



Deep Learning

Summary

◆ Neural Networks

- What is Neural Networks ?

◆ Architecture of Neural Network

- Activation Function : Step function / Sigmoid function / ReLU function
- Output Layer : Identity function / Softmax function
- Implementation

◆ Training Neural Network

- Overview & Data
- Loss function : Mean Squared Error / Cross Entropy Error
- Gradient : Gradient Descent
- Implementation

Outline

I

Neural Networks



II

Architecture of Neural Network



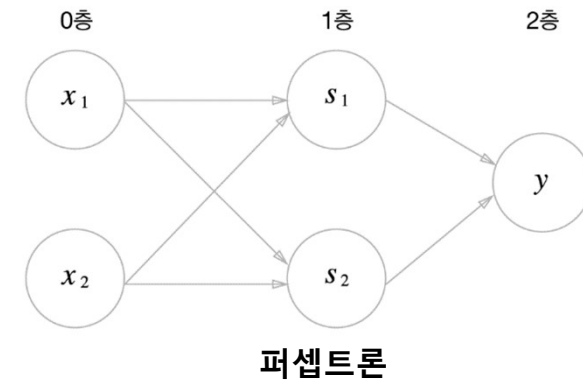
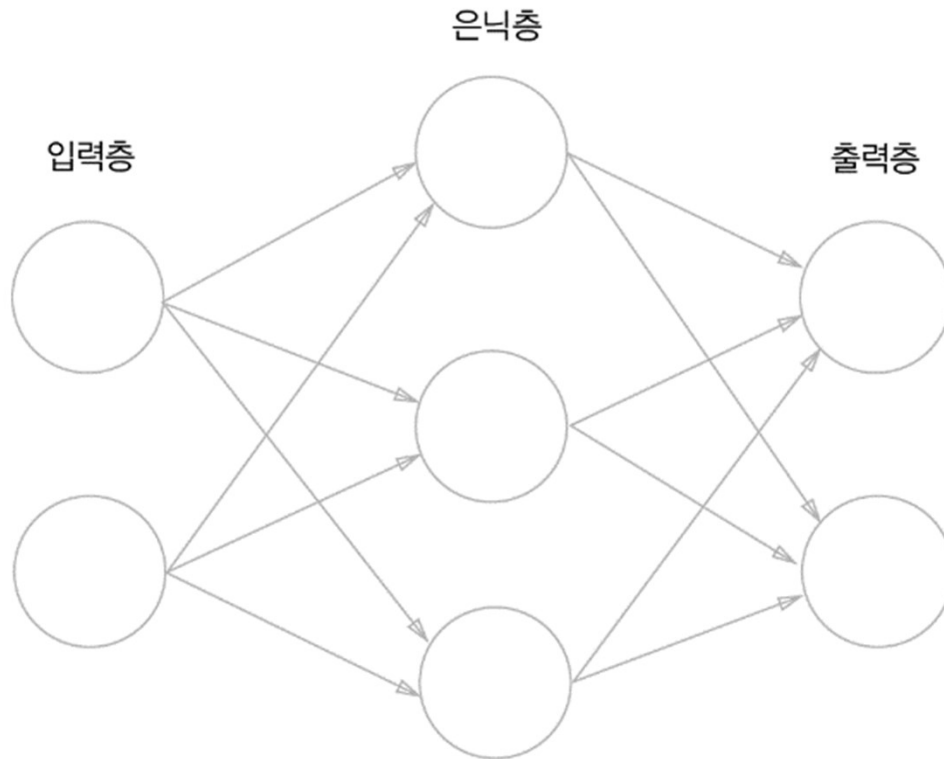
III

Training Neural Network



Neural Networks

◆ Neural Network



- 입력층(Input layer) : NN의 **입력**을 나타내는 층
- 출력층(Output layer) : NN의 **수행 결과**를 나타내는 층
- 은닉층(Hidden layer) : 사람의 눈에 **보이지 않는** 층



II

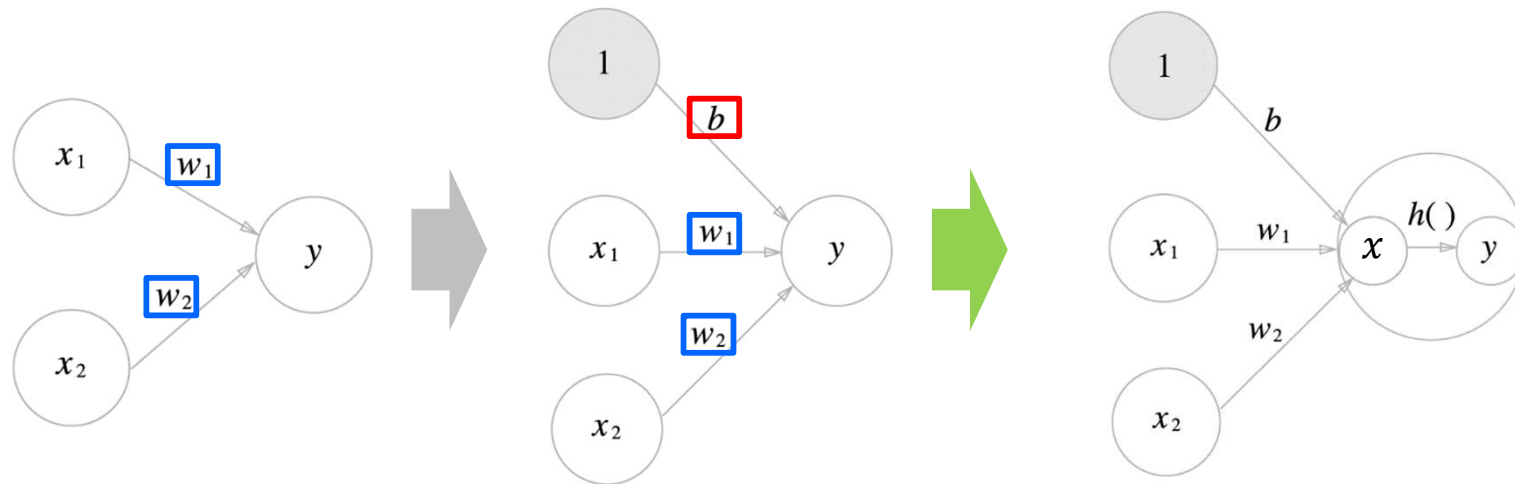
Architecture of Neural Network



- Activation Function
- Output Layer
- Implementation

Activation Function (1/5)

◆ Perceptron



$$y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (\underline{b} + \underline{w_1}x_1 + \underline{w_2}x_2 > 0) \end{cases}$$

