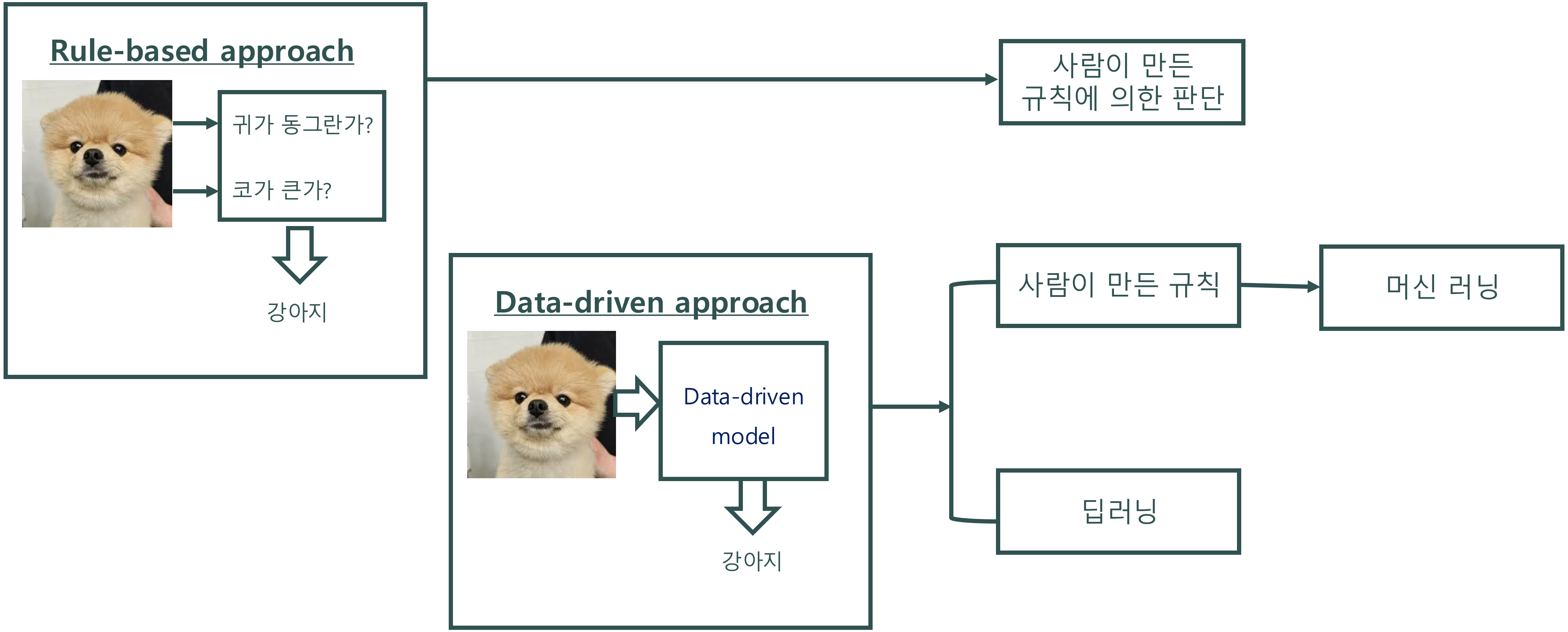
파이썬으로 배우는 딥러닝(Deep Learning)

3회차 수업
손실 함수부터 학습 구현까지, 신경망 훈련 따라잡기

목 차

Rule-based vs. Data-driven

- Rule-based 학습 : 명시적인 규칙을 사용하여 데이터를 분류하거나 예측하는 접근법
- Data-driven 학습 : 대량의 데이터에서 패턴을 학습하고 예측 모델을 생성하는 접근법



손실함수(Loss function)

◆ 손실함수란?

- 모델의 예측값과 실제값 간의 차이를 측정하는 함수
- 모델의 성능을 평가하고 모델의 가중치를 업데이트하여 학습을 진행하는데 중요한 역할을 함.
- 손실함수의 출력은 주어진 데이터셋에 대한 모델의 예측 정확도를 수치적으로 나타남

◆ 손실 함수의 종류

- 평균 제곱 오차(Mean Square Error)

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

- 교차 엔트로피 오차(Cross Entropy Error)

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

평균 제곱 오차(Mean Square Error)

- 숫자 이미지가 2 인 두 개의 이미지의 Softmax 값을 구한 두가지 경우

'2'일 확률이 가장 높다고 추정한 경우

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]

```
def sum_squares_error(y, t):
return 0.5*np.sum(y-t)**2
```

0.0975.....

정답 레이블과 오차가 작음

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

'7'일 확률이 가장 높다고 추정한 경우

y= [0.1, 0.05 ,0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

t= [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

```
def sum_squares_error(y, t):
return 0.5*np.sum(y-t)**2
```

0.5975.....

