight, (28, 28)))
plt.imshow(img)
plt.show()
```

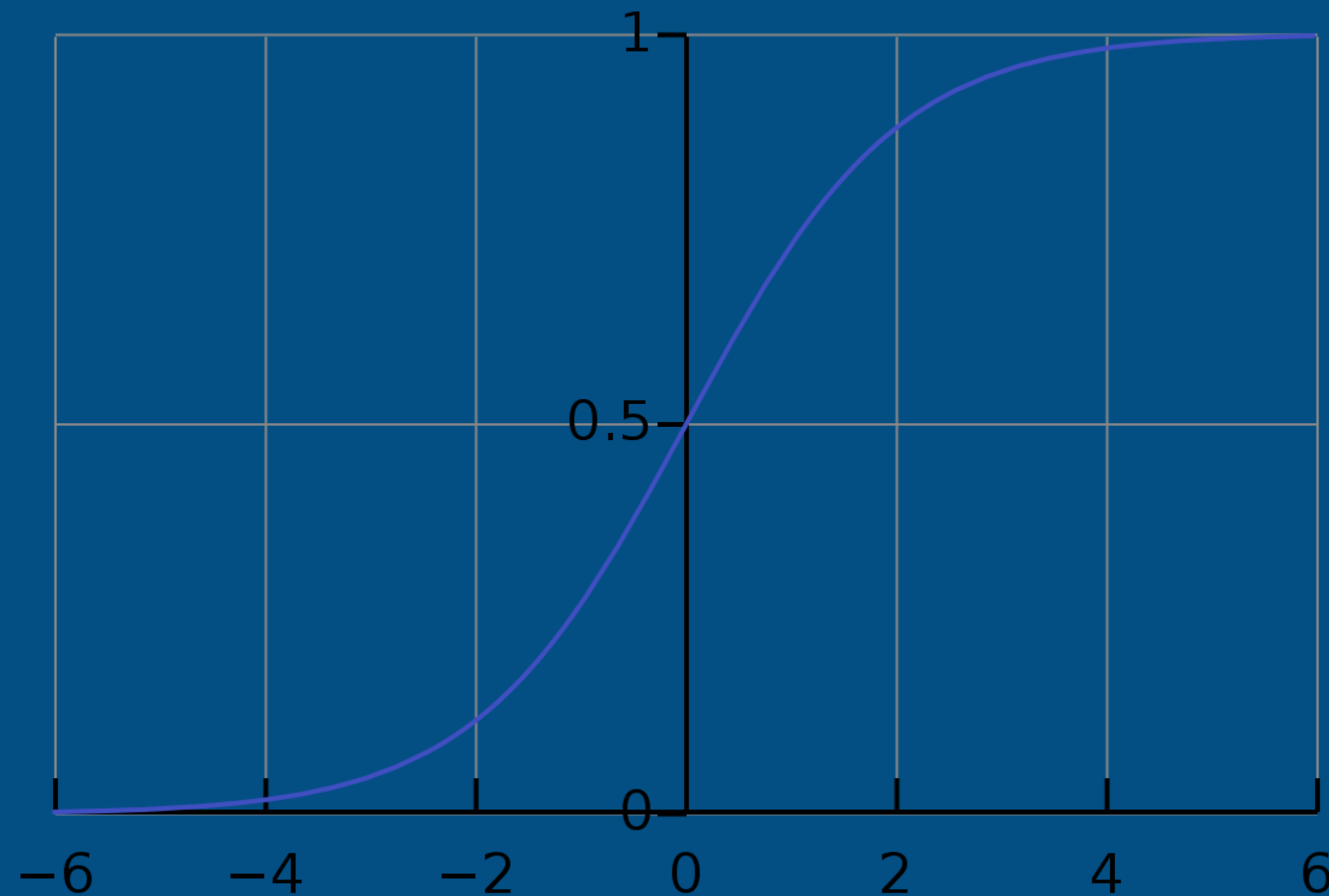
- MNIST 데이터 처리
 - 내적을 사용해 숫자가 가중치에 얼마나 가까운지 계산하기
→ 내적의 출력을 $[0, 1]$ 범위로 변환하려면 어떻게 해야 할까?

```
x_3 = train[2][0]
x_18 = train[17][0]

W = np.transpose(avg_eight)
np.dot(W, x_3)
np.dot(W, x_18)
```

- MNIST 데이터 처리

- 시그모이드 함수 (Sigmoid Function) : $\sigma(x) = \frac{1}{1+e^{-x}}$



- MNIST 데이터 처리
 - double 값과 벡터에 대한 간단한 시그모이드 함수 구현

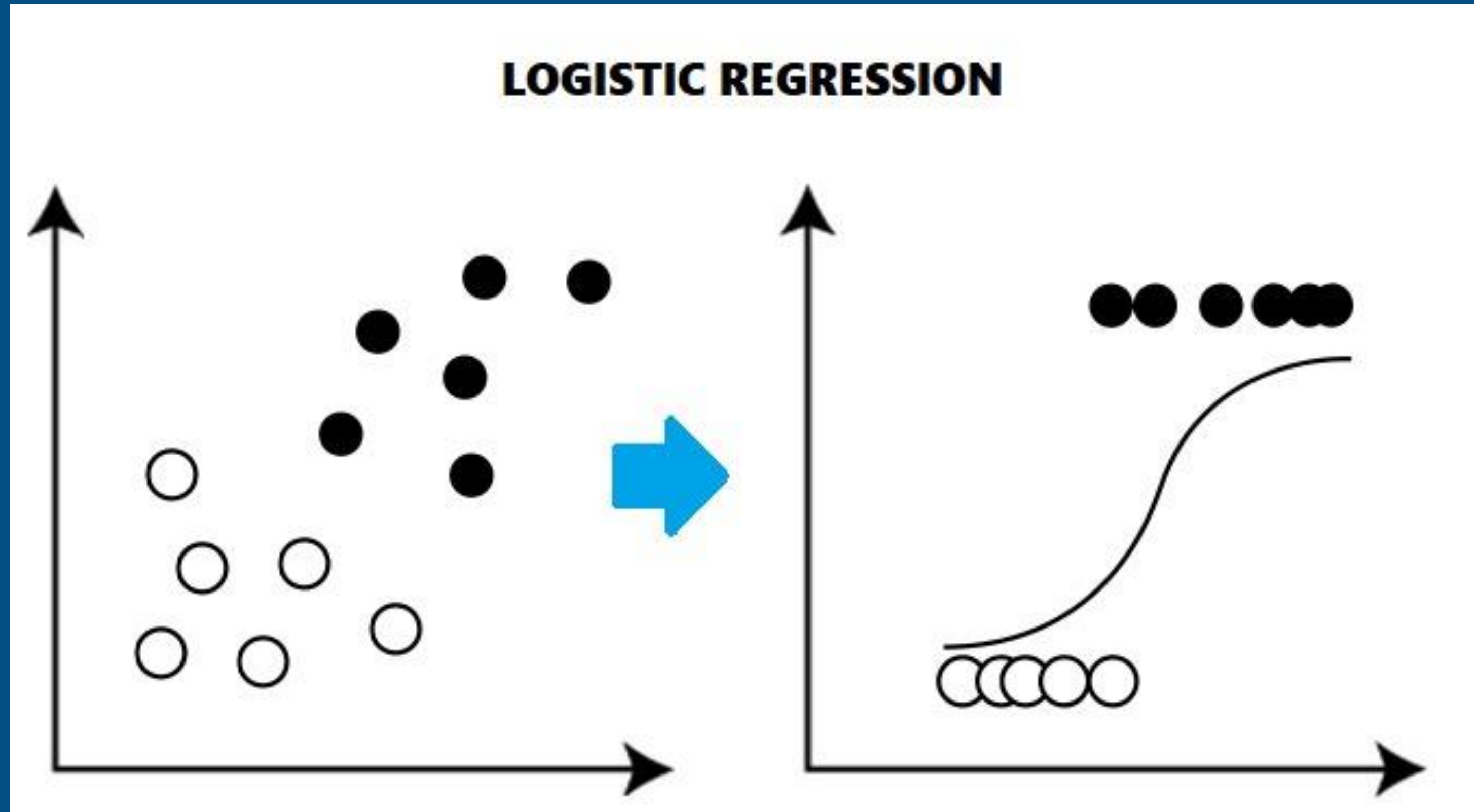
```
def sigmoid_double(x):  
    return 1.0 / (1.0 + np.exp(-x))  
  