편향(bias) : 뉴런의 활성화를 제어

가중치(weight) : 각 신호의 영향력을 제어

$$y = h(b + w_1x_1 + w_2x_2)$$

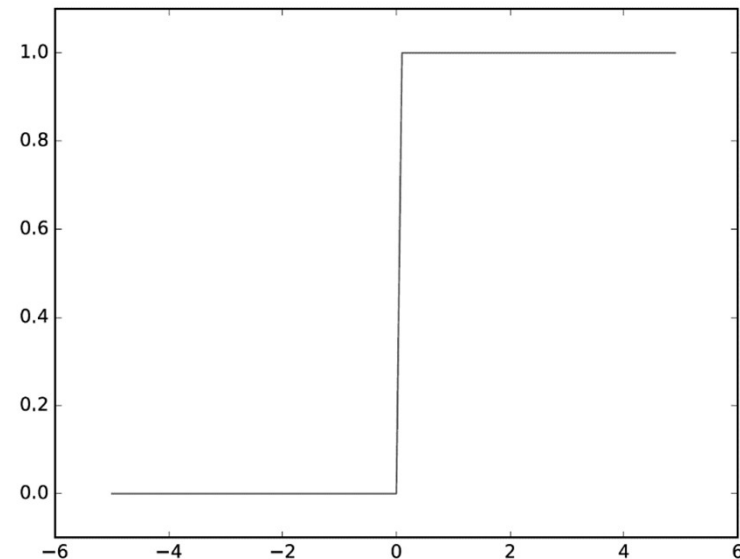
$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

입력 신호의 총합을 출력 신호로 변환
= **활성화 함수(Activation Function)**

Activation Function (2/5)

◆ Step Function

$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

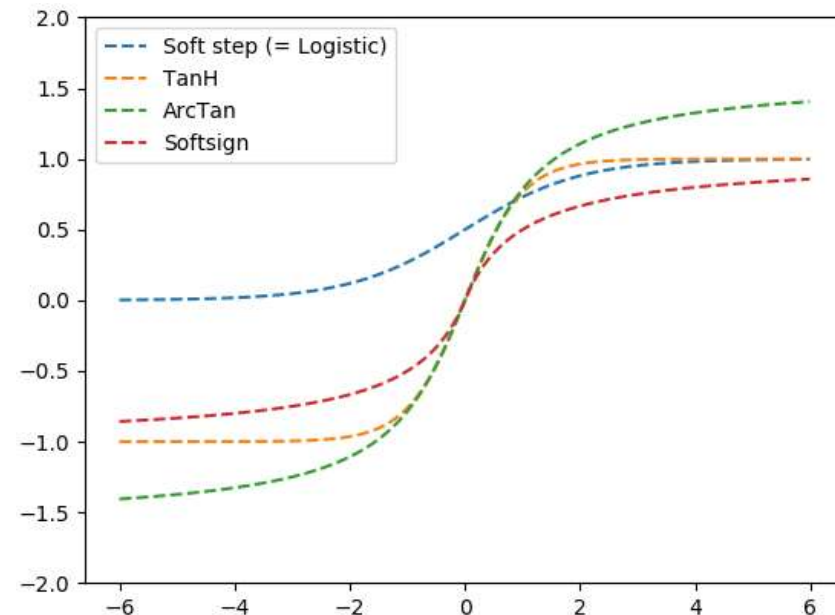


- 선형 함수의 결과를 **이진 분류기(비선형)**로 나타내기 위한 함수
- 입력이 **특정 값(0)**을 넘으면 **1**을 출력, 그 이외에는 **0**을 출력
- 주로 **퍼셉트론**에서 사용하는 활성화 함수

Activation Function (3/5)

◆ Sigmoid Function

$$h(x) = \frac{1}{1 + \exp(-x)}$$



- 선형 함수의 결과를 0~1 까지의 **비선형 형태로** 변형하기 위한 함수
- 신경망 초기 모델에 자주 사용됨
 - 양 극단에 가까울수록 변화량이 작아져 학습이 매우 더디게 진행 됨
 - 깊이가 깊어질수록 중앙보다 양극으로 쏠리는 형태이므로 데이터 양극화 발생
 - exp 계산 과정이 다른 선형함수들에 비해 매우 느림

Activation Function (4/5)

◆ Step Function & Sigmoid Function

- 두 함수 모두 비선형 함수
 - 신경망에서는 활성화 함수로 비선형 함수를 사용해야 함

$$h(x) = cx$$

$$h(x) = c^3x$$

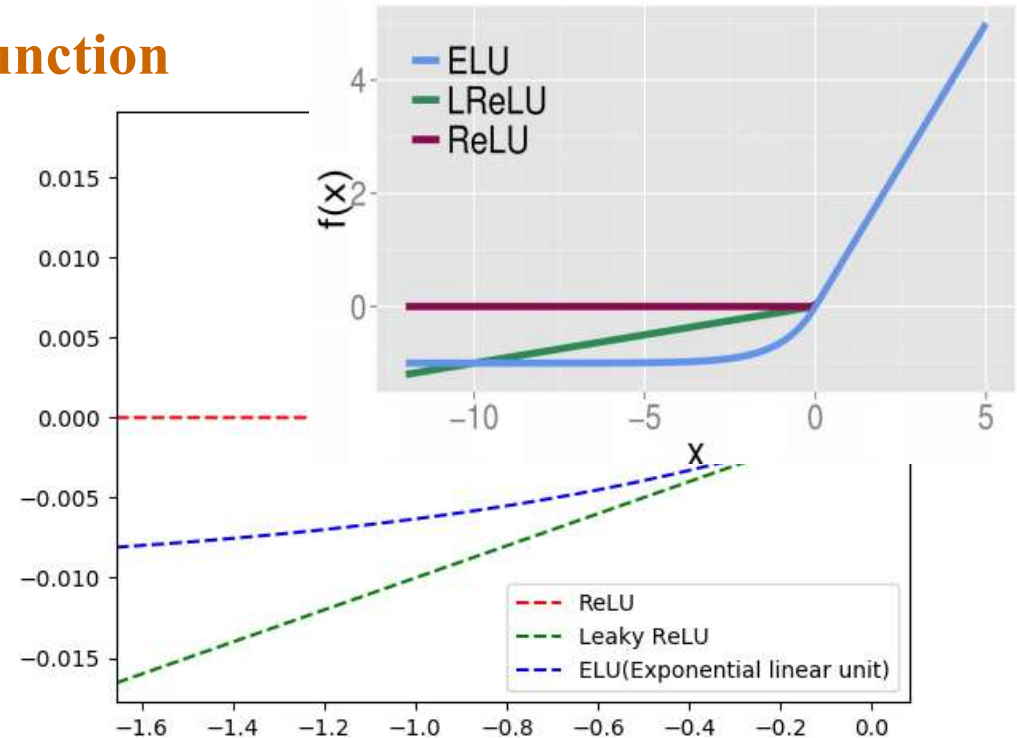
$$y(x) = h(h(h(x))) \rightarrow y(x) = c * c * c * x$$

- 두 함수는 유사한 동작을 수행
 - 입력이 작을 때의 출력은 0에 가깝고, 입력이 클 때 출력이 1에 가까워지는 구조
 - 입력이 아무리 작거나 커도 출력은 0과 1사이의 값을 반환함

Activation Function (5/5)