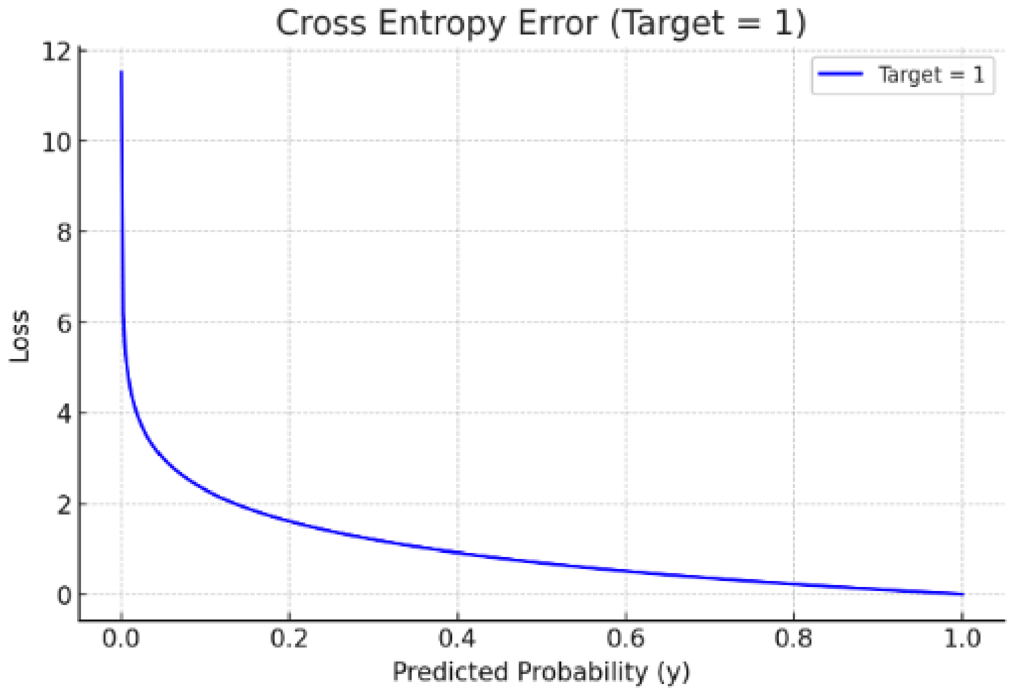
정답 레이블과 오차가 큼

교차 엔트로피 오차(Cross Entropy Error)

- 교차 엔트로피는 정보이론에서 확률분포사이의 거리를 재는 방법
- 데이터가 신경망을 거쳐 나온 확률 벡터와 라벨로 구한 교차 엔트로피

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

- y_k : 신경망이 k라고 예측한 확률
- t_k : 라벨을 원 핫 인코딩한 후 k번째 좌표
- 라벨이 k_0라고 하면 $E = -\log y_{k0}$



- 정답에 해당하는 출력이 커질수록 0에 다가가다가 그 출력이 1일때 0이 된다.
- 반대로 정답일 때의 출력이 작아질수록 오차는 커진다.

왜 손실 함수를 설정하는가?

- ◆ 신경망 학습에서는 최적의 매개변수(가중치와 편향)을 탐색할때 손실함수의 값을 가능한 한 작게하는 매개변수 값을 찾음.
- ◆ 이때 매개변수의 미분(정확히는 기울기)을 계산하고, 그 미분 값을 단서로 매개변수의 값을 서서히 갱신하는 과정을 반복함.
- ◆ Eg. 가상의 신경망이 있고, 그 신경망의 어느 한 가중치 매개변수가 있을때
  - 가중치 매개변수의 손실함수 미분 → 가중치 매개변수의 값을 아주 조금 변화 시켰을때 손실함수가 어떻게 변화하는가
  - 미분값이 음수 → 가중치 매개변수를 양의 방향으로 변화시켜 손실함수 값을 줄임.
  - 미분값이 양수 → 가중치 매개변수를 음의 방향으로 변화시켜 손실함수 값을 줄임.
  - 미분값이 0 → 가중치 매개변소를 어느쪽으로 움직여도 손실함수 값이 줄어들지 않음.

    * 미분값이 0이 될때 가중치 매개변수의 갱신이 멈춤. 학습 종료.

목 차

수치 미분

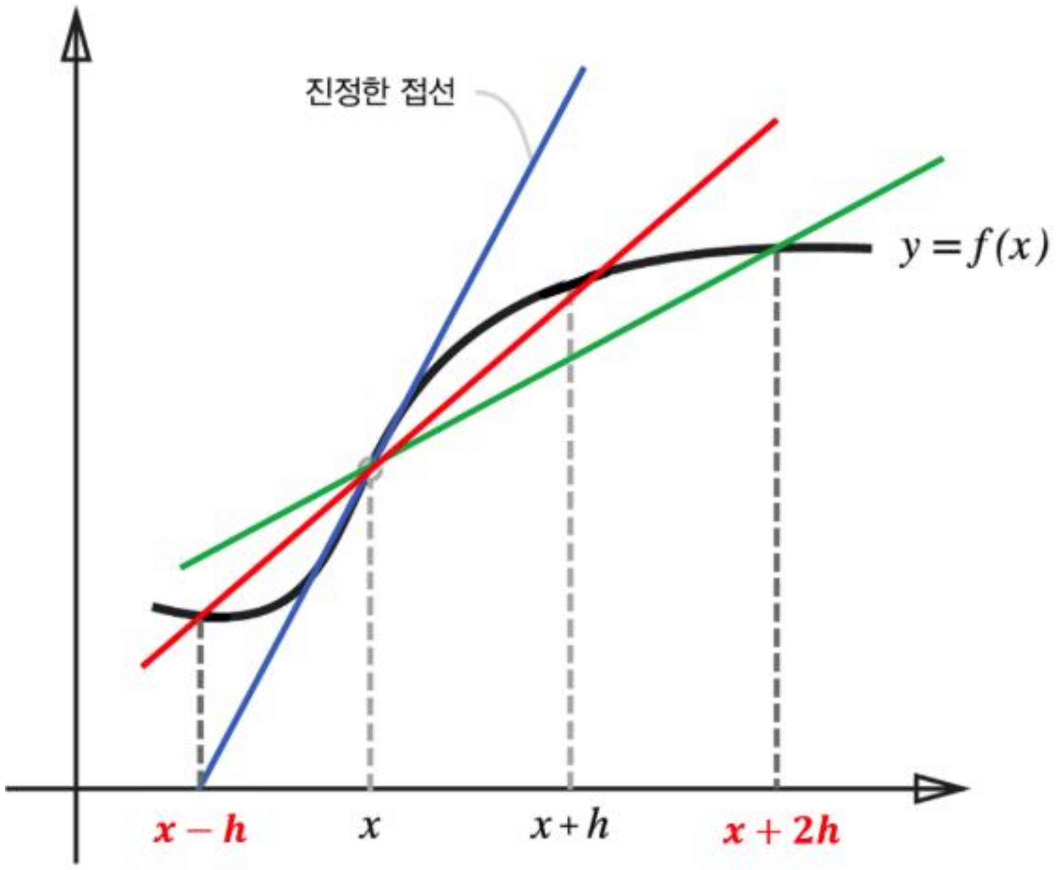
- 함수의 미붓값을 해석적(수학적 공식)으로 구하지 않고 컴퓨터를 이용해 수치적으로 근사하는 방법

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

$$\Downarrow$$

$$\frac{f(x+h) - f(x-h)}{2h}$$

- h의 값을 10^{-4} 으로 설정하여 해석적 방법과 최대한 근사한 미분값을 구함.