def sigmoid(z):  
    return np.vectorize(sigmoid_double)(z)
```

- MNIST 데이터 처리
 - 시그모이드 함수를 사용해 적용해도 문제가 있다. 예를 들어, $\text{sigmoid}(54.2)$ 와 $\text{sigmoid}(20.1)$ 은 실제로 구분하기 어렵다. 이 문제를 어떻게 해결해야 할까?
 - 내적값의 결과를 0 부근으로 옮기는 방식으로 해결할 수 있다.
보통 b 로 나타내는 수치인 편향치(Bias)를 추가한다.

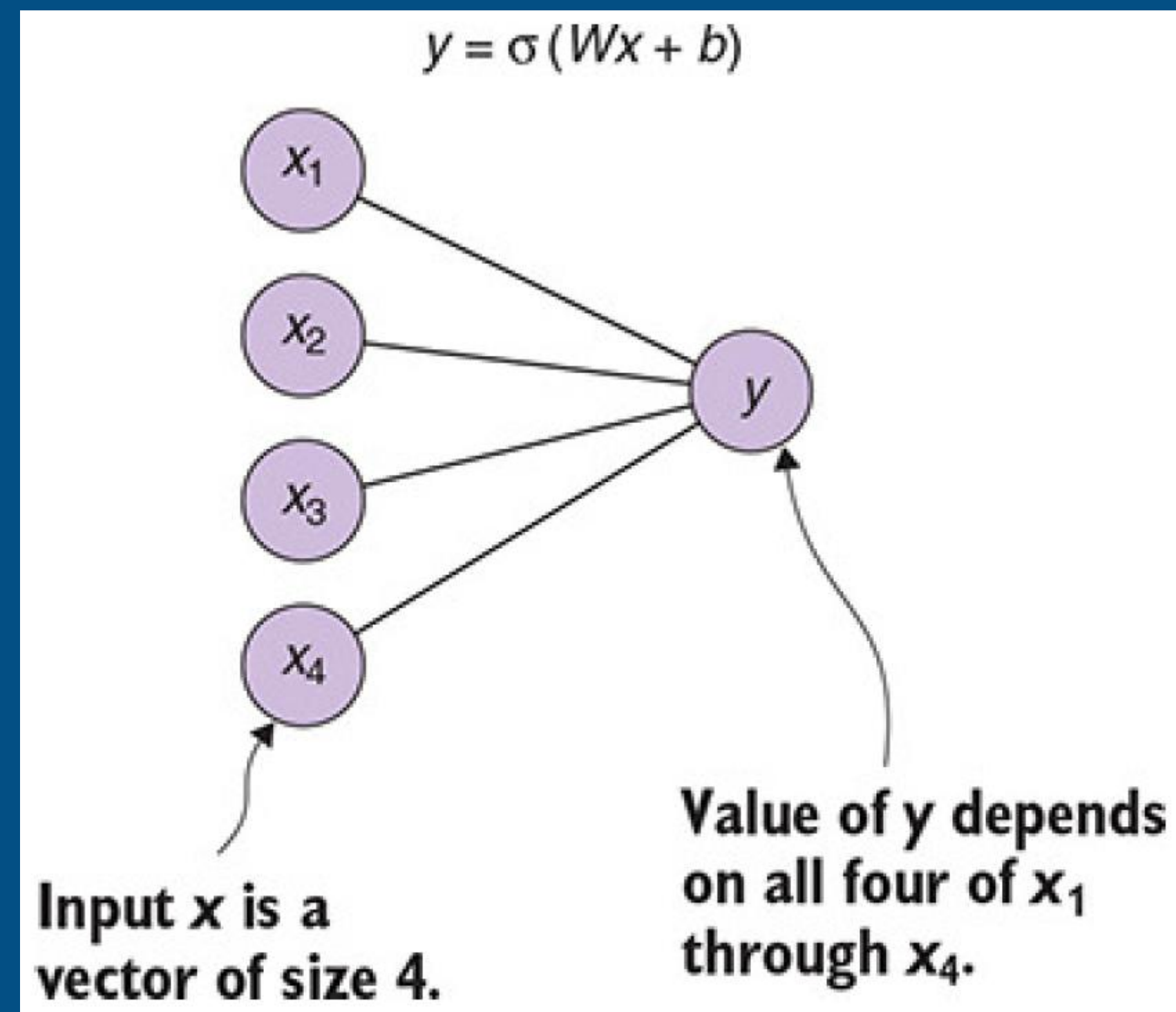
- MNIST 데이터 처리
 - 시그모이드와 내적을 적용해서 가중치와 편향치로부터 예측값 구하기

```
def predict(x, W, b):  
    return sigmoid_double(np.dot(W, x) + b)  
  
b = -45  
  
print(predict(x_3, W, b))  
print(predict(x_18, W, b))
```

- MNIST 데이터 처리
 - 로지스틱 회귀 (Logistic Regression)



- MNIST 데이터 처리
 - 로지스틱 회귀 (Logistic Regression)



- MNIST 데이터 처리
 - 결정 임계치를 사용한 모델 예측값 평가

```
def evaluate(data, digit, threshold, W, b):  
    total_samples = 1.0 * len(data)  
    correct_predictions = 0  
    for x in data:  
        if predict(x[0], W, b) > threshold and np.argmax(x[1]) == digit:  
            correct_predictions += 1  
        if predict(x[0], W, b) <= threshold and np.argmax(x[1]) != digit:  
            correct_predictions += 1  
    return correct_predictions / total_samples
```

- MNIST 데이터 처리
 - 세 데이터셋의 예측 정확도 구하기



```
evaluate(data=train, digit=8, threshold=0.5, W=W, b=b)

evaluate(data=test, digit=8, threshold=0.5, W=W, b=b)

