분석 멘토링 C조_3주차

4장. 신경망 학습

발표자: 김영은

INDEX

1. 신경망 학습

1. 신경망 학습

: 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 것
딥러닝 – 종단간 기계학습 / end-to-end로 학습

그림 4-2 규칙을 '사람'이 만드는 방식에서 '기계'가 데이터로부터 배우는 방식으로의 패러다임 전환 : 회색 블록은 사람이 개입하지 않음을 뜻한다.

-> 입력부터 출력까지 사람의 개입 없이 기계가 학습한다.

1. 신경망 학습

Q. 훈련/시험 데이터로 나눠서 학습과 실험을 수행하는 이유?
A. 범용 능력을 제대로 평가하기 위함 !

훈련 데이터에 포함되지 않은 데이터로도 문제를 올바르게 풀어내는 능력
https://lsjsj92.tistory.com/545

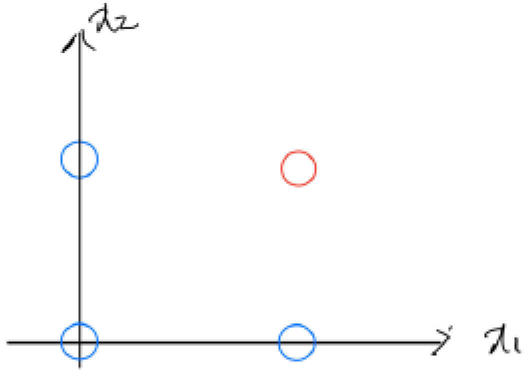
범용 능력 획득이 기계학습의 최종 목표 !!
데이터셋 하나로만 매개변수의 학습과 평가를 수행한다면 지나치게 최적화된 오버피팅이 발생할 수 있다 ~ !

1. 신경망 학습

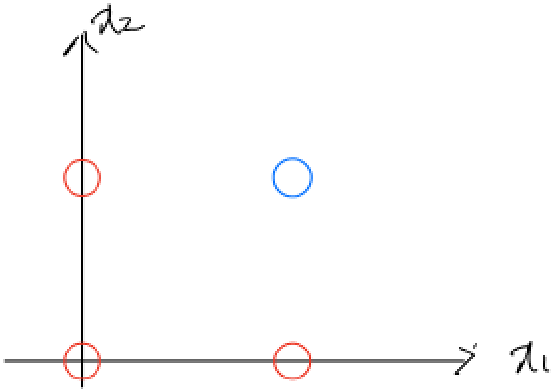
Cf. 퍼셉트론

1. 퍼셉트론에서 적절한 매개변수(가중치w/임계값θ,-b)을 찾으면,
같은 구조의 퍼셉트론(AND,NAND,OR 게이트) 모두 표현 가능했다.
퍼셉트론의 목표 : 적절한 매개변수(w/θ) 찾아, x값을 잘~ 분류하기.