◆ ReLU(Rectified Linear Unit) Function

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$



- 선형 함수의 결과를 선형 형태로 나타내기 위한 함수
- 입력이 특정 값(0)을 넘으면 입력을 그대로 출력, 그 이외에는 0을 출력
 - 깊이가 깊어질수록 비선형 함수와 비슷한 모습을 함
 - 계산과정이 단순해 다른 선형함수들에 비해 매우 빠름
 - 특정 노드의 가중치가 0이 되면 학습이 완료될 때까지 0으로 남음

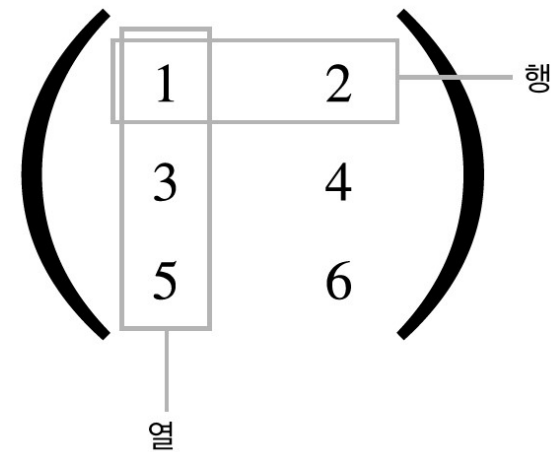
다차원 배열의 계산

◆ 다차원 배열

```
import numpy as np
A = np.array([1,2,3,4])
B = np.array([[1,2], [3,4], [5,6]])
np.ndim(B)
B.Shape
```

```
A = np.array([[1,2], [3,4]])
B = np.array([[5,6], [7,8]])
np.dot(A, B)
```

(1 2 3 4)



$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

A

B

$1 \times 5 + 2 \times 7$

$3 \times 5 + 4 \times 7$

Output Layer (1/2)

◆ Overview

- **분류(Classification) v.s. 회귀(Regression)**

- **분류** : 데이터가 어느 집단(Class)에 속하는지

- 예1) 사진 속 인물의 성별을 분류
- 예2) 사진 속 동물의 종을 분류



Softmax function

- **회귀** : 입력 데이터에서 수치를 예측

- 예1) 사진 속 인물의 몸무게를 예측
- 예2) 사진 속 동물의 크기를 예측

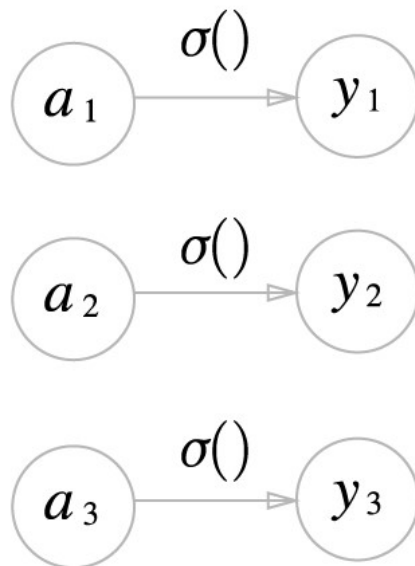


Identity function

Output Layer (2/2)

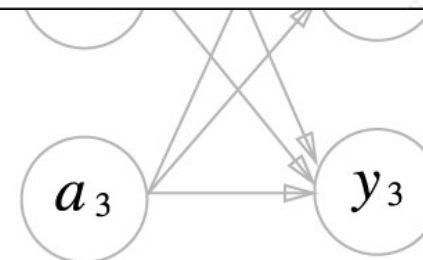
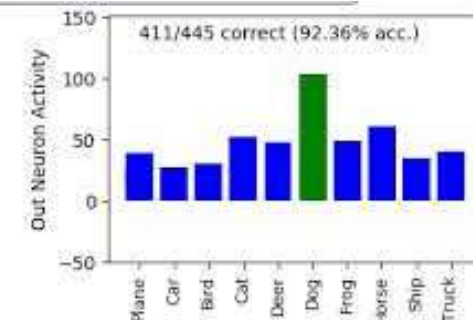
◆ Identity function & Softmax function

- Identity function



- 입력(Input)을 그대로 출력

- Softmax function



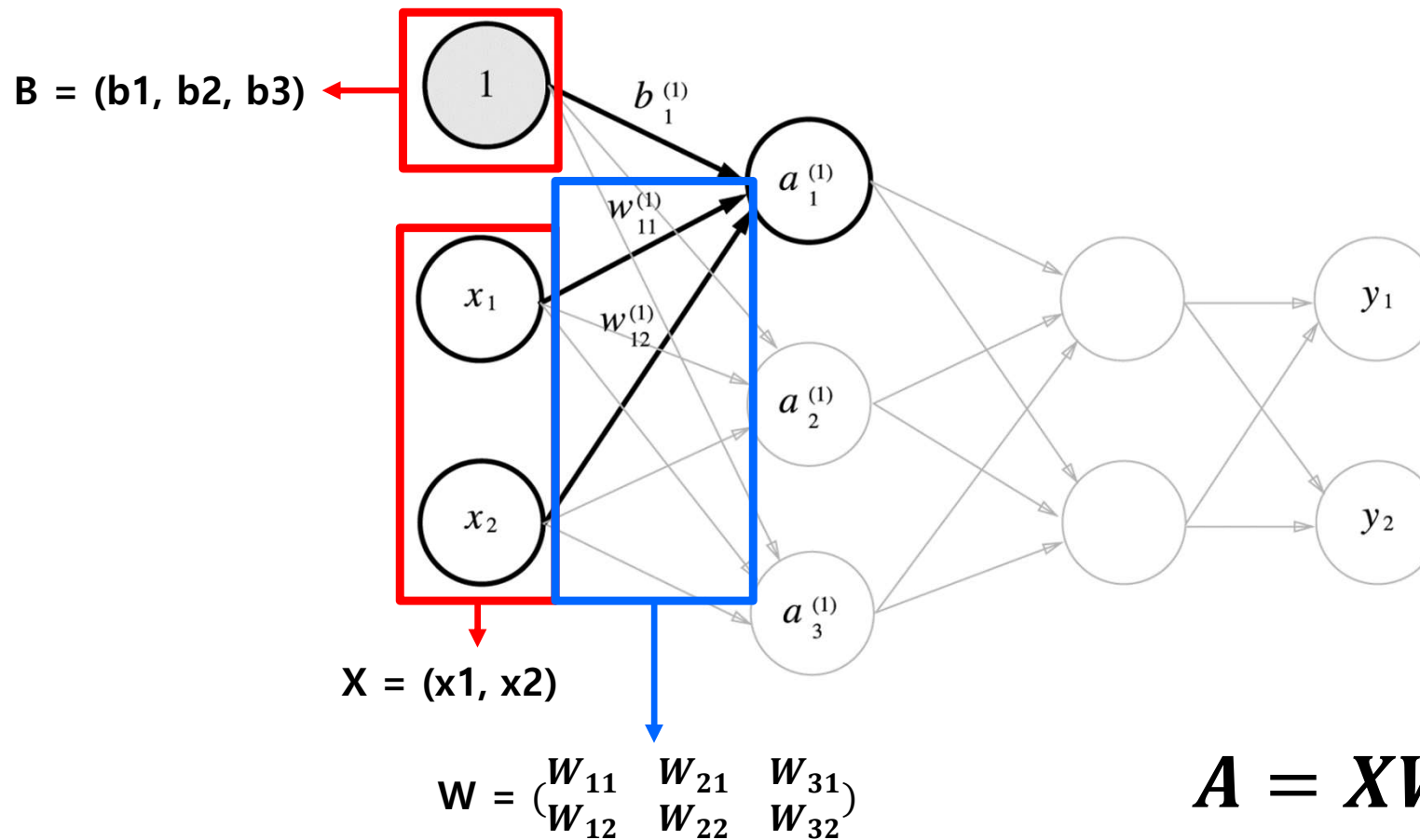
$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$