<실습 과제>

$y = 0.01x^2 + 0.1x$ 의 x좌표 5에서

수치미분 접선을 그리시오

참고 > ch03/gradient_1d.ipynb 에 작성함.

실습 과제

실습: ch03/gradient_1d.ipynb

```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def numerical_diff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)
```

수치미분함수

```python
def function_1(x):
    return 0.01*x**2 + 0.1*x
```

미분할 함수

```python
def tangent_line(f, x):
    d = numerical_diff(f, x)
    print(d)
    y = f(x) - d*x
    return lambda t: d*t + y
```

접선을 구하는 함수

```python
x = np.arange(0.0, 20.0, 0.1)
y = function_1(x)
plt.xlabel("x")
plt.ylabel("f(x)")

tf = tangent_line(function_1, 5)
y2 = tf(x)

plt.plot(x, y)
plt.plot(x, y2)
plt.show()
```

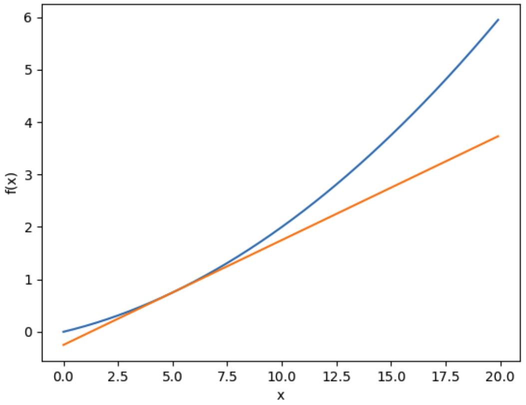
목 차

일변수 미분 vs 다변수 미분(편미분)

- 한 개의 변수에 대해 미분하는 것을 일변수 미분이라고 한다.

$$f(x) = x^2$$

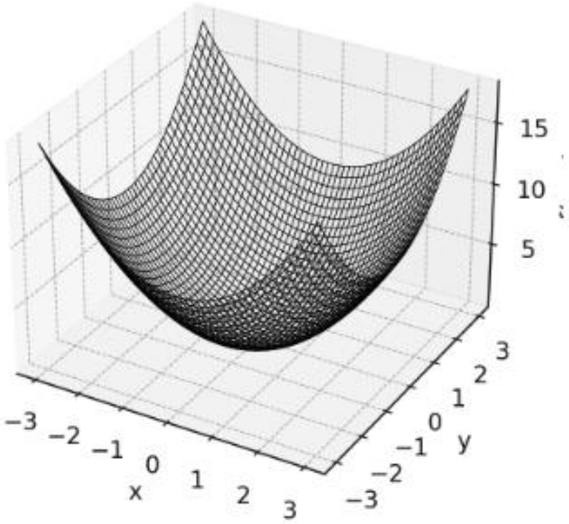
$$\frac{f(x+h) - f(x-h)}{2h}$$



- 아래 함수와 같이 변수가 2개(점)일때 각 축에 대해 미분하는 것을 편미분이라고 한다.

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

$$\frac{f(\mathbf{x} + h\vec{v}) - f(\mathbf{x})}{h}$$



<실습 과제>

$f(x_0, x_1) = x_0^2 + x_1^2$의 각 좌표의 접선의 기울기를 2차원 평면에 표현하시오.

참고 > ch03/gradient_2d.ipynb 에 작성함.

실습 과제

실습: ch03/gradient_2d.ipynb

```
# coding: utf-8
# cf.http://d.hatena.ne.jp/white_wheels/20100327/p3
import numpy as np
import matplotlib.pylab as plt
from mpl_toolkits.mplot3d import Axes3D


def _numerical_gradient_no_batch(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) # x와 형상이 같은 배열을 생성

    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = float(tmp_val) + h
        fxh1 = f(x)

        # f(x-h) 계산
        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val # 값 복원

    return grad
```

2차원을 고려한
수치미분 원소 함수

실습 과제

실습: ch03/gradient_2d.ipynb

```python
def numerical_gradient(f, X):
    if X.ndim == 1:
        return _numerical_gradient_no_batch(f, X)
    else:
        grad = np.zeros_like(X)

        for idx, x in enumerate(X):
            grad[idx] = _numerical_gradient_no_batch(f, x)

        return grad


def function_2(x):
    if x.ndim == 1:
        return np.sum(x**2)
    else:
        return np.sum(x**2, axis=1)


def tangent_line(f, x):
    d = numerical_gradient(f, x)
    print(d)
    y = f(x) - d*x
    return lambda t: d*t + y
```

변수가 하나일때 수치미분

변수가 두개 이상일때 수치 미분

변수가 하나일때

변수가 두개일때

접선을 구하는 함수

실습 과제

실습: ch03/gradient_2d.ipynb

```python

if __name__ == '__main__':
    x0 = np.arange(-2, 2.5, 0.25)
    x1 = np.arange(-2, 2.5, 0.25)
    X, Y = np.meshgrid(x0, x1)

    X = X.flatten()
    Y = Y.flatten()

    print(X)
    print(Y)

    grad = numerical_gradient(function_2, np.array([X, Y]) )
    print(grad)

    plt.figure()
    plt.quiver(X, Y, -grad[0], -grad[1],  angles="xy",color="#666666")#,headwidth=10,scale=40,color="#444444")
    plt.xlim([-2, 2])
    plt.ylim([-2, 2])
    plt.xlabel('x0')
    plt.ylabel('x1')
    plt.grid()
    plt.legend()
    plt.draw()
    plt.show()