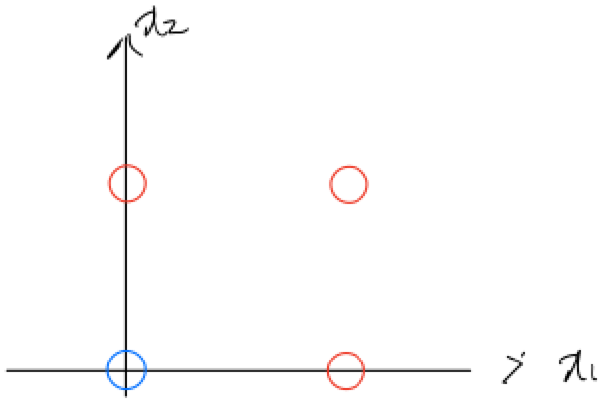
AND



NAND



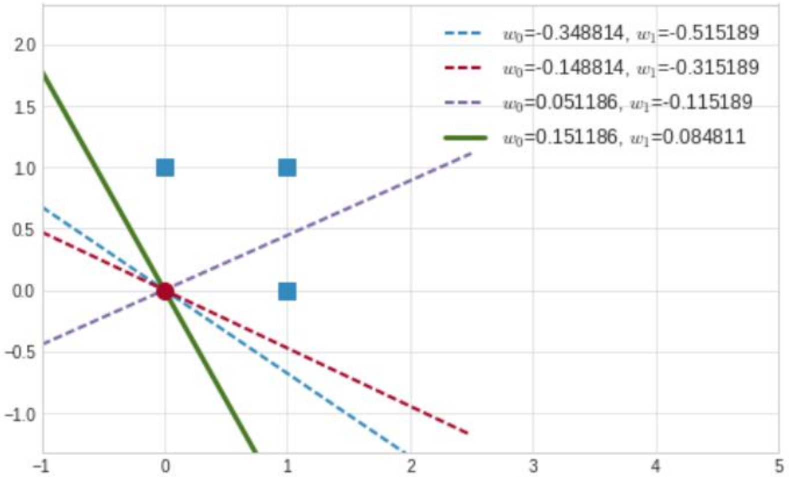
OR

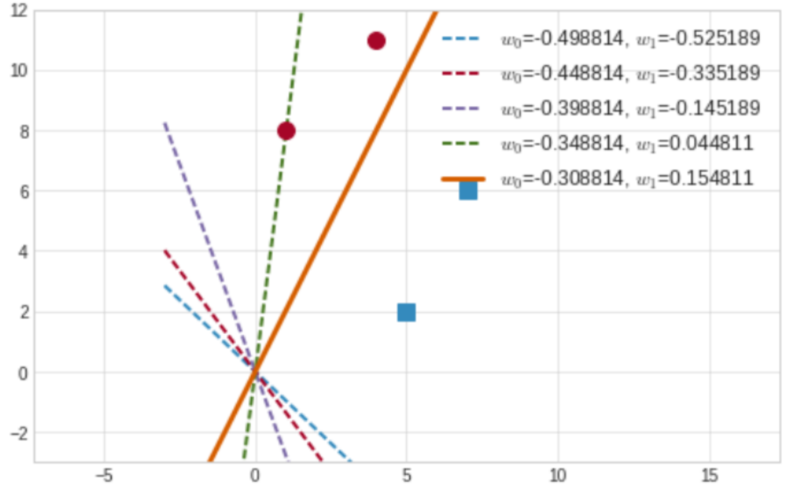


Cf. 퍼셉트론

2. 퍼셉트론에서는 매개변수를 수작업으로 설정했으나, 실제 신경망에서는 수만개의 매개변수가 있기 때문에 신경망 학습이 필수적이다.

3. 퍼셉트론도 직선으로 분리할 수 있는 문제라면 데이터로부터 자동으로 학습이 가능했으나, 비선형 분리 문제는 자동 학습이 불가능하다. -> 퍼셉트론 수렴 정리
https://nbviewer.jupyter.org/github/metamath1/ml-simple-works/blob/master/perceptron/perceptron.ipynb





2. 손실 함수

2. 손실 함수

신경망 학습은 최적의 매개변수 값을 탐색한다.

⬇

기준 지표 필요! => “손실 함수(Loss function)”

손실 함수를 사용하여 신경망 성능의 ‘나쁨의 정도’를 표현한다.

MSE & 교차 엔트로피 오차 사용

2. 손실 함수

1) 평균 제곱 오차 MSE

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

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

```
import numpy as np
# 예1: '2'일 확률이 가장 높다고 추정함 (0.6)
mean_squared_error(np.array(y), np.array(t))
```

```
0.09750000000000031
```

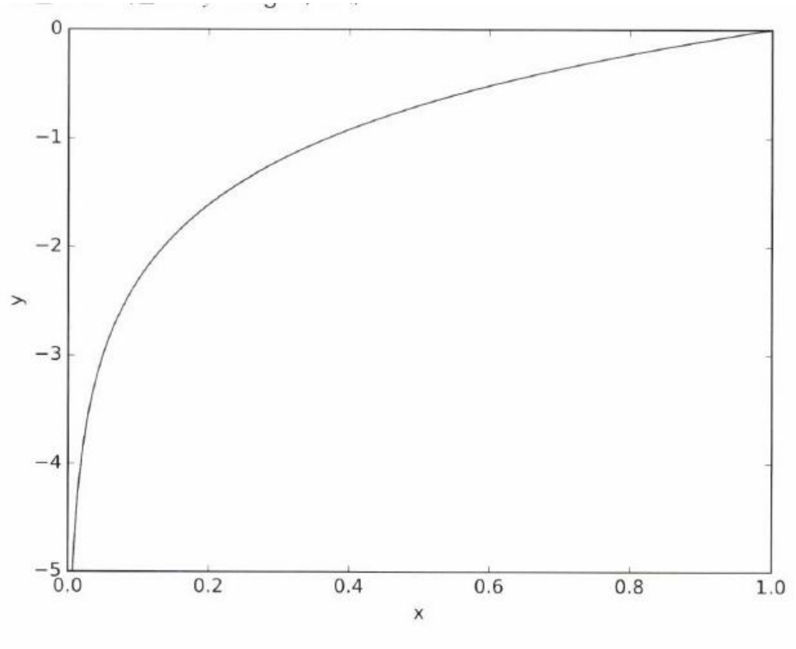
```
# 예2 '7'일 확률이 가장 높다고 추정함 (0.6)
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
mean_squared_error(np.array(y), np.array(t))
```

```
0.5975000000000003
```