eight_test = [x for x in test if np.argmax(x[1]) == 8]
evaluate(data=eight_test, digit=8, threshold=0.5, W=W, b=b)
```

- 지금까지의 과정에서 무엇이 잘못되었고, 무엇을 개선할 수 있을까?
 - 이 모델은 특정 숫자(8)만 구분할 수 있다. 훈련 데이터셋과 검정 데이터셋에서 8은 10% 뿐이다. → 10가지 숫자 모두를 정확하게 예측하는 모델을 만들어야 한다.
 - 모델에 들어가는 파라미터가 작다. → 데이터의 다양성을 잡아낼 수 있도록 훨씬 더 많은 파라미터를 효과적으로 사용하는 알고리즘을 찾아야 한다.
 - 주어진 예측에서는 숫자가 8이냐 8이 아니냐를 결정하는 기준값을 선택하는 것 뿐이다. → 예측값이 실제 결과와 얼마나 가까운지에 대한 개념을 공식화해야 한다.
 - 모델의 파라미터를 만들 때 직감에 따라 값을 직접 조정했다. → 훈련 데이터에 대해 얼마나 잘 예측하는지에 따라 모델의 파라미터를 갱신하는 절차를 마련해야 한다.

- 단순한 인공 신경망으로의 로지스틱 회귀
 - 앞에서 우리는 이항 분류에 로지스틱 회귀를 사용했다.
샘플 데이터를 나타내는 특징 벡터 x 를 가져와서 여기에 가중치 행렬 W 를 곱한 후 편향치 b 를 더하는 알고리즘에 넣어준다. 그리고 예측값 y 를 0과 1 사이로 나타내서 마무리하기 위해 이 값에 시그모이드 함수 $y = \sigma(Wx + b)$ 를 취한다.
 - 특징 벡터 x 는 유닛이라고도 하는 뉴런 집합으로 해석할 수 있다. 이 유닛은 W 와 b 의 값에 의해 변경되어 y 로 연결된다. 이어서 활성화 함수로 시그모이드 함수를 사용해 $Wx + b$ 의 결과를 $[0, 1]$ 범위에 대응시킨다. 만약 이 값이 1에 가깝다면 뉴런 y 를 활성화하고, 0에 가깝다면 비활성화한다.

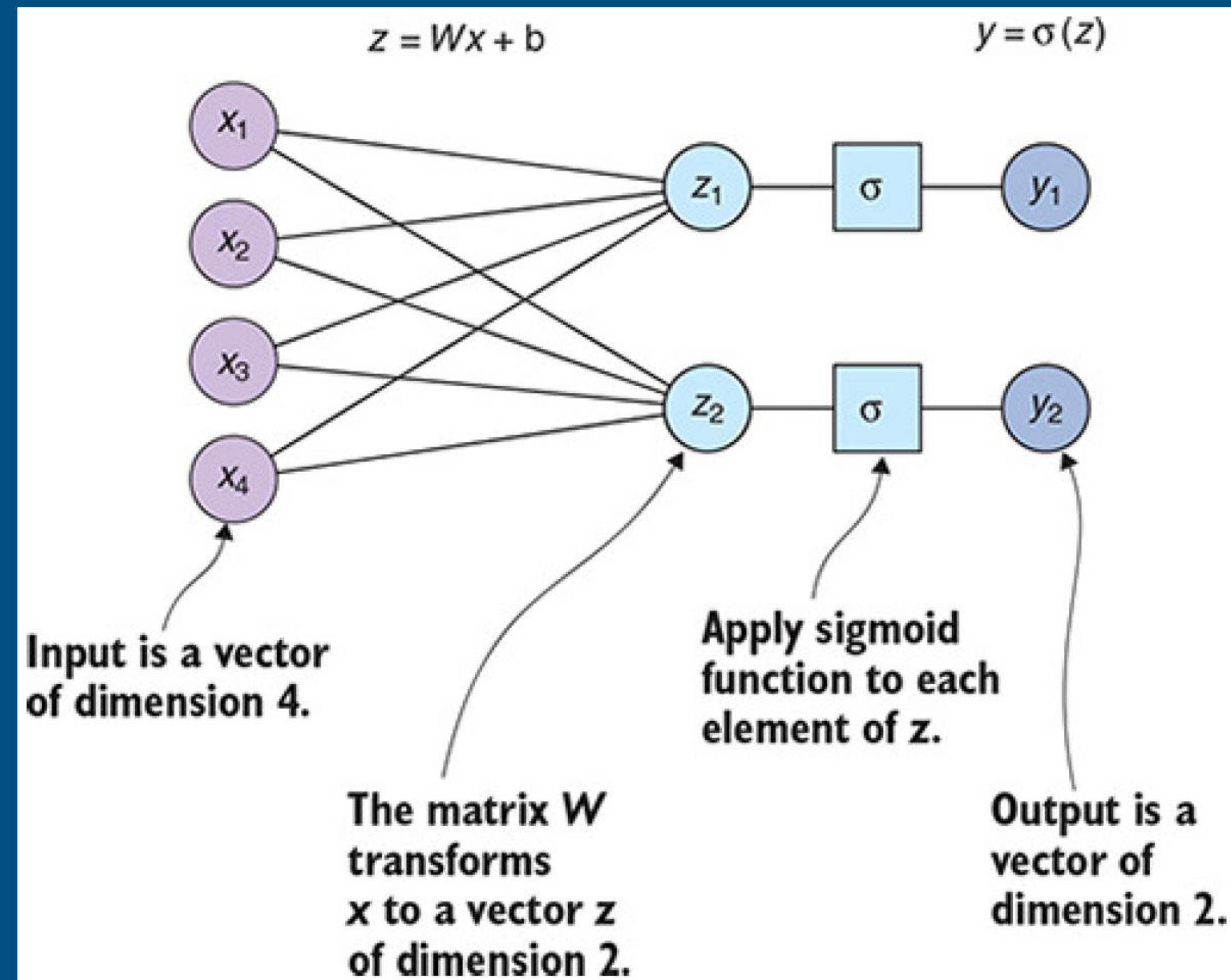
- 1차원 이상의 결과를 갖는 신경망
 - 앞에서 손글씨 숫자 인식 문제를 8과 나머지 숫자로 분류하는 이항 분류로 단순화했다. 하지만 각 숫자를 예측해보고 싶을 것이다. 이제 모델을 수정해 보자.
 - 우선 길이 10의 결과 벡터 y 를 만든다. (y 는 각 숫자별 가능도를 나타내는 값을 갖는다.)

$$y = \begin{bmatrix} y_0 \\ y_1 \\ \dots \\ y_8 \\ y_9 \end{bmatrix}$$

- 1차원 이상의 결과를 갖는 신경망
 - 다음으로 가중치와 편향치를 변경한다. (W 는 원래 길이 784의 벡터였다.)
여기서는 W 를 (10, 784) 차원의 행렬로 만든다. 그러면 W 와 입력 벡터 x 의 행렬곱 Wx 로 길이 10의 벡터를 구할 수 있다. 여기에 길이 10인 벡터로 편향치를 만들면 이를 Wx 에 더할 수 있다. 마지막으로 벡터 z 를 각 값에 적용해 시그모이드를 계산할 수 있다.

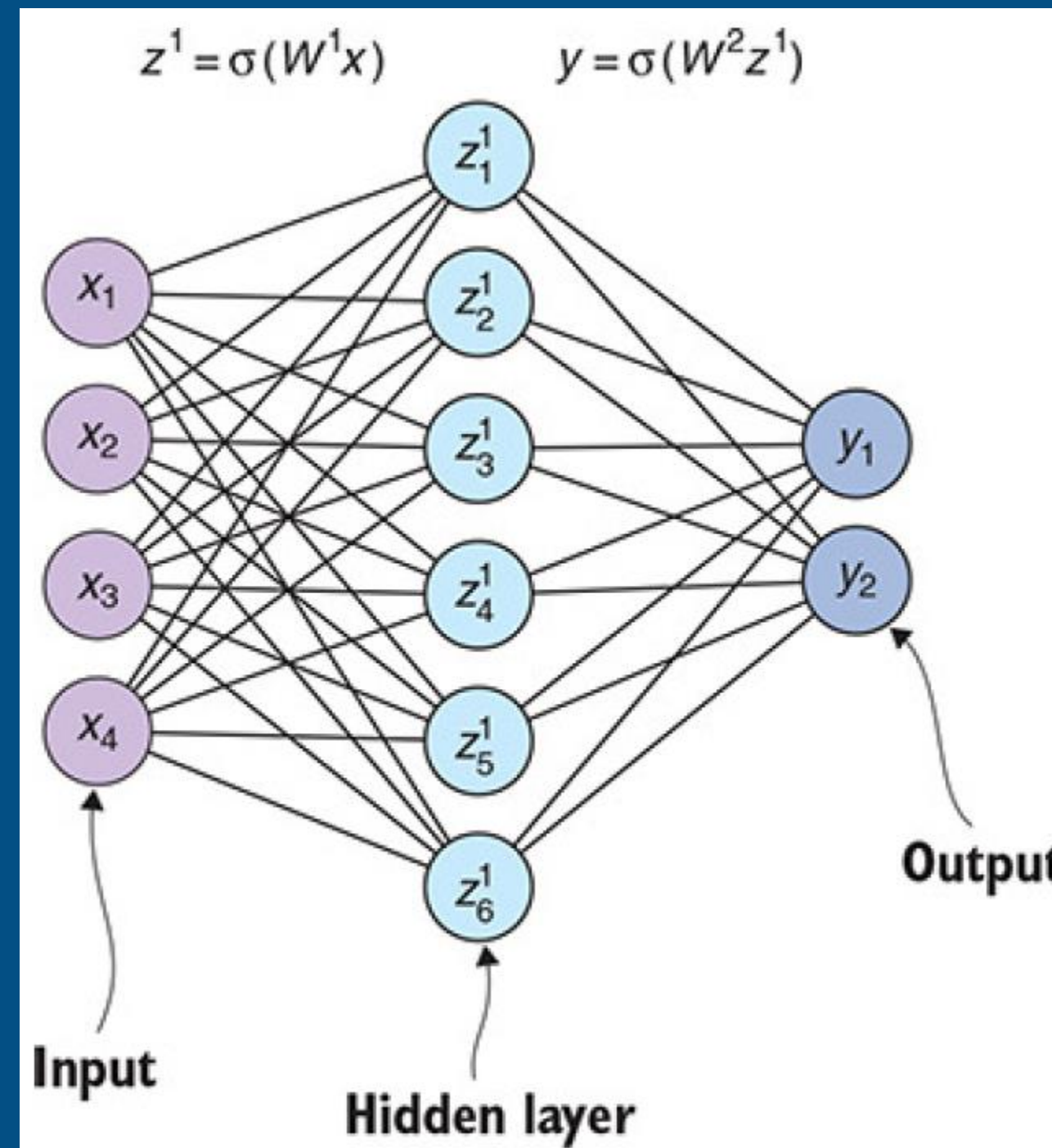
$$\sigma(z) = \begin{bmatrix} \sigma(z_0) \\ \sigma(z_1) \\ \dots \\ \sigma(z_8) \\ \sigma(z_9) \end{bmatrix}$$

- 1차원 이상의 결과를 갖는 신경망
 - 순방향 신경망 (Feed-Forward Network)

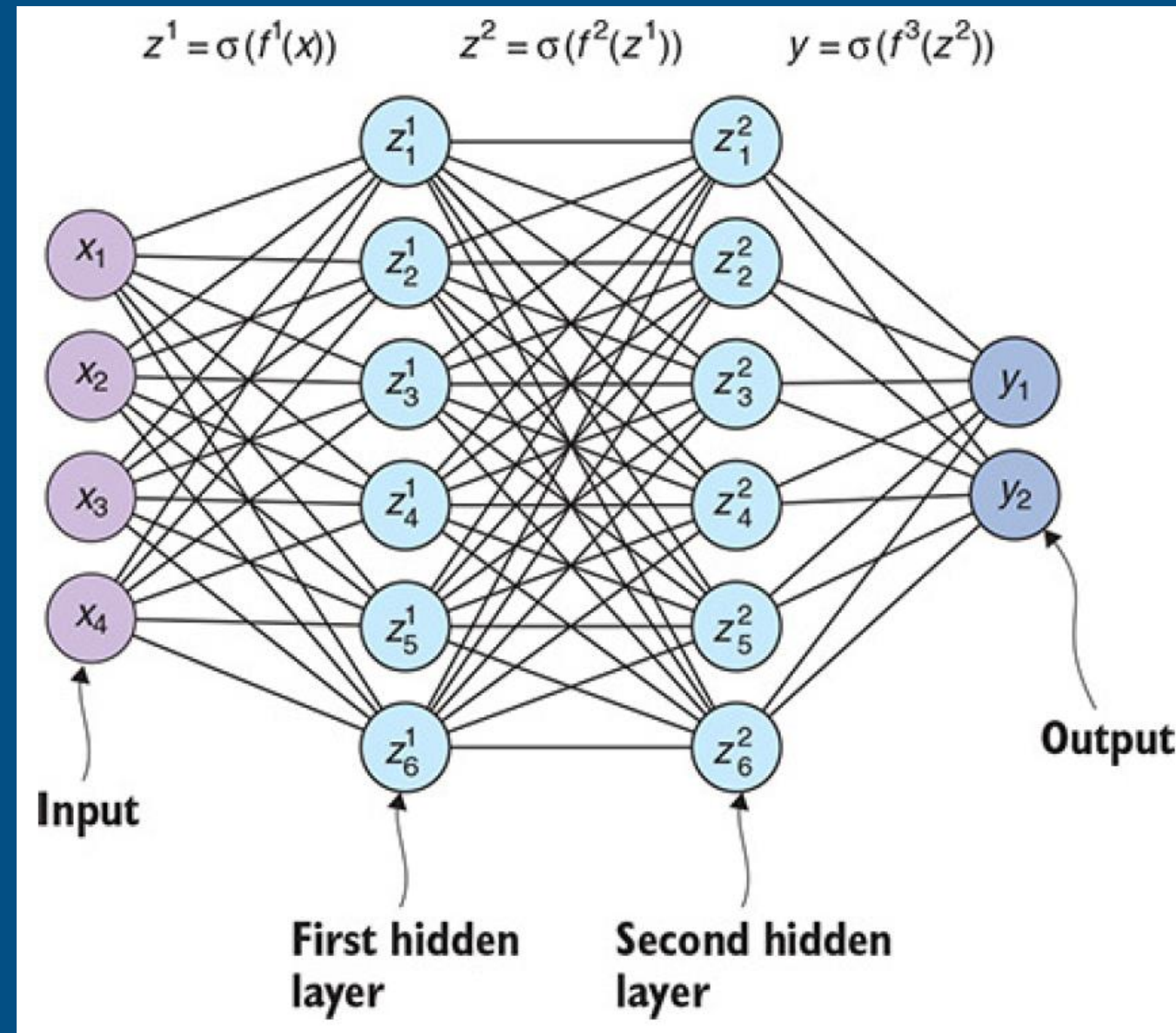


- 앞에서 설명했던 내용을 짧게 정리해보자.
 1. 입력 뉴런 x 에 $z = Wx + b$ 형식으로 간단한 변형을 취한다. 선형대수학 용어로 이런 변형을 아핀 선형 변환(Affine-Linear Transformation)이라고 한다.
여기서는 과정을 쉽게 따라갈 수 있게 z 라는 중간 변수를 사용했다.
 2. 활성화 함수인 시그모이드 함수 $y = \sigma(z)$ 를 취해 결과 뉴런 y 를 구한다.
이 결과로 얼마나 많은 y 가 활성화되는지 확인할 수 있다.
- 순방향 신경망의 핵심은 이런 과정을 반복적으로 취해
이 두 단계로 이루어진 간단한 과정을 여러 번 적용하는 것이다.