- 모든 입력(Input)이 출력에 영향을 미침
- 0과 1.0 사이의 실수를 반환함 (출력의 총합은 1)

> 확률로 해석이 가능

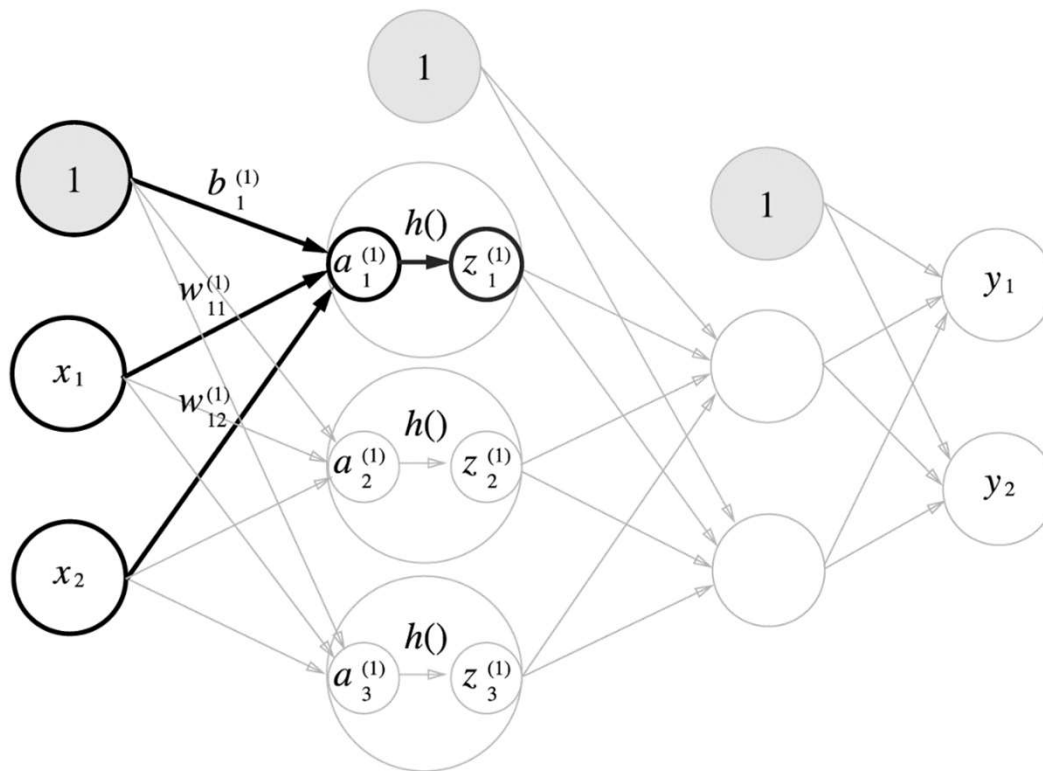
Implementation (1/7)

◆ 3층 신경망 구현하기



Implementation (2/7)

◆ 3층 신경망 구현하기



```
import numpy as np
X = np.array([1.0, 0.5])
W1 = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
B1 = np.array([0.1, 0.2, 0.3])
```

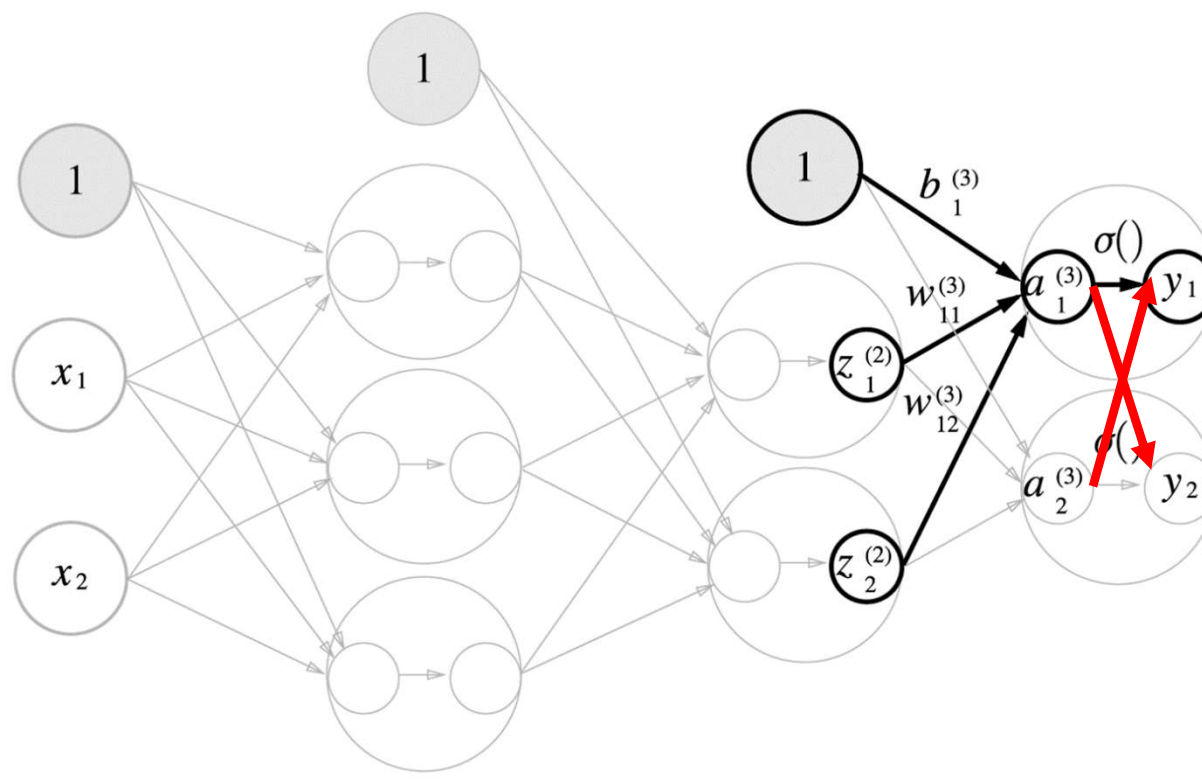
```
A1 = np.dot(X, W1) + B1
print(A1)
Z1 = sigmoid(A1)
print(Z1)
```

[0.1, 0.7, 1.1]

[0.57444252, 0.66818777, 0.75026011]

Implementation (3/7)