2. 손실 함수

2) 교차 엔트로피 오차 CEE

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




```
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y + delta))
```

-> Y가 0이 되면 계산 불가. 0이 되지 않도록 delta 더하여 CEE 계산

```
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
cross_entropy_error(np.array(y), np.array(t))
```

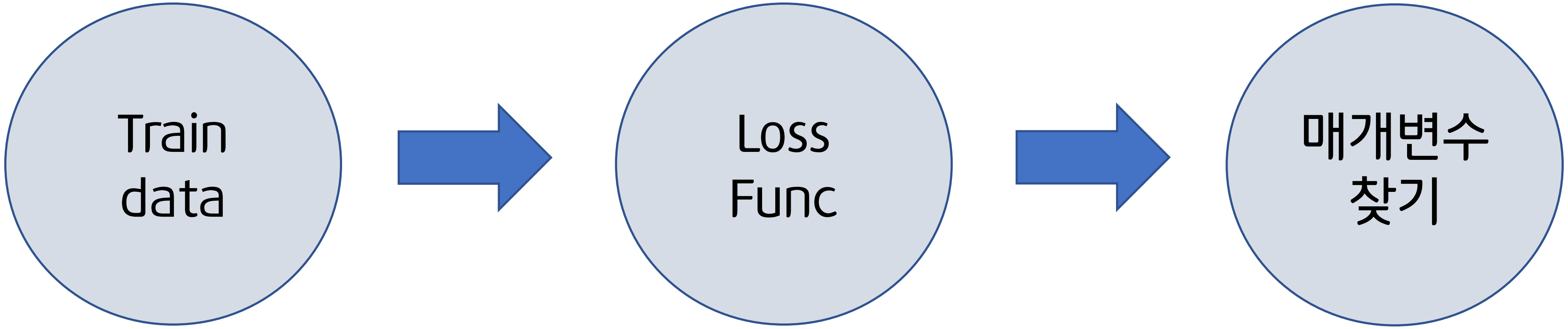
0.51082545709933802

```
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
cross_entropy_error(np.array(y), np.array(t))
```

2.3025840929945458

2. 손실 함수

<신경망 학습의 목표>



<평균 교차 엔트로피 오차>

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} log y_{nk}$$ -> 시간 오래 걸림 (-)

2. 손실 함수

3) 미니 배치 학습

: 훈련 데이터로부터 일부만 골라 학습을 수행

```
train_size = x_train.shape[0]
batch_size = 10
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
t_batch = t_train[batch_mask]
```

np.random.choice(60000, 10) : 60000 미만의 수 중에서 무작위로 10개 뽑기

2. 손실 함수

4) 배치용 교차 엔트로피 오차

```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    batch_size = y.shape[0]
    return -np.sum(t * np.log(y)) / batch_size
```

```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t])) / batch_size
```

5) 왜 '정확도' 대신 손실 함수를 신경망 학습 기준 지표로 설정하는가?

가중치 매개변수 손실 함수의 미분(기울기)를 계산하고 그 미분 값을 단서로 매개변수의 값을 서서히 갱신하는 과정을 반복한다.