→ 많은 층을 쌓아서 다층 신경망을 구성하는 것

- 만약 한 층을 더 추가하면 어떤 단계를 거치게 될까?
 - 입력값 x 에 $z^1 = W^1x + b^1$ 을 구하자.
 - 중간 결과 z^1 으로부터 $y = W^2z^1 + b^2$ 을 구해서 출력값 y 를 구하자.



- 순방향 전달 (Forward Pass) : 활성화 함수를 신경망의 모든 층에서 하나 혹은 여러 데이터에 적용하는 것 (순방향 : 입력 → 출력)



- 지금까지 이야기했던 모든 개념을 정리해보자.
- 순차 신경망은 특징 (혹은 입력 뉴런) x 를 예측값 (혹은 출력 뉴런) y 에 대응시키는 과정이다. 이 과정은 순서대로 하나씩 함수층을 쌓아가면서 이루어진다.
- 층은 입력값에 변화를 가해 출력값을 도출하는 곳이다. 특정 크기의 데이터에 대한 층의 출력을 구하는 한 차례의 처리를 순방향 전달이라고 한다. 마찬가지로 순차 신경망에서 순방향 전달은 입력에서 출력 방향으로, 즉 순방향으로 각 층의 출력값을 순서대로 구하는 것이다.
- 시그모이드 함수는 실수 뉴런 벡터를 사용해서 $[0,1]$ 의 범위에 대응시키는 방식으로 활성화시키는 활성화 함수다. 이 값이 1에 가까우면 활성화 상태라고 해석한다.

- 지금까지 이야기했던 모든 개념을 정리해보자.
- 가중치 행렬 W 와 편향치 b 가 주어졌을 때 아핀 선형 변환 $Wx + b$ 는 층을 구성한다. 이런 종류의 층을 보통 밀집층 혹은 완전 연결층이라고 한다.
- 구현에 따라 밀집층에는 활성화 함수가 포함되기도 하고 아니기도 하다. $\sigma(Wx + b)$ 를 아핀 선형 변환이 아닌 층으로 볼 수도 있다. 일반적으로 활성화 함수만을 층으로 간주한다. 결국 밀집층에 활성화 함수를 추가할 것인지 아닌지는 기능의 일부를 어떻게 분할하고 논리적인 단위로 그룹화할 것인지에 대한 약간 다른 견해일 뿐이다.
- 순방향 신경망은 밀집층과 활성화 함수로 구성된 순차 신경망이다. 이런 구조를 다층 퍼셉트론(Multilayer Perceptron) 혹은 짧게 MLP라고 부른다.

- 지금까지 이야기했던 모든 개념을 정리해보자.
- 입력 뉴런이나 출력 뉴런이 아닌 모든 뉴런은 은닉 유닛이고, 입력 및 출력 뉴런은 가시 유닛이다. 여기에는 은닉 유닛은 신경망 안에, 가시 유닛은 바로 관측 가능한 곳에 있다는 개념이 깔려 있다. 마찬가지로 입력과 출력 사이의 층은 은닉층(Hidden Layer)이라고 한다. 두 개 층 이상을 가지는 순차 신경망은 최소 1개 이상의 은닉층을 가진다.
- 많은 은닉층을 가진 거대한 신경망을 구축하려고 많은 층을 쌓은 경우를 심층 신경망(Deep Neural Network)이라고 하며, 그래서 딥러닝(Deep Learning)이라는 이름이 붙었다.

- 손실 함수(Loss Function)란 무엇인가
 - 예측값의 적중률이 어느 정도 되는지를 수치화한 것 (목적 함수(Objective Function)라고도 한다.)

$$\sum_i \text{Loss}(W, b, X_i, \hat{y}_i) = \sum_i \text{Loss}(y_i, \hat{y}_i)$$

- 여기서 $\text{Loss}(y_i, \hat{y}_i) \geq 0$ 이고, Loss는 미분 가능 함수다.
- 주어진 데이터에 대한 알고리즘의 파라미터가 얼마나 적합한지 평가한다.
- 훈련 목표는 주어진 파라미터에 맞추는 좋은 전략을 찾아서 손실을 최소화하는 것이다.

- 평균제곱오차 (Mean Square Error; MSE)

$$\text{MSE}(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^k (y_i - \hat{y}_i)^2$$

- 모든 관측된 예제에 대해 거리 제곱을 측정한 후 평균을 구함으로써 예측값이 실제 라벨에 얼마나 가까운지 측정하게 된다.

- 평균제곱오차 (Mean Square Error; MSE)
 - 평균제곱오차 손실 함수의 미분값

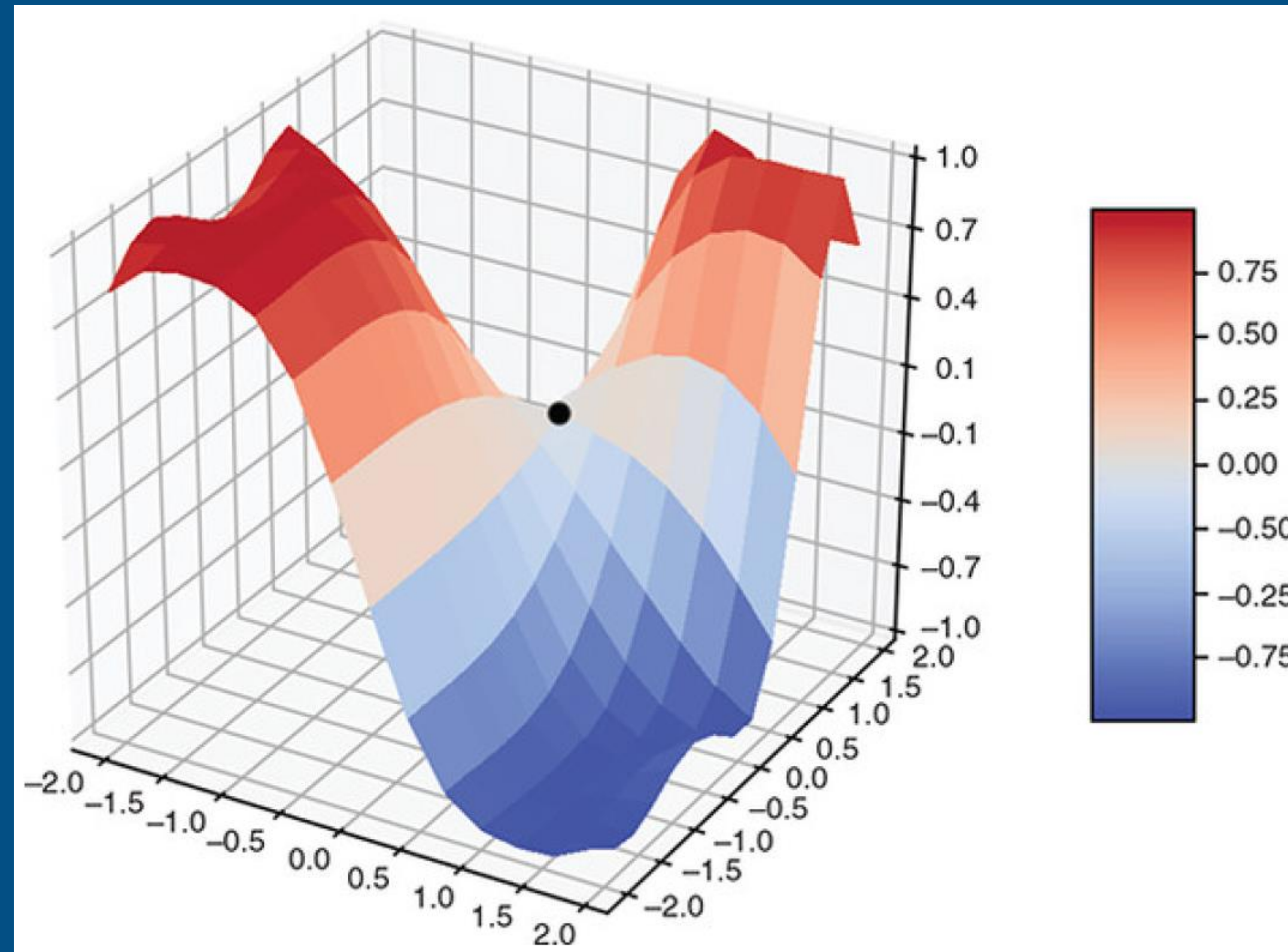
```
import random
import numpy as np