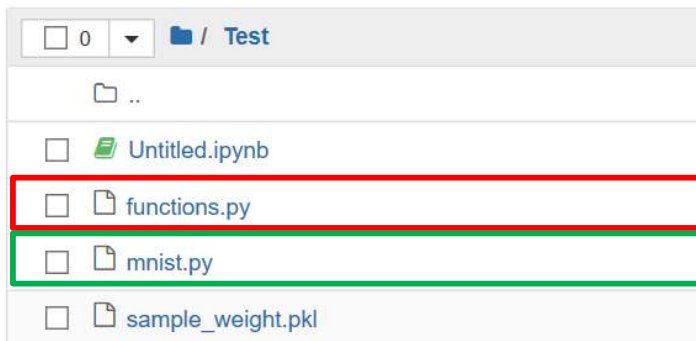
◆ 3층 신경망 구현하기



Implementation (4/7)

◆ MNIST(손글씨 숫자) 인식

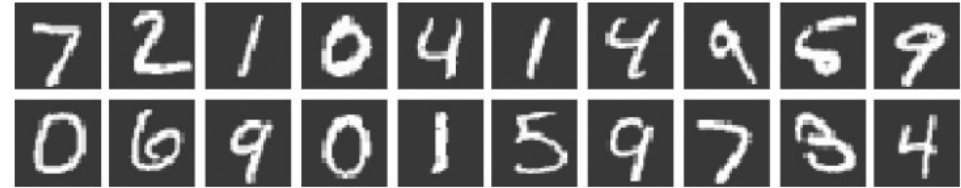
- Step 1. Upload some files



```
def sigmoid(x):  
  
def load_mnist(normalize=True, flatten=True, one_hot_label=False):  
    """MNIST 데이터셋 읽기  
  
    Parameters  
    -----  
    normalize : 이미지의 픽셀 값을 0.0~1.0 사이의 값으로 정규화할지 정한다.  
    one_hot_label :  
        one_hot_label이 True면, 레이블을 원-핫(one-hot) 배열로 돌려준다.  
        one-hot 배열은 예를 들어 [0,0,1,0,0,0,0,0,0,0]처럼 한 원소만 1인 배열이다.  
    flatten : 입력 이미지를 1차원 배열로 만들지를 정한다.  
  
    Returns  
    -----  
    (훈련 이미지, 훈련 레이블), (시험 이미지, 시험 레이블)  
    """  
    if not os.path.exists(save_file):  
        init_mnist()  
  
    with open(save_file, 'rb') as f:  
        dataset = pickle.load(f)  
  
    if normalize:  
        for key in ('train_img', 'test_img'):  
            dataset[key] = dataset[key].astype(np.float32)  
            dataset[key] /= 255.0  
  
    if one_hot_label:  
        dataset['train_label'] = _change_one_hot_label(dataset['train_label'])  
        dataset['test_label'] = _change_one_hot_label(dataset['test_label'])  
  
    if not flatten:  
        for key in ('train_img', 'test_img'):  
            dataset[key] = dataset[key].reshape(-1, 1, 28, 28)  
  
    return (dataset['train_img'], dataset['train_label']), (dataset['test_img'], dataset['test_label'])
```

Implementation (5/7)

◆ MNIST(손글씨 숫자) 인식



- Step 2. Import libraries

```
In [1]: # coding: utf-8
import sys, os
import numpy as np
import pickle
from mnist import load_mnist
from functions import sigmoid, softmax
```

- Step 3. Define functions

```
In [2]: def get_data():
        (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True,
        one_hot_label=False)
        return x_test, t_test
```

```
In [2]: def init_network():
        with open("sample_weight.pkl", 'rb') as f:
            network = pickle.load(f)
        return network
```

Implementation (6/7)

◆ MNIST(손글씨 숫자) 인식

- Step 3. Define functions



```
In [4] : def predict(network, x):
          W1, W2, W3 = network['W1'], network['W2'], network['W3']
          b1, b2, b3 = network['b1'], network['b2'], network['b3']

          a1 = np.dot(x, W1) + b1
          z1 = sigmoid(a1)
          a2 = np.dot(z1, W2) + b2
          z2 = sigmoid(a2)
          a3 = np.dot(z2, W3) + b3
          y = softmax(a3)

          return y
```

Implementation (7/7)

◆ MNIST(손글씨 숫자) 인식



- Step 4. Initialization

```
In [5]: x, t = get_data()
        network = init_network()
```

```
Out [5]: Downloading train-images-idx3-ubyte.gz ... Done
          Downloading train-labels-idx1-ubyte.gz ... Done
          Downloading t10k-images-idx3-ubyte.gz ... Done
          Downloading t10k-labels-idx1-ubyte.gz ... Done
          Converting train-images-idx3-ubyte.gz to NumPy Array ... Done
          Converting train-labels-idx1-ubyte.gz to NumPy Array ... Done
          Converting t10k-images-idx3-ubyte.gz to NumPy Array ... Done
          Converting t10k-labels-idx1-ubyte.gz to Numpy Array ... Done
          Creating pickle file ...
```

- Step 5. Test

```
In [5]: accuracy_cnt = 0
        for i in range(len(x)):
            y = predict(network, x[i])
            p = np.argmax(y) # 확률이 가장 높은 원소의 인덱스를 얻는다.
            if p == t[i]:
                accuracy_cnt += 1
        print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