```

2차원의 빈 평면을
만드는 함수

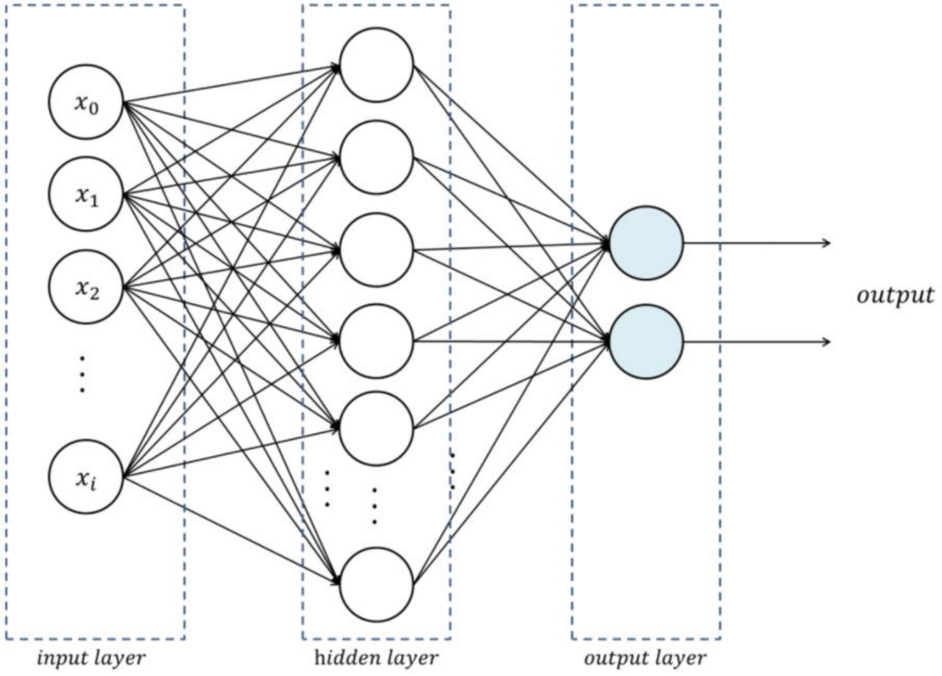
2차원 평면을 1차원
으로 평탄화

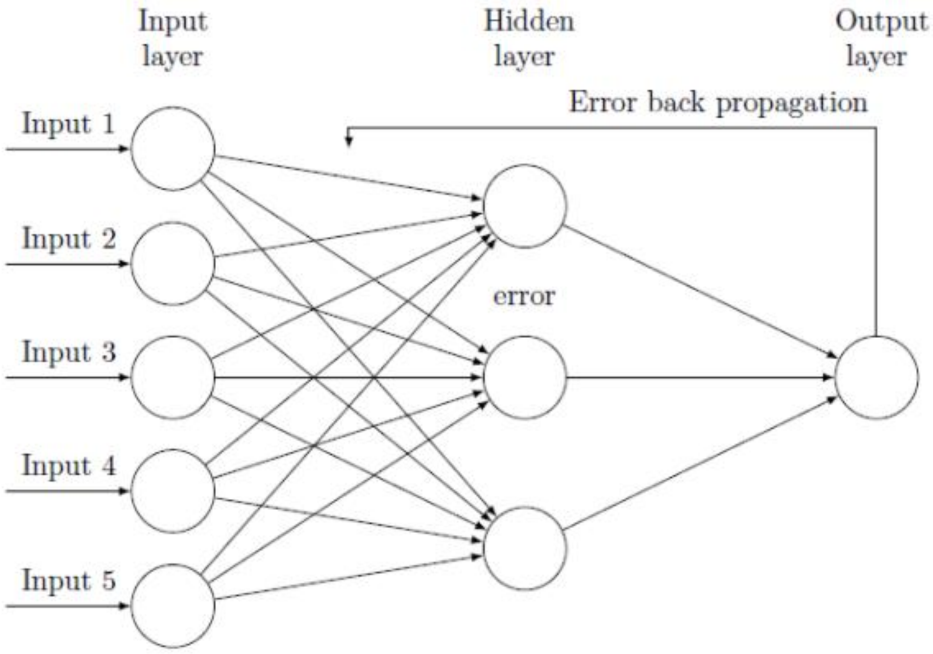
각 좌표에 대한 기울기 계산

각 좌표에 대한 기울기를 화살
표로 표현하는 부분

DNN의 학습 : 순전파와 역전파

- Forward Propagation : Input → Output 방향으로 출력값을 계산하는 과정. 실측값과 차이인 오차(loss) 계산.
- Back Propagation : Output → Input 방향으로 가중치를 갱신하는 과정. 딥 러닝에서 학습이 이루어짐