class MSE:
    def __init__(self):
        pass

    @staticmethod
    def loss_function(predictions, labels):
        diff = predictions - labels
        return 0.5 * sum(diff * diff)[0]

    @staticmethod
    def loss_derivative(predictions, labels):
        return predictions - labels
```

- 손실 함수에서의 최솟값 찾기
 - 함수를 미분해 특정 지점에서의 기울기를 구한다.



- 최솟값을 찾는 경사하강법 (Gradient Descent) 알고리즘

1. 현재의 파라미터 집합 W 에 대한 Loss의 기울기 Δ 를 구한다.

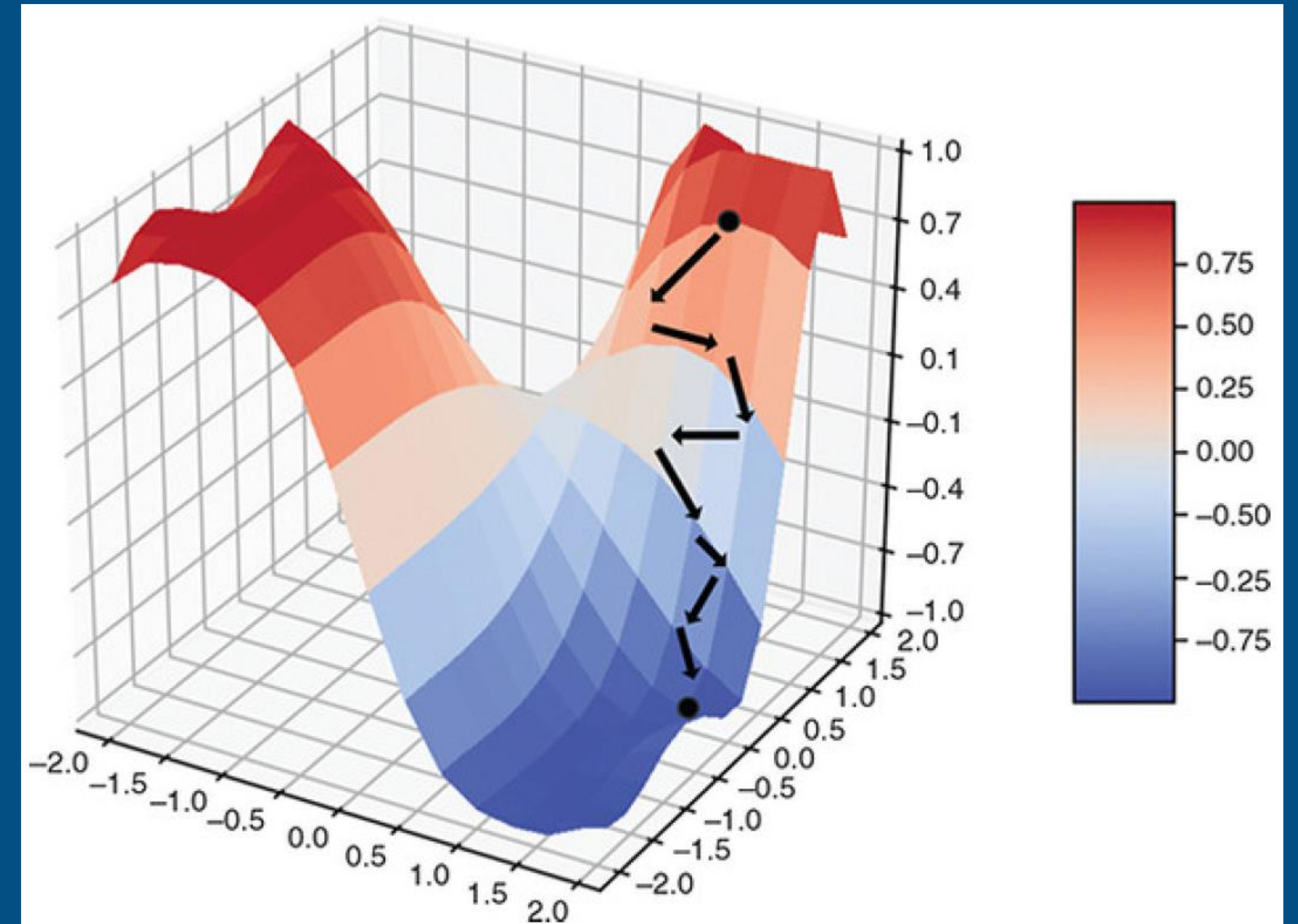
(각 W 에 대해 Loss의 미분값을 구함)

2. 이제 W 에서 Δ 를 빼서 갱신한다.

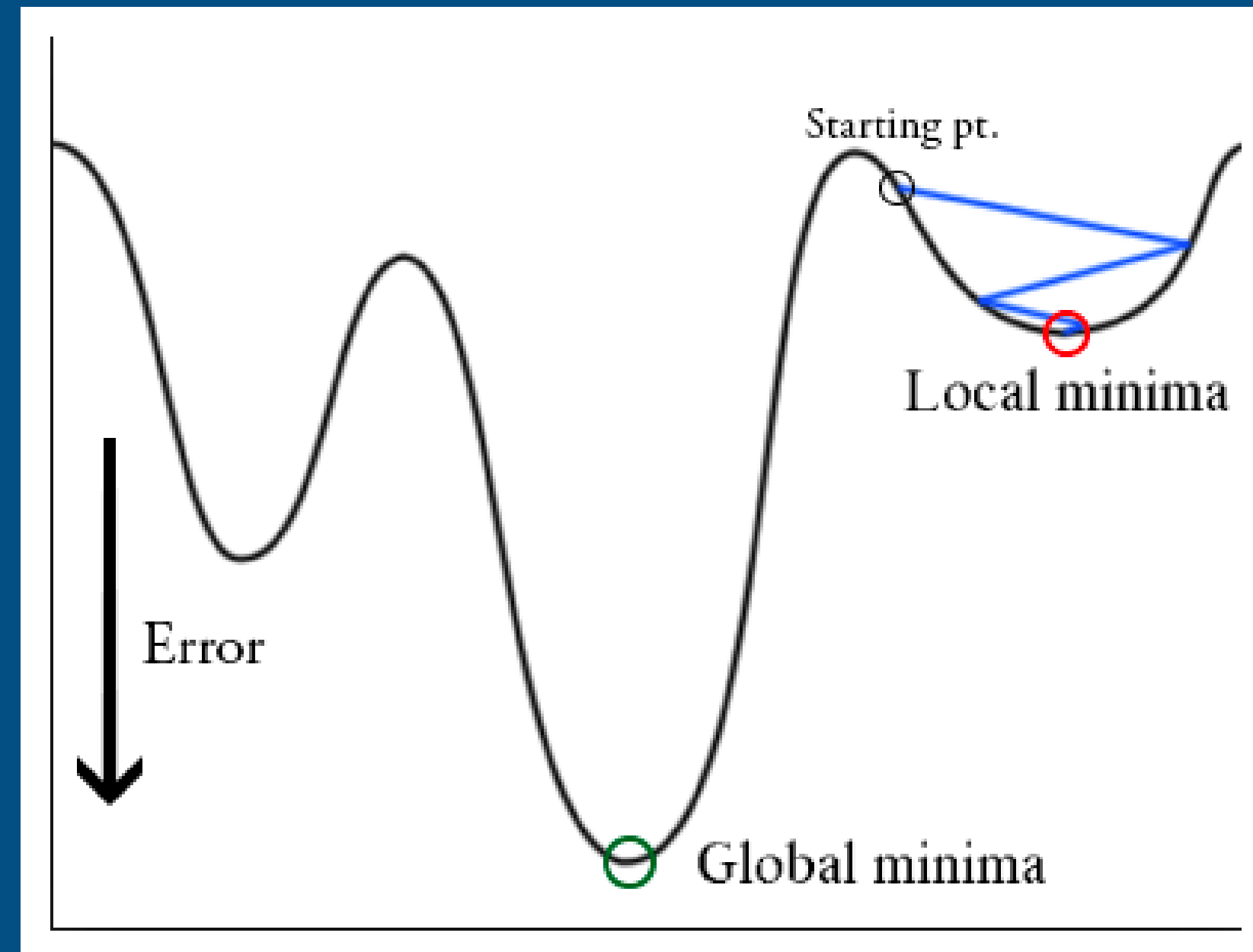
이 과정을 기울기를 따라간다고 한다.

만약 Δ 가 가장 가파르게 상승하는 방향을
가리킨다면 이 값을 뺌으로써
가장 가파르게 하강할 것이기 때문이다.

3. Δ 가 0이 될 때까지 반복한다.



- 최솟값을 찾는 경사하강법 (Gradient Descent) 알고리즘
 - 지역 최솟값과 전역 최솟값



- 최솟값을 찾는 경사하강법 (Gradient Descent) 알고리즘
 - 모든 파라미터의 기울기를 살펴보려면 훈련 데이터셋의 모든 샘플에 대해 미분을 구하고 집계해야 한다. 모든 샘플을 다루기란 실질적으로 불가능하다.
→ 확률적 경사하강법의 등장!

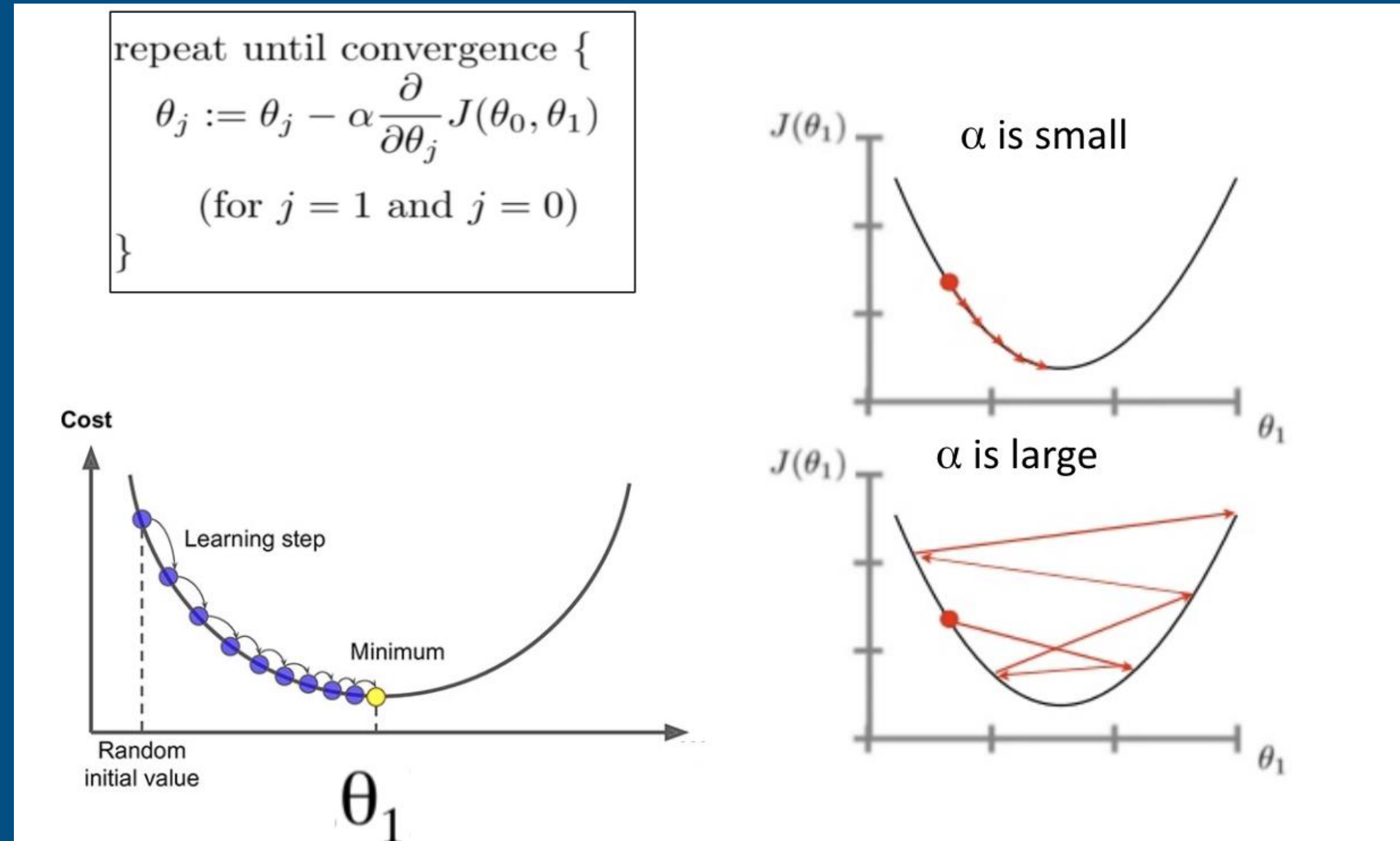
- 확률적 경사하강법 (Stochastic Gradient Descent, SGD)

- 훈련 데이터 셋에서 샘플 데이터의 일부를 가져온다. → 미니배치 (Mini-Batch)