```
Out [5]: Accuracy:0.9352
```



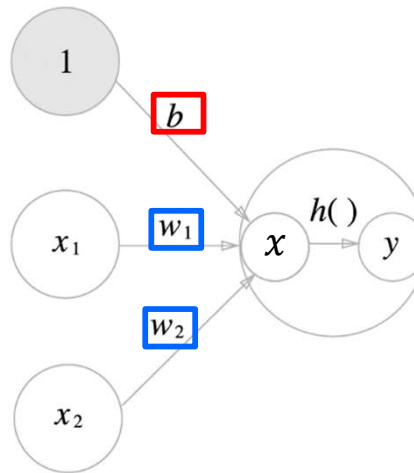

Training Neural Network



- Overview & Data
- Loss function
- Gradient
- Implementation

Overview & Data (1/3)

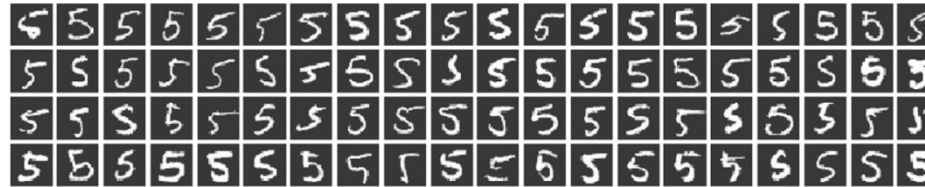
◆ Training (학습)



- 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 것
손실 함수(Loss Function) 의 값이 작다.
 ↳ 경사 하강법 (Gradient Descent)

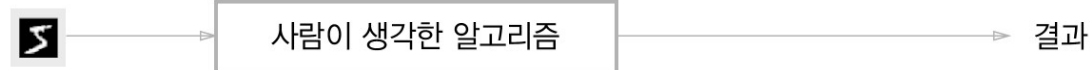
Overview & Data (2/3)

◆ 데이터 주도 학습



• 사람 중심 접근

- 경험과 직관을 바탕으로 시행착오를 거듭하며 수행



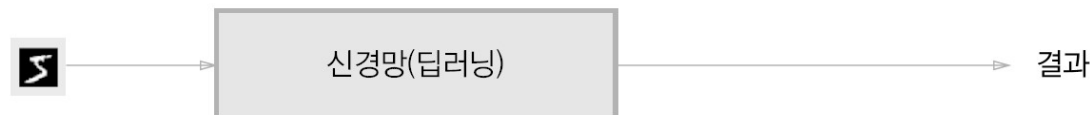
• 데이터 주도 학습

- 사람의 개입을 최소화하고 수집한 데이터로부터 패턴을 찾으려 시도
- 기계 학습
 - 이미지에서 특징을 추출하고 특징의 패턴을 기계학습 기술로 학습



- 신경망

- 이미지에 포함된 중요한 특징까지 스스로 학습



Overview & Data (3/3)

◆ Training data vs Test data

dataset	Training data	Test data
MNIST	60000	10000
CIFAR-10	50000	10000
KDD99	4898430	311029

모델의 범용성을 평가하기 위해 사용



- 오버피팅(Overfitting)

- 훈련 데이터만 잘 판별하고 시험 데이터는 잘 판별하지 못하는 현상

Loss Function (1/3)

◆ Overview

- 최적의 매개변수 값(weight, bias)을 탐색하기 위한 지표
- 평균 제곱 오차(MSE), 교차 엔트로피 오차(Cross-Entropy Error) 가 주로 사용됨

- 평균 제곱 오차(Mean Squared Error, MSE)

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Class 수 실제 결과 신경망의 예측 결과

0.00975...

y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

0.5975...

- 교차 엔트로피 오차(Cross Entropy Error, CEE)

$$E = - \sum_k t_k \log y_k$$

Class 수 실제 결과 신경망의 예측 결과

0.51082..

y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

2.30258

Loss Function (2/3)

◆ Loss Function in Training

- 앞서 설명한 내용은 하나의 데이터에 대한 손실 함수
- 실제로는 **훈련 데이터 모두**에 대한 손실 함수를 사용해야함

• 교차 엔트로피 오차(Cross Entropy Error, CEE)

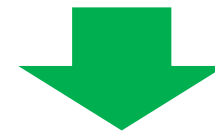
$$E = -\frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk}$$

Diagram illustrating the components of the Cross Entropy Error formula:

- N : 학습 데이터 수 (Learning data count)
- n : Class 수 (Class count)
- t_{nk} : 실제 결과 (Actual result)
- y_{nk} : 신경망의 예측 결과 (Neural network prediction result)