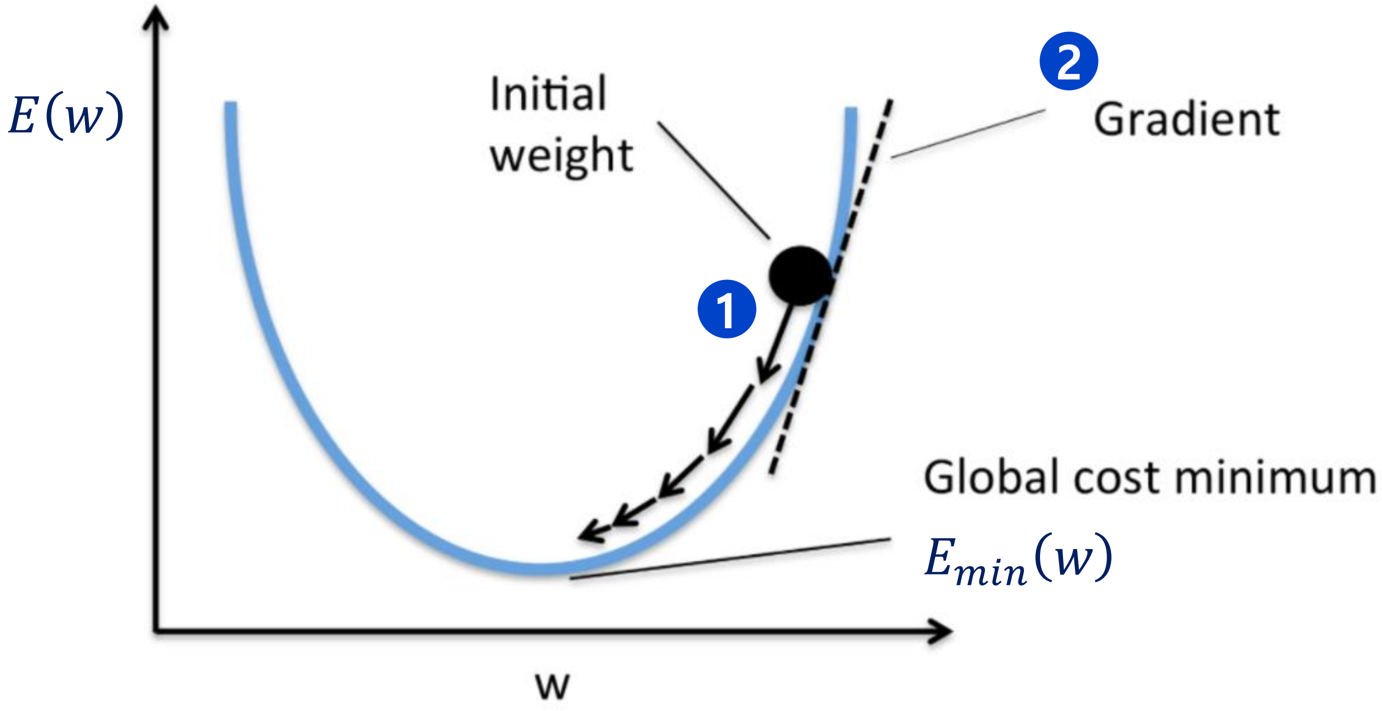


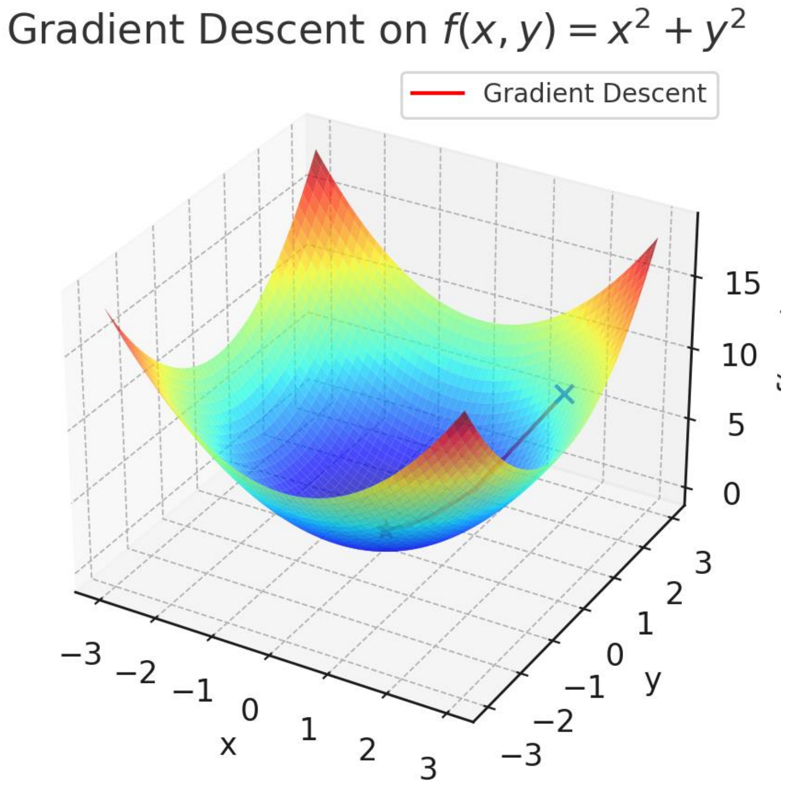


경사하강법(Gradient Descent)(1)

- 역전파는 출력값과 실제값의 차이인 오차가 최소가 되도록 가중치를 갱신함.
- 이때 가중치 갱신 방법으로 경사 하강법을 사용함.

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla f(\mathbf{x}_n)$$





경사하강법(Gradient Descent)(2)

- 오차가 낮아지는 방향으로 이동할 목적으로 미분
- 가중치 업데이트 : 미분값을 가중치에서 빼 줌으로써 w 값을 줄임

$$w^{+} = w - \underbrace{\eta}_{1} * \underbrace{\frac{\partial E}{\partial w}}_{2}$$

1 **Learning Rate : 학습률**
(학습 보폭은?)