- 각 미니배치는 일정한 길이를 갖는다. → 미니배치 크기
- 배치에서 사용한 각 층 및 샘플에 대해 기울기를 계산하고
파라미터별로 다음과 같은 갱신 규칙 (= 최적화기 (Optimizer))을 적용할 수 있다.

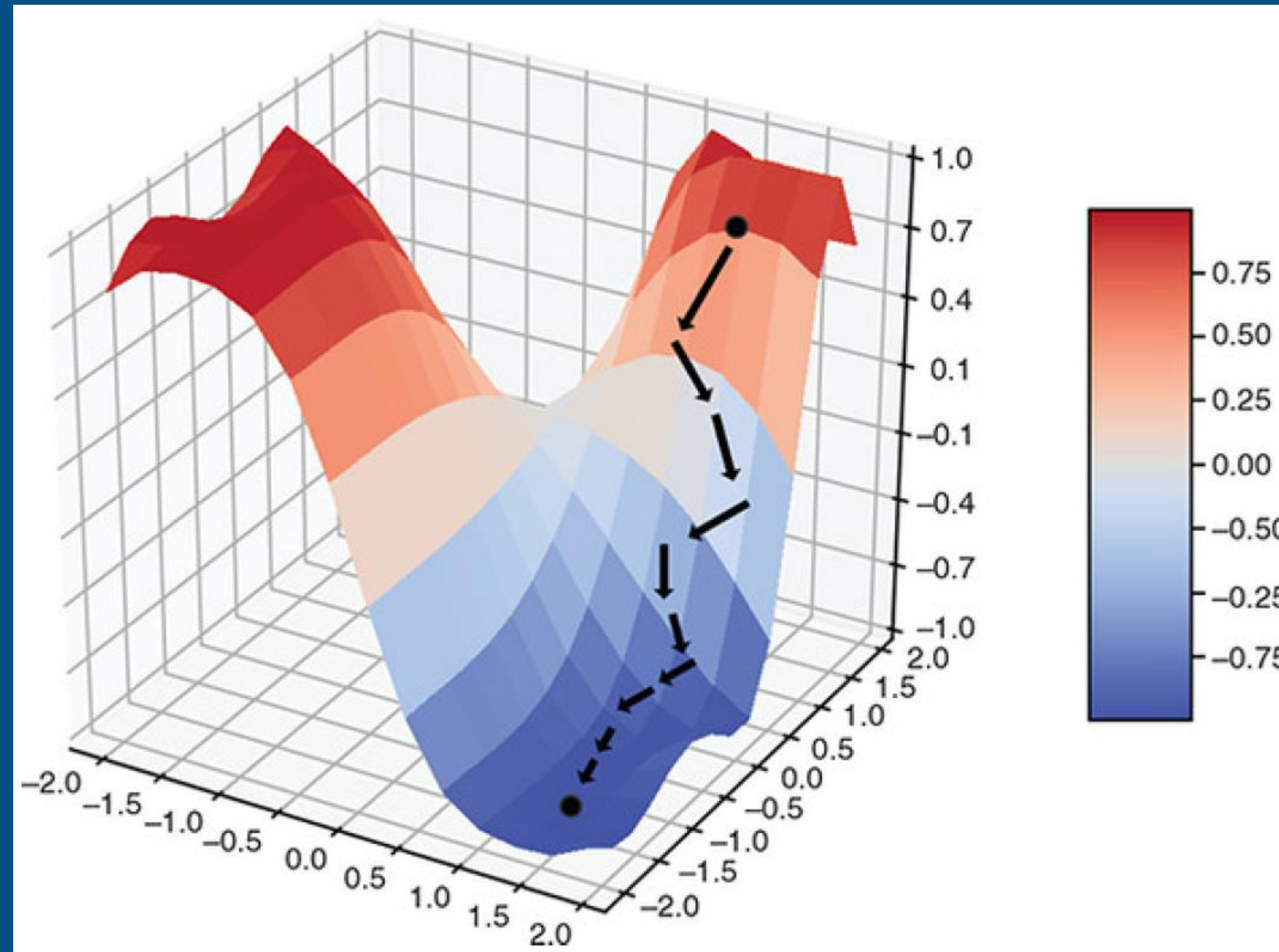
$$W^i \leftarrow W^i - \alpha \sum_{j=1}^k \Delta_j W^i$$
$$b^i \leftarrow b^i - \alpha \sum_{j=1}^k \Delta_j b^i$$

- 여기서 $\alpha > 0$ 은 학습률 (Learning Rate)로 전반적인 신경망 훈련 정도를 나타낸다.

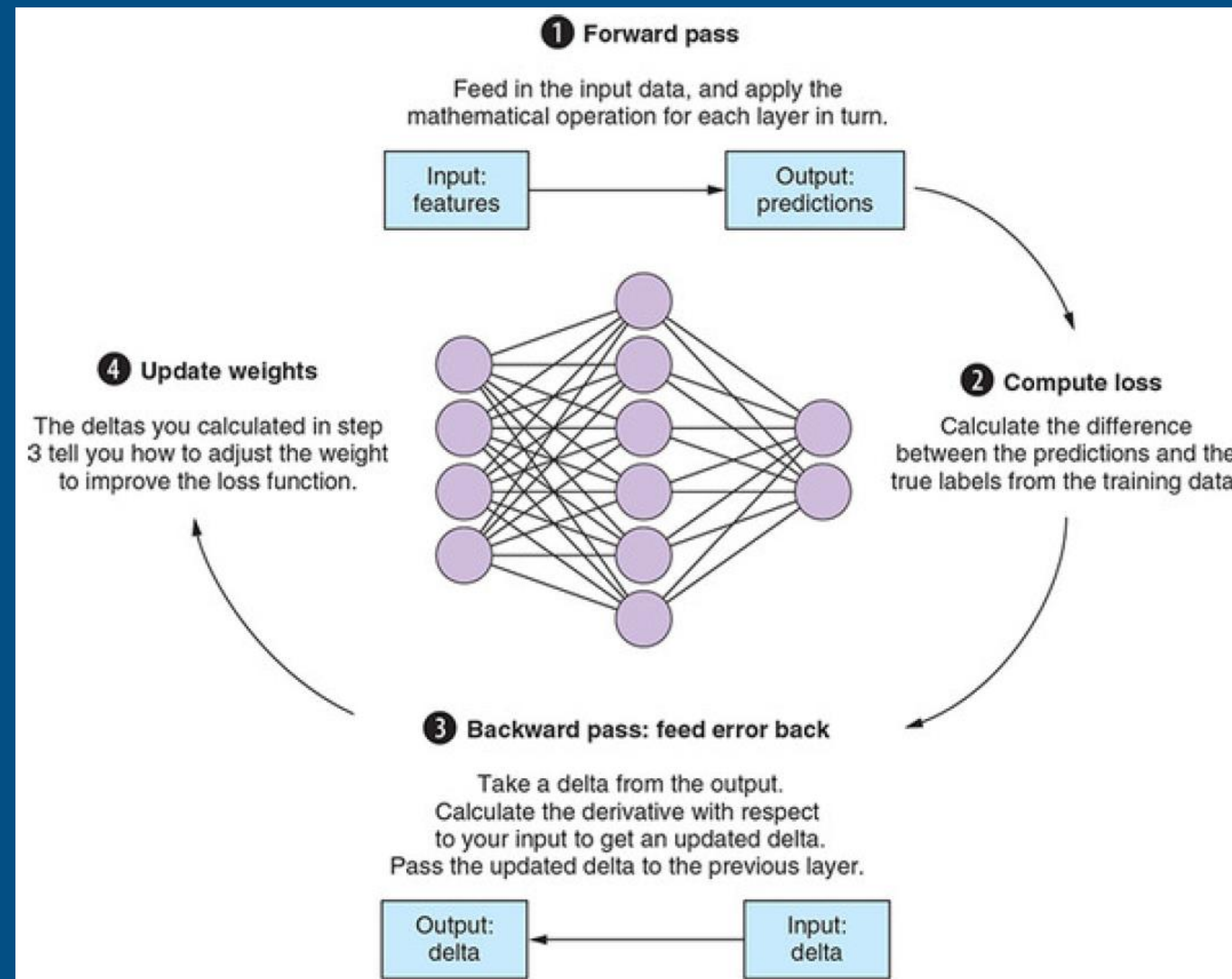
- 확률적 경사하강법 (Stochastic Gradient Descent, SGD)
 - 학습률의 중요성



- 확률적 경사하강법 (Stochastic Gradient Descent, SGD)
 - 일반 경사하강법과 확률적 경사하강법의 차이



- 신경망에 기울기를 역으로 전파하기
 - 역전파(Backpropagation) 알고리즘 = 기울기를 구하는 알고리즘



- 신경망 훈련의 전체 흐름

1. 훈련 데이터를 순방향으로 전달한다. 이 단계에서 입력 데이터 샘플 x 를 신경망에 통과시켜 다음과 같이 예측값에 도달하도록 해야 한다.

- a. 아핀 선형 함수 $Wx + b$ 를 구한다.
- b. 결과에 시그모이드 함수 $\sigma(x)$ 를 적용한다. 이때 계산 단계에서의 x 는 이전 단계의 출력값을 의미한다.
- c. 이 두 단계를 출력층에 도달할 때까지 반복한다.

이 예시에서는 두 개 층을 사용했지만 실제 층수는 얼마가 되든 상관없다.

2. 손실 함수를 평가한다.

이 단계에서는 샘플 x 의 라벨 y 와 예측값 \hat{y} 간의 손실값을 구하는 식으로 이 둘을 비교한다.

- 신경망 훈련의 전체 흐름

3. 오차항을 역으로 전파한다. 이 단계에서는 손실값을 신경망에 역으로 전달한다.