dataset	Training data
MNIST	60000
CIFAR-10	50000
KDD99	4898430

한 번에 처리하기에 너무 많은 양



미니배치(Mini-batch)

Loss Function (3/3)

◆ The reason for using the loss function


- 손실 함수의 궁극적 목적은 신경망이 높은 **정확도**를 나타내도록 매개 변수를 찾는 것
- 실제 신경망 학습에서는 **손실 함수의 미분(기울기)을 계산**하고 결과에 따라 **매개 변수를 갱신**
 - 기울기 < 0 : 매개변수를 **양**의 방향으로 변화
 - 기울기 > 0 : 매개변수를 **음**의 방향으로 변화
 - 기울기 $= 0$: 매개변수 갱신 중지

• 정확도 지표

- 훈련 데이터 : 100개


32%  32.036%
매개 변수 갱신

32%  70%
매개 변수 갱신

32%  32% **기울기 = 0**
매개 변수 갱신

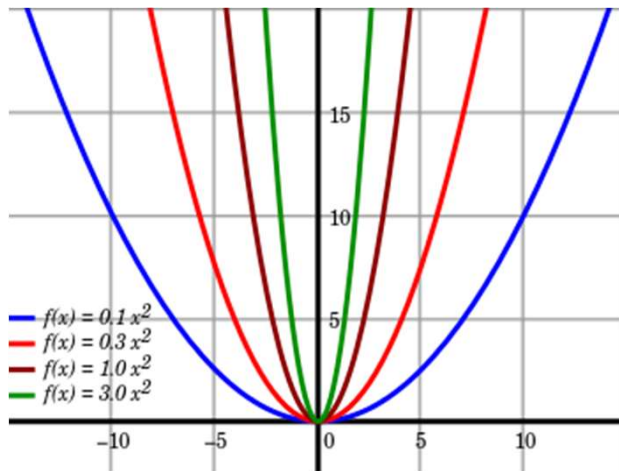
• 손실 함수 지표

- 훈련 데이터 : 100개

0.92543  0.93432 **기울기 != 0**
매개 변수 갱신

Gradient (1/3)

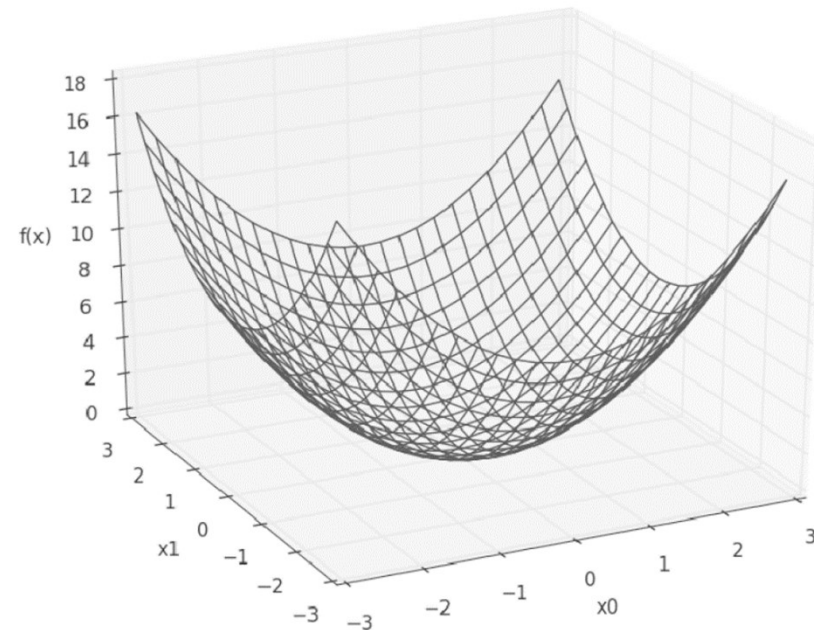
◆ 미분



$$y = ax^2$$



$$\frac{dy}{dx} = 2ax$$



$$f(x_0, x_1) = ax_0^2 + bx_1^2$$



$$\frac{\partial f}{\partial x_0} = 2ax_0 + bx_1^2 \quad \frac{\partial f}{\partial x_1} = ax_0^2 + 2bx_1$$

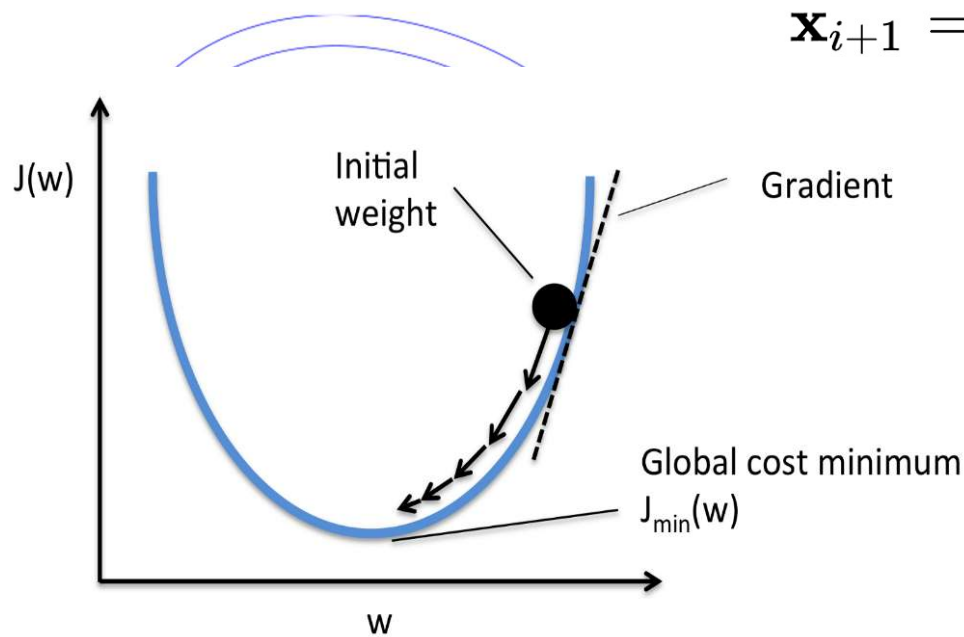
Gradient (2/3)

◆ Gradient Descent(경사 하강법)

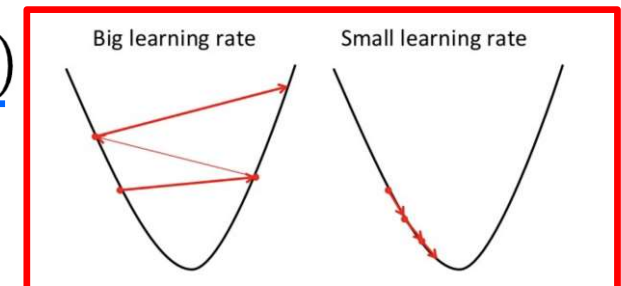
- 실제 신경망 학습에서는 손실 함수의 미분(기울기)을 계산하고 결과에 따라 **매개 변수를 갱신**

- 기울기 < 0 : 매개변수를 **양**의 방향으로 변화
- 기울기 > 0 : 매개변수를 **음**의 방향으로 변화
- 기울기 = 0 : 매개변수 갱신 중지

- 함수의 기울기(경사)를 구하여 **기울기가 낮은 쪽으로 계속 이동**시켜서 **극값**에 이를 때까지 반복시키는 것



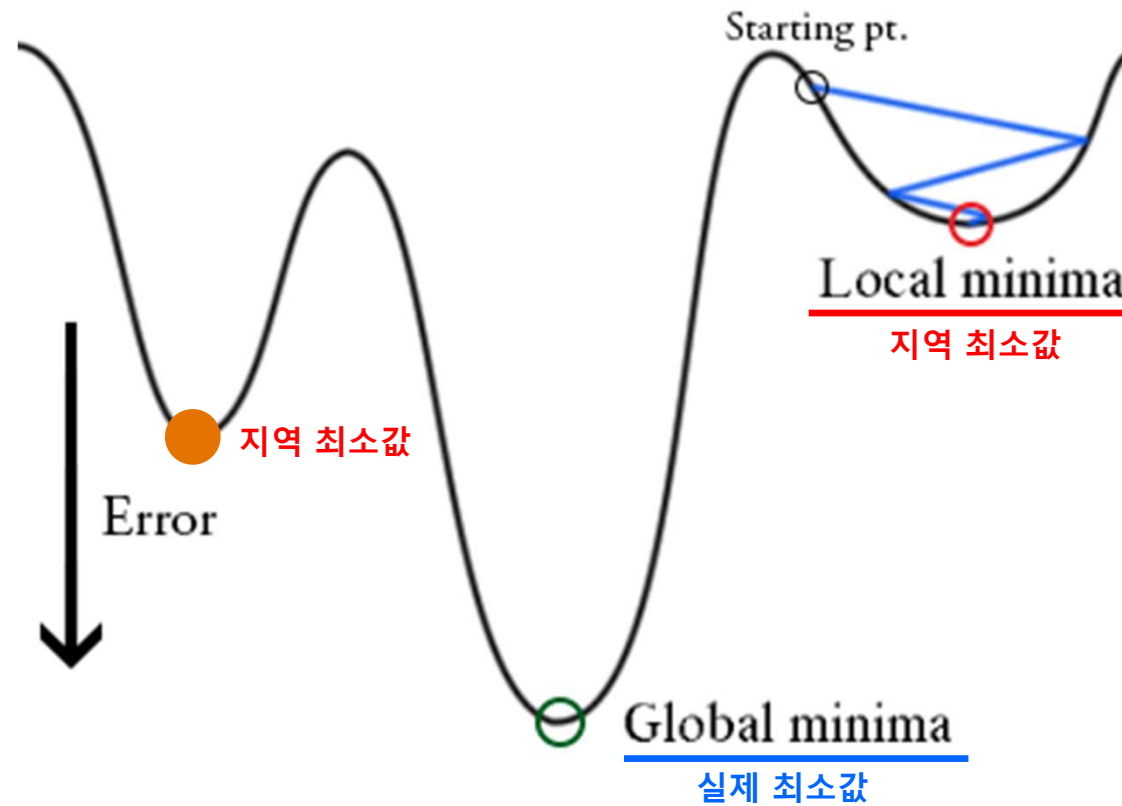
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \underbrace{\gamma_i}_{\text{매개변수}} \underbrace{\nabla f(\mathbf{x}_i)}_{\text{기울기}}$$



Gradient (3/3)

◆ Gradient Descent(경사 하강법)

- **Local Minimum**에 빠질 가능성이 높음
- 매우 느린 **수렴 속도**



Implementation (1/6)

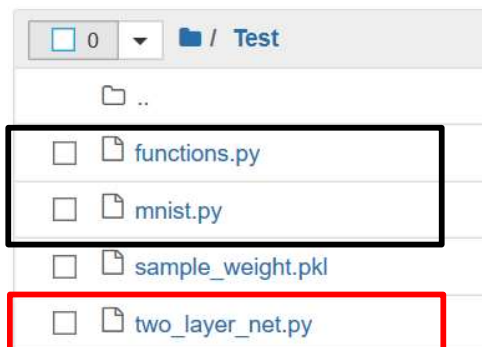
◆ Algorithm

- 1. Mini-batch
 - 훈련 데이터 중 일부를 무작위로 가져옴 → 미니배치
 - 미니배치의 Loss function 값을 줄이는 것이 목표
- 2. 기울기 산출
 - 미니배치의 Loss function 값을 줄이기 위해 가중치 매개변수에 대해 기울기를 계산
- 3. 매개변수 갱신
 - 가중치 매개변수를 기울기 방향으로 아주 조금 갱신
- 4. 반복
 - Step 1 ~ Step 3까지 반복

Implementation (2/6)

◆ MNIST(손글씨 숫자) 인식

- Step 1. Upload some files



class TwoLayerNet:

```
def __init__(self, in_dim, out_dim):  
    # 가중치 초기화  
    self.params = {}  
    self.params['W1'] = np.random.randn(in_dim, out_dim)  
    self.params['b1'] = np.zeros(out_dim)  
    self.params['W2'] = np.random.randn(out_dim, out_dim)  
    self.params['b2'] = np.zeros(out_dim)
```

```
def predict(self, x):  
    W1, W2 = self.params['W1'], self.params['W2']  
    b1, b2 = self.params['b1'], self.params['b2']  
  
    a1 = np.dot(x, W1) + b1  
    z1 = sigmoid(a1)  
    a2 = np.dot(z1, W2) + b2  
    y = softmax(a2)  
  
    return y
```

```
# x : 입력 데이터, t : 정답  
def loss(self, x, t):  
    y = self.predict(x)  
  
    return cross_entropy_loss(y, t)
```

```
def accuracy(self, x, t):  
    y = self.predict(x)  
    y = np.argmax(y, axis=1)  
    t = np.argmax(t, axis=1)  
  
    accuracy = np.sum(y == t) / float(x.shape[0])  
    return accuracy
```

```
def gradient(self, x, t):  
    W1, W2 = self.params['W1'], self.params['W2']  
    b1, b2 = self.params['b1'], self.params['b2']  
    grads = {}  
  
    batch_num = x.shape[0]  
  
    # forward  
    a1 = np.dot(x, W1) + b1  
    z1 = sigmoid(a1)  
    a2 = np.dot(z1, W2) + b2  
    y = softmax(a2)  
  
    # backward  
    dy = (y - t) / batch_num  
    grads['W2'] = np.dot(z1.T, dy)  
    grads['b2'] = np.sum(dy, axis=0)  
  
    da1 = np.dot(dy, W2.T)  
    dz1 = sigmoid_grad(a1) * da1  
    grads['W1'] = np.dot(x.T, dz1)  
    grads['b1'] = np.sum(dz1, axis=0)  
  
    return grads
```

기울기 계산

정확도 계산

Implementation (3/6)

◆ MNIST(손글씨 숫자) 인식



- Step 2. Import libraries

```
In [1] :      # coding: utf-8
            import sys, os
            import numpy as np
            import matplotlib.pyplot as plt
            from mnist import load_mnist
            from two_layer_net import TwoLayerNet
```

- Step 3. Initialization

```
In [2] :      (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True,
                                                                one_hot_label=True)
                                                                네트워크 초기화
            network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
```

```
In [3] :      iters_num = 10000 # 반복 횟수를 적절히 설정한다.
            train_size = x_train.shape[0]
            batch_size = 100 # 미니배치 크기
            learning_rate = 0.1  $\xrightarrow{\text{red arrow}} \mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i \nabla f(\mathbf{x}_i)$ 

            train_loss_list = []
            train_acc_list = []
            test_acc_list = []
```

Implementation (4/6)

◆ MNIST(손글씨 숫자) 인식

- Step 4. Training network



In [4] : `iter_per_epoch = max(train_size / batch_size, 1) # 1에폭당 반복 수`

```
for i in range(iters_num):
```

4. 반복

```
    batch_mask = np.random.choice(train_size, batch_size) # 미니배치 획득
```

```
    x_batch = x_train[batch_mask]
```

```
    t_batch = t_train[batch_mask]
```

1. Mini-batch

```
    grad = network.gradient(x_batch, t_batch) # 기울기 계산
```

2. 기울기 산출

```
    for key in ('W1', 'b1', 'W2', 'b2'): # 매개변수 갱신
```

```
        network.params[key] -= learning_rate * grad[key]
```

3. 매개변수 갱신

```
    loss = network.loss(x_batch, t_batch) # 학습 경과 기록
```

```
    train_loss_list.append(loss)
```

```
    if i % iter_per_epoch == 0: # 1에폭당 정확도 계산
```

```
        train_acc = network.accuracy(x_train, t_train)
```

```
        test_acc = network.accuracy(x_test, t_test)
```

```
        train_acc_list.append(train_acc)
```

```
        test_acc_list.append(test_acc)
```

```
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```


Implementation (5/6)

◆ MNIST(손글씨 숫자) 인식

- Step 4. Training network



Out [4] : train acc, test acc | 0.10441666666666667, 0.1028
train acc, test acc | 0.7906666666666666, 0.7937
train acc, test acc | 0.8779333333333333, 0.8823
train acc, test acc | 0.8979666666666667, 0.9016
train acc, test acc | 0.9078, 0.911
train acc, test acc | 0.9139, 0.9168
train acc, test acc | 0.91855, 0.9185
train acc, test acc | 0.9243333333333333, 0.9266
train acc, test acc | 0.9269333333333334, 0.9283
train acc, test acc | 0.93125, 0.9302
train acc, test acc | 0.934, 0.933
train acc, test acc | 0.9369166666666666, 0.9366
train acc, test acc | 0.939, 0.9386
train acc, test acc | 0.9408333333333333, 0.9386
train acc, test acc | 0.9431, 0.9415
train acc, test acc | 0.9454, 0.943
train acc, test acc | 0.9470333333333333, 0.9457

Implementation (6/6)

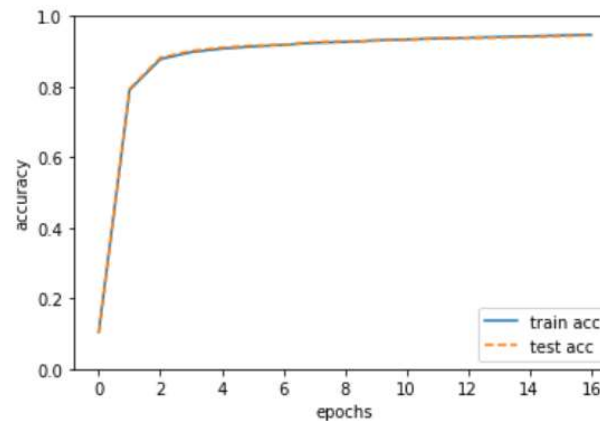
◆ MNIST(손글씨 숫자) 인식


- Step 5. Print result



```
In [5]: markers = {'train': 'o', 'test': 's'}  
x = np.arange(len(train_acc_list))  
plt.plot(x, train_acc_list, label='train acc')  
plt.plot(x, test_acc_list, label='test acc', linestyle='--')  
plt.xlabel("epochs")  
plt.ylabel("accuracy")  
plt.ylim(0, 1.0)  
plt.legend(loc='lower right')  
plt.show()
```

Output [5] :





Thank You !