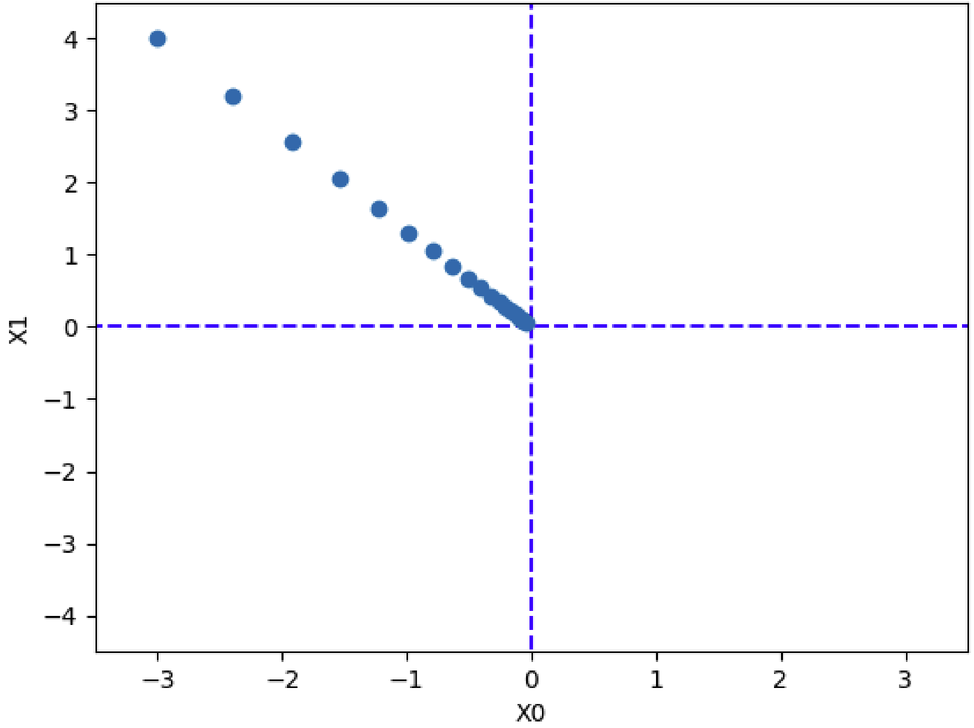
2 **Gradient : 기울기**
(학습 방향은?)

실습 과제

<실습 과제>

$f(x_0, x_1) = x_0^2 + x_1^2$의 Gradient descent 과정을 2차원 평면에 표현하시오.

참고 > ch03/gradient_method.ipynb 에 작성함.



실습 과제

실습: ch03/gradient_method.ipynb

```python
# coding: utf-8
import os, sys
print(os.getcwd())

import numpy as np
import matplotlib.pylab as plt

#from gradient_2d import numerical_gradient
def _numerical_gradient_no_batch(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) # x와 형상이 같은 배열을 생성

    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = float(tmp_val) + h
        fxh1 = f(x)

        # f(x-h) 계산
        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val # 값 복원

    return grad


def numerical_gradient(f, X):
    if X.ndim == 1:
        return _numerical_gradient_no_batch(f, X)
    else:
        grad = np.zeros_like(X)

        for idx, x in enumerate(X):
            grad[idx] = _numerical_gradient_no_batch(f, x)

        return grad
```

수치미분을 계산하는 원소 함수

변수가 1개일때의 수치미분 수행

변수가 2개이상일때의 수치미분 수행

실습 과제

실습: ch03/gradient_method.ipynb

```python
def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x = init_x
    x_history = []

    for i in range(step_num):
        x_history.append( x.copy() )

        grad = numerical_gradient(f, x)
        x -= lr * grad

    return x, np.array(x_history)


def function_2(x):
    return x[0]**2 + x[1]**2

init_x = np.array([-3.0, 4.0])

lr = 0.1
step_num = 20
x, x_history = gradient_descent(function_2, init_x, lr=lr,\
                                step_num=step_num)

plt.plot( [-5, 5], [0,0], '--b')
plt.plot( [0,0], [-5, 5], '--b')
plt.plot(x_history[:,0], x_history[:,1], 'o')

plt.xlim(-3.5, 3.5)
plt.ylim(-4.5, 4.5)
plt.xlabel("X0")
plt.ylabel("X1")
plt.show()
```

수치미분으로 구한 기울기 를 이용해 중심으로 이동

미분전 원래 함수

기울기값을 이용해 중심으 로 이동시키는 함수 호출

신경망에서 기울기

- 신경망의 기존 가중치에 기울기는 반영하여 새롭게 조정된 가중치를 구하는 과정이 학습 과정이다.

기존 가중치

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

새롭게 조정된 가중치

$$w^{+} = w - \eta * \frac{\partial E}{\partial w}$$

손실함수 기울기값

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$$

<실습 과제>

정규분포로 초기화된 가중치(2x3)에

Cross entropy error 손실함수를 써서

새로운 가중치를 산출해 보시오

참고 > ch03/gradient_simplenet.ipynb 에 작성함.

실습 과제

실습: ch03/gradient_simplenet.ipynb

```python
# coding: utf-8
import sys, os
print(os.getcwd())
current_dir = os.path.dirname(os.getcwd())
print(current_dir)
os.chdir(current_dir)

import numpy as np
from common.functions import softmax, cross_entropy_error
from common.gradient import numerical_gradient


class simpleNet:
    def __init__(self):
        self.W = np.random.randn(2,3) # 정규분포로 초기화

    def predict(self, x):
        return np.dot(x, self.W)

    def loss(self, x, t):
        z = self.predict(x)
        y = softmax(z)
        loss = cross_entropy_error(y, t)

        return loss

x = np.array([0.6, 0.9])
t = np.array([0, 0, 1])

net = simpleNet()

f = lambda w: net.loss(x, t)
dW = numerical_gradient(f, net.W)

print(dW)
```

Cross entropy 함수 import

신경망 출력값 계산

손실값 계산

손실값을 미분하여 가중치의 기울기 계산

목 차

학습 알고리즘 구현하기

- 학습 알고리즘의 4 단계

전제 : 신경망을 학습 시킨다는 것은 가중치와 편향을 조정하는 과정을 의미한다.

Steep1 - 미니 배치 :

훈편 데이터 중 일부를 랜덤 선발하여 하나의 세트로 만들며 이를 미니 배치라고 함.

Steep2 – 기울기 산출 :

미니 배치의 손실 함수 값을 줄이기 위해 그 값을 가장 작게하는 방향(기울기)을 구한다.

Steep3 – 매개변수 갱신 :

Step2에서 구한 기울기 방향으로 가중치 매개변수를 아주 조금 갱신함.