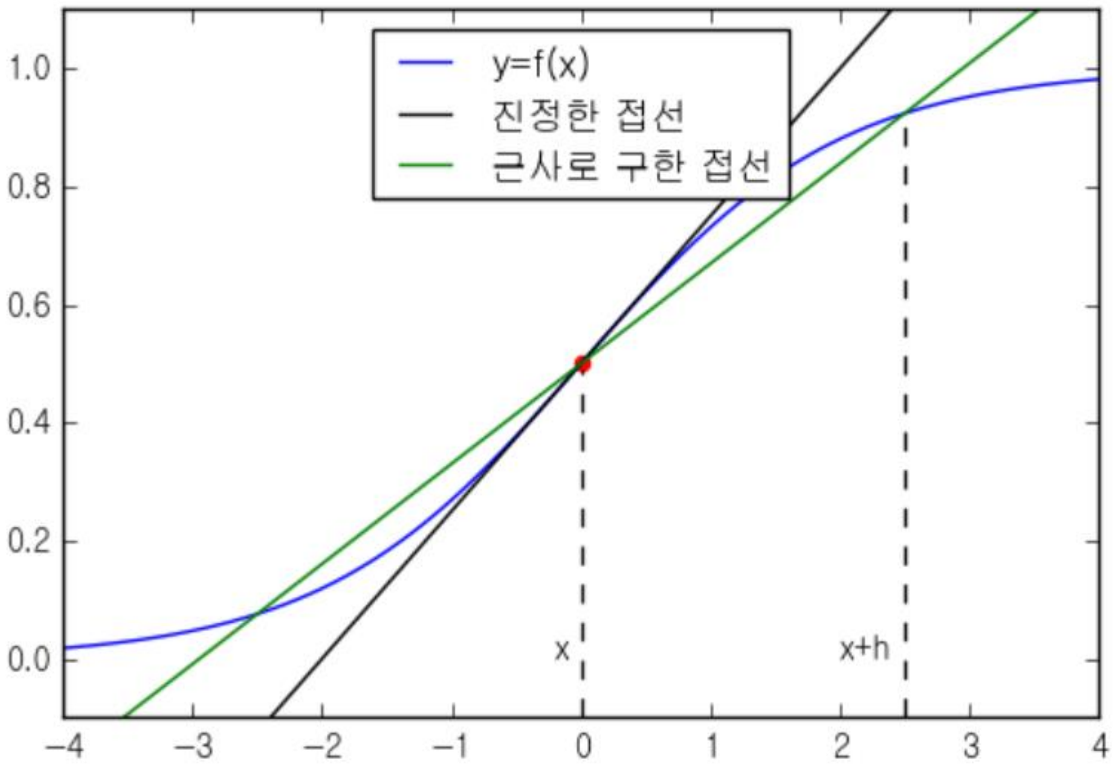
미분값이 0이면 가중치 매개변수를 어느 쪽으로 움직여도 손실 함수의 값은 줄어들지 않는다. 그래서 가중치 매개변수의 갱신이 멈춘다.

정확도를 지표로 하면 매개변수의 미분이 대부분의 장소에서 0이 되기 때문이다.
정확도는 불연속적인 수치 & 미소한 변화에는 반응 X

3. 수치 미분

3. 수치 미분

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



<수치 미분 특징>

1. h를 가능한 작은 값을 대입하기

-> 반올림 오차 문제 발생

-> h로 10^{-4} 정도의 값을 사용

2. X의 기울기 <-> (x+h)와 x 사이의 기울기

-> h를 무한히 0으로 좁히지 못하여 오차 발생

-> x를 중심으로 그 전후의 “차분”을 계산한다.

3. 수치 미분

(수치) 미분	편미분	기울기 gradient
$y = 0.01x^2 + 0.1x$	$f(x_0, x_1) = x_0^2 + x_1^2$	
$\frac{df(x)}{dx}$	$\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}$	$(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1})$

4. 기울기

4. 기울기

https://angeloyeo.github.io/2019/08/25/gradient.html

```
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
```