층 방향으로 미분을 구하는 방식으로 이를 실행하며 연쇄법칙으로 가능하다.

순방향 전달을 통해 입력 데이터를 신경망에서 한 방향으로 전달하는 동안

역방향 전달을 통해 오차항을 반대 방향으로 전달한다.

- 오차항 Δ 는 순방향 전달의 역순으로 전파한다.
- Δ 의 초기값으로 사용할 손실 함수의 미분(함수 형태)을 구한다.
순방향 전달 과정에서는 전파되는 오차항을 Δ 라고 부른다는 것을 다시 한 번 확인해서 틀리지 않도록 하자.
- 입력값에 대한 시그모이드 함수의 미분값을 구한다. 이는 간단히 $\sigma \cdot (1 - \sigma)$ 로 나타낸다.
다음 층으로 Δ 를 전달하려면 $\sigma(1 - \sigma) \cdot \Delta$ 식으로 요소 간의 곱을 구하면 된다.
- 그 결과 x 에 대한 아핀 선형 변환 $Wx + b$ 의 미분값이 간단히 W 로 나온다. Δ 를 전달하려면 $W^T \cdot \Delta$ 를 구한다.
- 신경망의 첫 층에 도달할 때까지의 c와 d 두 단계를 반복한다.

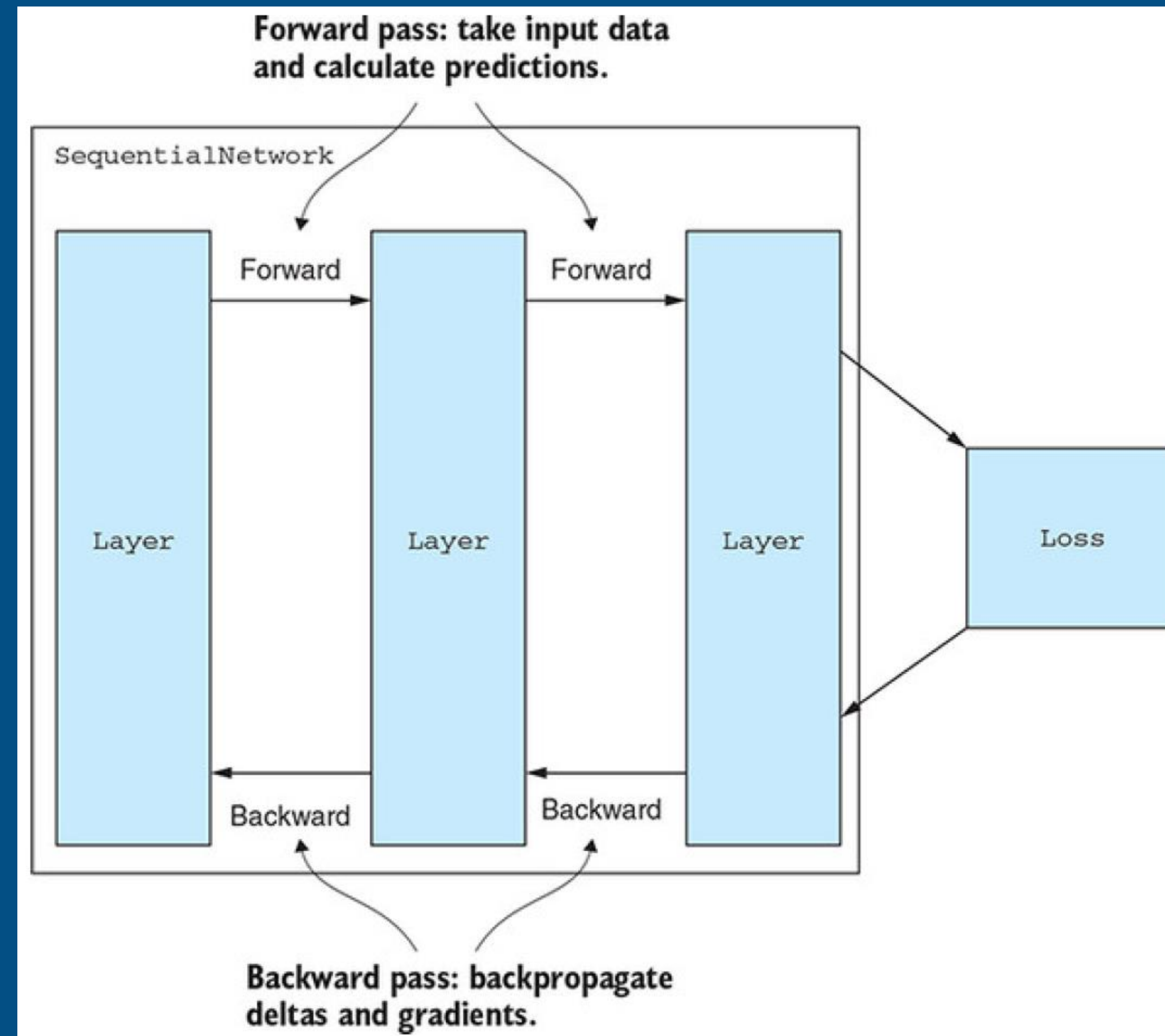
- 신경망 훈련의 전체 흐름

4. 1차 미분 정보로 가중치를 갱신한다. 첫 단계에서는

신경망 파라미터(가중치와 편향치) 갱신 과정에서 구해진 변화량을 사용한다.

- a. 시그모이드 함수에는 파라미터가 없으므로 여기서는 따로 해야 할 일이 없다.
- b. 각 층의 편향치에 도달하는 갱신된 Δb 의 값은 간단히 Δ 다.
- c. 각 층의 가중치에 대한 갱신된 ΔW 의 값은 $\Delta \cdot x^T$ 이다.
(변화량을 곱하기 전에 x 를 전치해주어야 한다.)
- d. 한편 x 에는 단일 데이터를 쓴다. 하지만 여기서 이야기한 모든 내용은 미니배치로 전달된다.
만약 x 가 샘플의 미니배치라면(x 는 모든 열이 입력 벡터인 행렬이다) 순방향과 역방향 전달에서의 계산은 완전히 동일해 보일 것이다.

- 순방향 신경망을 Python으로 구현할 때의 클래스 다이어그램



- Python에서의 신경망층
 - 기본 층 구현