반복

딥러닝의 학습 단위 : Batch와 Epoch

◆ 배치(Batch)

- 데이터 세트 내의 데이터들을 한 개씩 학습 → 컴퓨터 I/O의 제약, 불안정한 학습

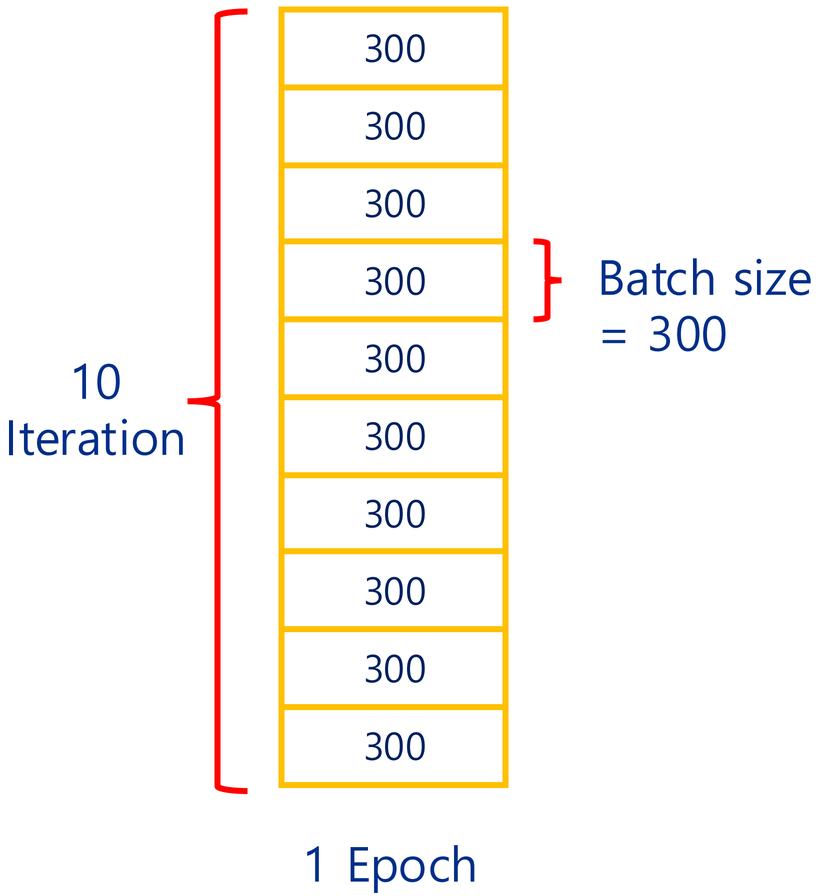
➔ 데이터 세트를 작은 덩어리(mini batch)로 나누어 처리하는 것이 효율적임

◆ 에폭(Epoch)

- 전체 데이터 세트의 순전파/역전파를 한번 완료한 횟수를 의미
  여러 번 수행해야 경사하강법의 효과를 볼 수 있음
  - 에폭이 크면 : overfitting 문제
  - 에폭이 작으면 : underfitting 문제

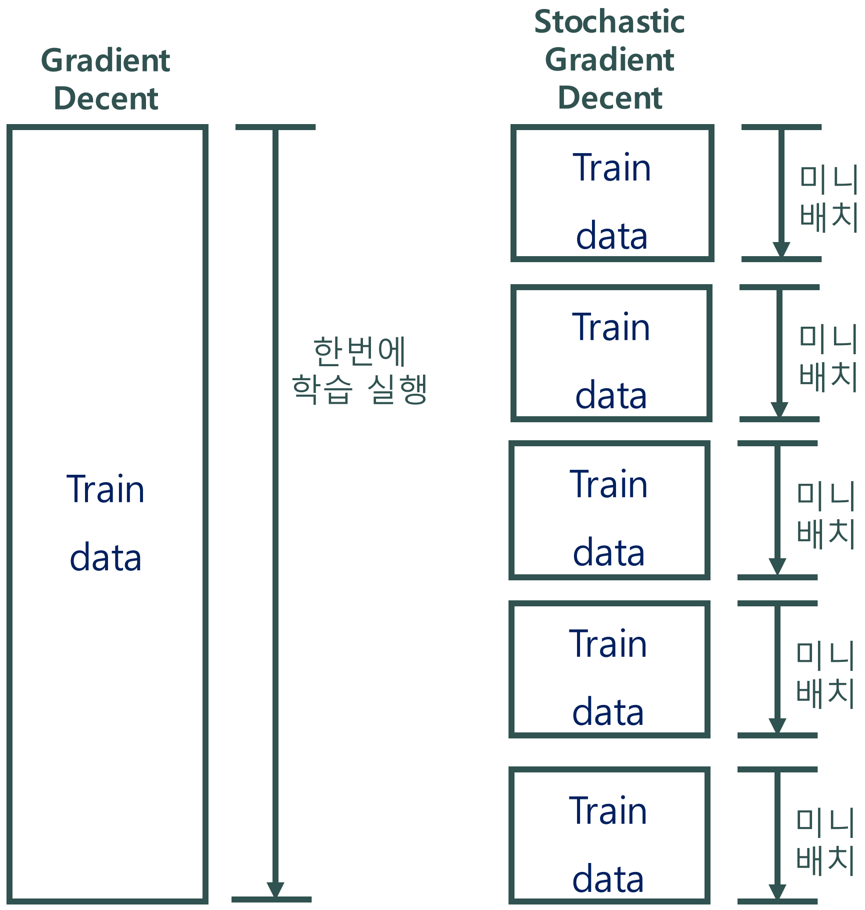
◆ 반복(Iteration)