```
import numpy as np

class Layer:
    def __init__(self):
        self.params = []

        self.previous = None
        self.next = None

        self.input_data = None
        self.output_data = None

        self.input_delta = None
        self.output_delta = None
```

- Python에서의 신경망층
 - 층에 선제층과 후속층 연결

```
def connect(self, layer):  
    self.previous = layer  
    layer.next = self
```

- Python에서의 신경망층
 - 순차 신경망의 층에서의 순방향 및 역방향 전달

```
def forward(self):
    raise NotImplementedError

def get_forward_input(self):
    if self.previous is not None:
        return self.previous.output_data
    else:
        return self.input_data

def backward(self):
    raise NotImplementedError

def get_backward_input(self):
    if self.next is not None:
        return self.next.output_delta
    else:
        return self.input_delta

def clear_deltas(self):
    pass

def update_params(self, learning_rate):
    pass

def describe(self):
    raise NotImplementedError
```

- Python에서의 활성화층
 - 시그모이드 함수의 미분 구현



```
def sigmoid_prime_double(x):  
    return sigmoid_double(x) * (1 - sigmoid_double(x))  
  
def sigmoid_prime(x):  
    return np.vectorize(sigmoid_prime_double)(x)
```

- Python에서의 활성화층
 - 시그모이드 활성화층

```
class ActivationLayer(Layer):
    def __init__(self, input_dim):
        super(ActivationLayer, self).__init__()

        self.input_dim = input_dim
        self.output_dim = input_dim

    def forward(self):
        data = self.get_forward_input()
        self.output_data = sigmoid(data)

    def backward(self):
        delta = self.get_backward_input()
        data = self.get_forward_input()
        self.output_delta = delta * sigmoid_prime(data)

    def describe(self):
        print("|-- " + self.__class__.__name__)
        print(" |-- dimensions: ({}, {})".format(self.input_dim, self.output_dim))
```


- Python에서의 활성화층
 - 순방향 신경망의 구성 요소로 Python에서의 밀집층

```
class DenseLayer(Layer):
    def __init__(self, input_dim, output_dim):
        super(DenseLayer, self).__init__()

        self.input_dim = input_dim
        self.output_dim = output_dim

        self.weight = np.random.randn(output_dim, input_dim)
        self.bias = np.random.randn(output_dim, 1)

        self.params = [self.weight, self.bias]

        self.delta_w = np.zeros(self.weight.shape)
        self.delta_b = np.zeros(self.bias.shape)
```

- Python에서의 활성화층
 - 밀집층의 순방향 전달

```
def forward(self):  
    data = self.get_forward_input()  
    self.output_data = np.dot(self.weight, data) + self.bias
```

- Python에서의 활성화층
 - 밀집층의 역방향 전달

```
def backward(self):  
    data = self.get_forward_input()  
    delta = self.get_backward_input()  
  
    self.delta_b += delta  
    self.delta_w += np.dot(delta, data.transpose())  
  
    self.output_delta = np.dot(self.weight.transpose(), delta)
```

- Python에서의 활성화층
 - 밀집층 가중치 갱신 메커니즘

```
def update_param(self):
    self.weight -= rate * self.delta_w
    self.bias -= rate * self.delta_b

def clear_deltas(self):
    self.delta_w = np.zeros(self.weight.shape)
    self.delta_b = np.zeros(self.bias.shape)

def describe(self):
    print("|--- " + self.__class__.__name__)
    print("  |-- dimensions: ({}, {})"
          .format(self.input_dim, self.output_dim))
```

- Python으로 순차 신경망 만들기
 - 순차 신경망 초기화

```
class SequentialNetwork:
    def __init__(self, loss=None):
        print("Initialize Network...")
        self.layers = []
        if loss is None:
            self.loss = MSE()
```

- Python으로 순차 신경망 만들기
 - 순차적으로 층 추가

```
def add(self, layer):  
    self.layers.append(layer)  
    layer.describe()  
    if len(self.layers) > 1:  
        self.layers[-1].connect(self.layers[-2])
```


- Python으로 순차 신경망 만들기
 - 순차 신경망의 `train()` 메서드

```
def train(self, training_data, epochs, mini_batch_size,
          learning_rate, test_data=None):
    n = len(training_data)
    for epoch in range(epochs):
        random.shuffle(training_data)
        mini_batches = [
            training_data[k:k + mini_batch_size] for
            k in range(0, n, mini_batch_size)
        ]
        for mini_batch in mini_batches:
            self.train_batch(mini_batch, learning_rate)
        if test_data:
            n_test = len(test_data)
            print("Epoch {0}: {1} / {2}"
                  .format(epoch, self.evaluate(test_data), n_test))
        else:
            print("Epoch {0} complete".format(epoch))
```

- Python으로 순차 신경망 만들기
 - 신경망에서 규칙 및 순방향 전달과 역방향 전달 갱신하기

```
def update(self, mini_batch, learning_rate):
    learning_rate = learning_rate / len(mini_batch)
    for layer in self.layers:
        layer.update_params(learning_rate)
    for layer in self.layers:
        layer.clear_deltas()

def forward_backward(self, mini_batch):
    for x, y in mini_batch:
        self.layers[0].input_data = x
        for layer in self.layers:
            layer.forward()
        self.layers[-1].input_delta = \
            self.loss.loss_derivative(self.layers[-1].output_data, y)
        for layer in reversed(self.layers):
            layer.backward()
```

- Python으로 순차 신경망 만들기
 - 평가

```
def single_forward(self, x):  
    self.layers[0].input_data = x  
    for layer in self.layers:  
        layer.forward()  
    return self.layers[-1].output_data  
  
def evaluate(self, test_data):  
    test_results = [(  
        np.argmax(self.single_forward(x)),  
        np.argmax(y)  
    ) for (x, y) in test_data]  
    return sum(int(x == y) for (x, y) in test_results)
```

- 신경망으로 손글씨 숫자 분류하기
 - 신경망 인스턴스화

```
from dlgo.nn import load_mnist
from dlgo.nn import network
from dlgo.nn.layers import DenseLayer, ActivationLayer

training_data, test_data = load_mnist.load_data()

net = network.SequentialNetwork()

net.add(DenseLayer(784, 392))
net.add(ActivationLayer(392))
net.add(DenseLayer(392, 196))
net.add(ActivationLayer(196))
net.add(DenseLayer(196, 10))
net.add(ActivationLayer(10))
```

- 신경망으로 손글씨 숫자 분류하기
 - 훈련 데이터를 적용한 신경망 인스턴스 실행



```
net.train(training_data, epochs=10, mini_batch_size=10,  
          learning_rate=3.0, test_data=test_data)
```

감사합니다!

스터디 듣느라 고생 많았습니다.