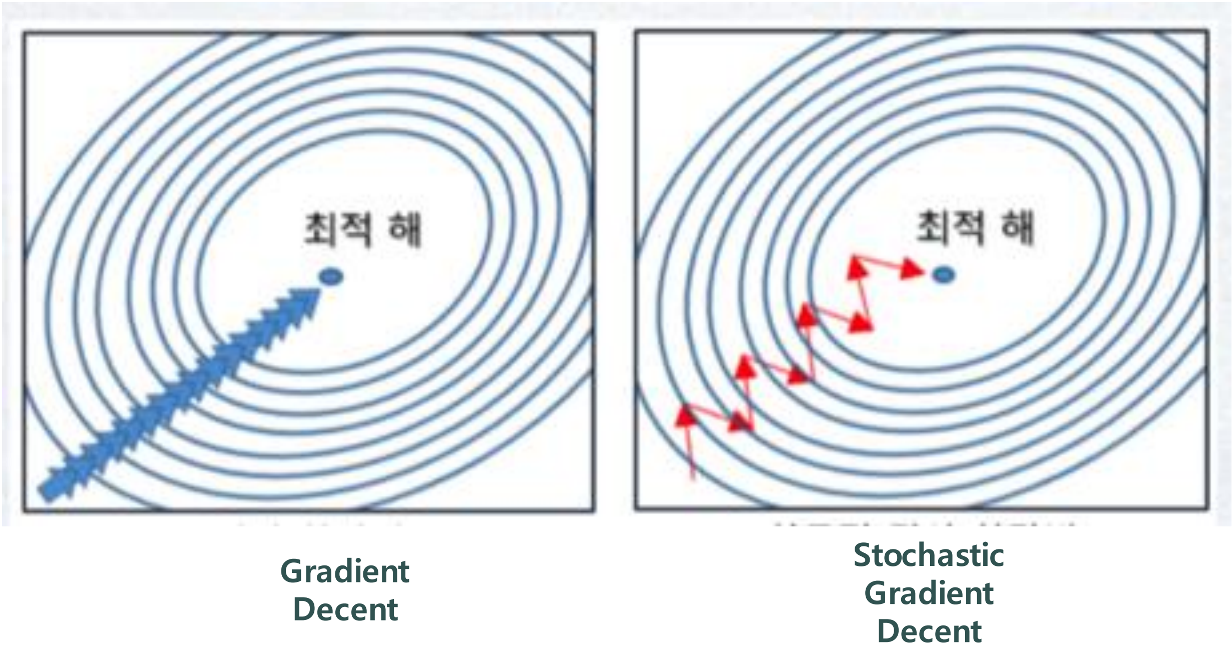
- 에폭을 나누어 실행하는 횟수 = Mini batch의 갯수



미니배치 학습 : Stochastic Gradient Descent

- 훈련 데이터 중 무작위로 선별한 데이터를 미니 배치라고 함.
- 효율적인 계산, 수렴 속도 개선, 학습속도 개선





<실습 과제>

TwoLayerNet 클래스를 작성해보자

참고 > ch03/two_layer_net.ipynb

twol_ayer_net.py 소스 코드(1)

실습: ch03/two_layer_net.ipynb

```python
# coding: utf-8
import sys, os
current_dir = os.path.dirname(__file__)
parent_dir = os.path.dirname(current_dir)
sys.path.append(parent_dir)

from common.functions import *
from common.gradient import numerical_gradient


class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    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
```

신경망의 가중치를 초기화함

신경망의 출력값을 계산하는 함수

twol_ayer_net.py 소스 코드(2)

실습: ch03/two_layer_net.py

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

        return cross_entropy_error(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

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

        return grads
```

신경망 출력값의 손실 값을 계산하는 함수

예측의 정확도를 계산하는 함수

가중치에 대한 수치 미분을 수행하는 함수

two_layer_net.py 소스 코드(3)

실습: ch03/two_layer_net.py

```python

    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
```

가중치에 대한 역전파 방식
미분을 수행하는 함수

<실습 과제>

MNIST 숫자 데이터를 인식하고 검증하는 코드를 구현해 보자

참고 > ch03/train_neuralnet.ipynb 에 작성함.

train_neuralnet.ipynb 소스 코드(1)

실습: ch03/train_neuralnet.ipynb

```python
import os, sys
print(os.getcwd())
current_dir = os.path.dirname(os.getcwd())
print(current_dir)
os.chdir(current_dir)

import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from ch03.two_layer_net import TwoLayerNet

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0]
batch_size = 100   # 미니배치 크기
learning_rate = 0.1

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

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)

```

숫자 이미지 데이터 다운로드 from Internet

Input(784), Hidden(50), Output(10) 구성의 신경망 생성

train_neuralnet.ipynb 소스 코드(2)

실습: ch03/train_neuralnet.ipynb

```python

for i in range(iters_num):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

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

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

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

    # 1에폭당 정확도 계산
    if i % iter_per_epoch == 0:
        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(i, "train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

```

train_neuralnet.ipynb 소스 코드(3)

```
# 그래프 그리기
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()
```

실습: ch03/train_neuralnet.ipynb

학습 과정 및 결과

```
/mnt/batch/tasks/shared/LS_root/mounts/clusters/el203/code/Users/el20/DL/ch03
/mnt/batch/tasks/shared/LS_root/mounts/clusters/el203/code/Users/el20/DL
0 train acc, test acc | 0.09915, 0.1009
600 train acc, test acc | 0.7858, 0.7917
1200 train acc, test acc | 0.8772, 0.8791
1800 train acc, test acc | 0.8961333333333333, 0.8973
2400 train acc, test acc | 0.9063, 0.9082
3000 train acc, test acc | 0.9125666666666666, 0.9122
3600 train acc, test acc | 0.9177, 0.9197
4200 train acc, test acc | 0.9223166666666667, 0.9223
4800 train acc, test acc | 0.9258666666666666, 0.9244
5400 train acc, test acc | 0.9294666666666667, 0.9296
6000 train acc, test acc | 0.9317666666666666, 0.9306
6600 train acc, test acc | 0.93485, 0.9336
7200 train acc, test acc | 0.9363166666666667, 0.9373
7800 train acc, test acc | 0.9388333333333333, 0.9392
8400 train acc, test acc | 0.9409166666666666, 0.9386
9000 train acc, test acc | 0.9436833333333333, 0.9418
9600 train acc, test acc | 0.9450833333333334, 0.9422
```