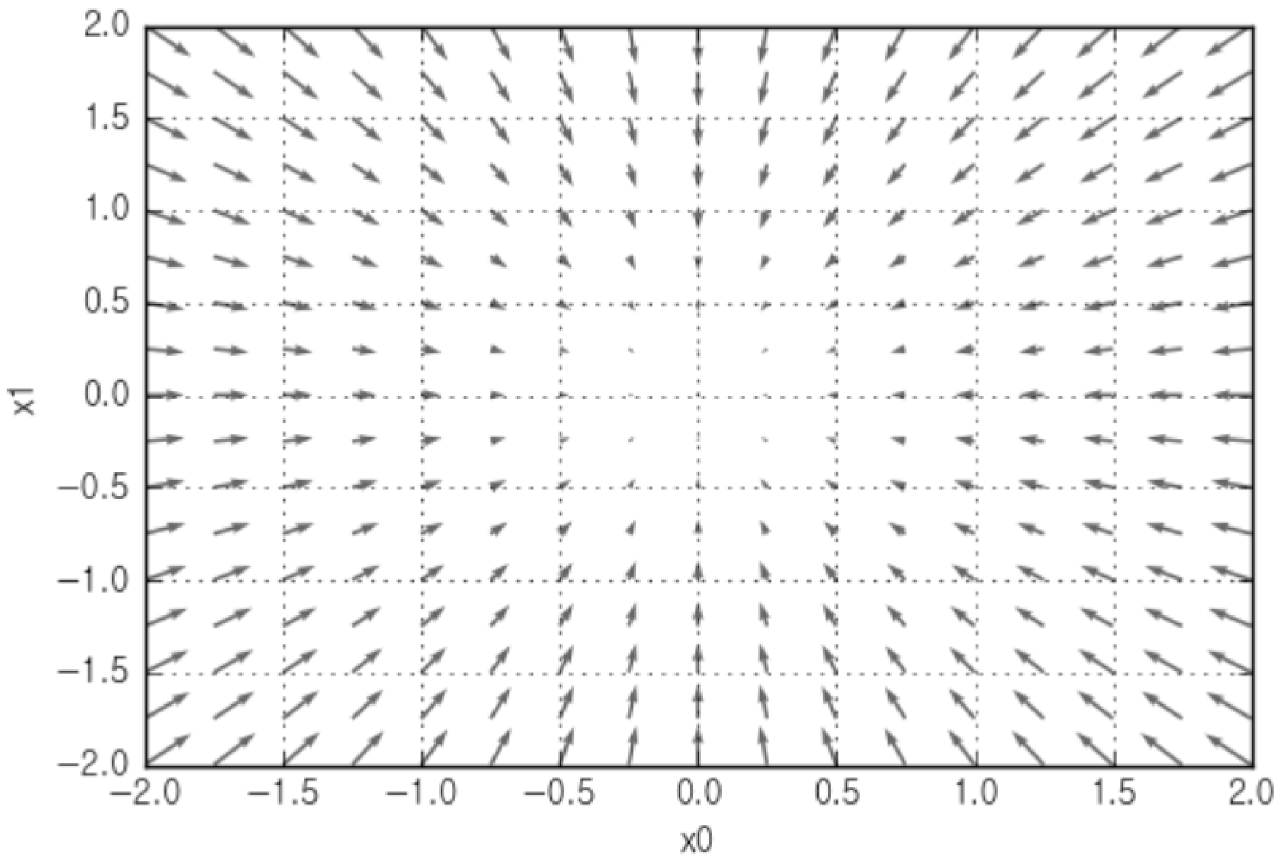
$$f(x_0, x_1) = x_0^2 + x_1^2$$

```
_numerical_gradient_no_batch(function_2, np.array([3.0, 4.0]))
```

```
array([ 6.,  8.])
```

4. 기울기

http://www.gisdeveloper.co.kr/?p=7407



기울기 결과에 마이너스를 붙인 벡터 그리기

기울기는 방향을 가진 벡터(화살표)로 그려짐
'가장 낮은 곳'에서 멀어질 수록 화살표가 커짐을 알 수 있음.
<u>기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 줄이는 방향</u>.

4. 기울기

https://hub1.tistory.com/27

신경망 : 손실 함수가 최솟값이 될 때의 최적의 매개변수를 학습 시 찾기
최솟값을 경사법을 활용하여 찾는다.

경사법 : 현 위치에서 기울어진 방향으로 일정 거리만큼 이동하여 함수의 값을 점차 줄이는 방법.
- 경사 하강법(gradient descent method) : 최소값 찾기
- 경사 상승법(gradient ascent method) : 최대값 찾기

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$
$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

-> 에타(η) = 학습률(learning rate) : 한번에 얼마나 학습해야 할지. 매개변수 값을 얼마나 갱신해야 할지를 정함. Hyper parameter이기 때문에 사람이 직접 설정해야 하는 매개변수. 너무 크면 큰 값으로 발산. 너무 작으면 갱신되지 않은 채 끝남.

4. 기울기

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

    for i in range(step_num):
        grad = numerical_gradient(f, x)
        x -= lr * grad
    return x
```

경사법으로 f(x0,x1) = x0 ** 2 + x1 ** 2의 최소값을 구하라.

```python
init_x = np.array([-3.0, 4.0])
gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)
```

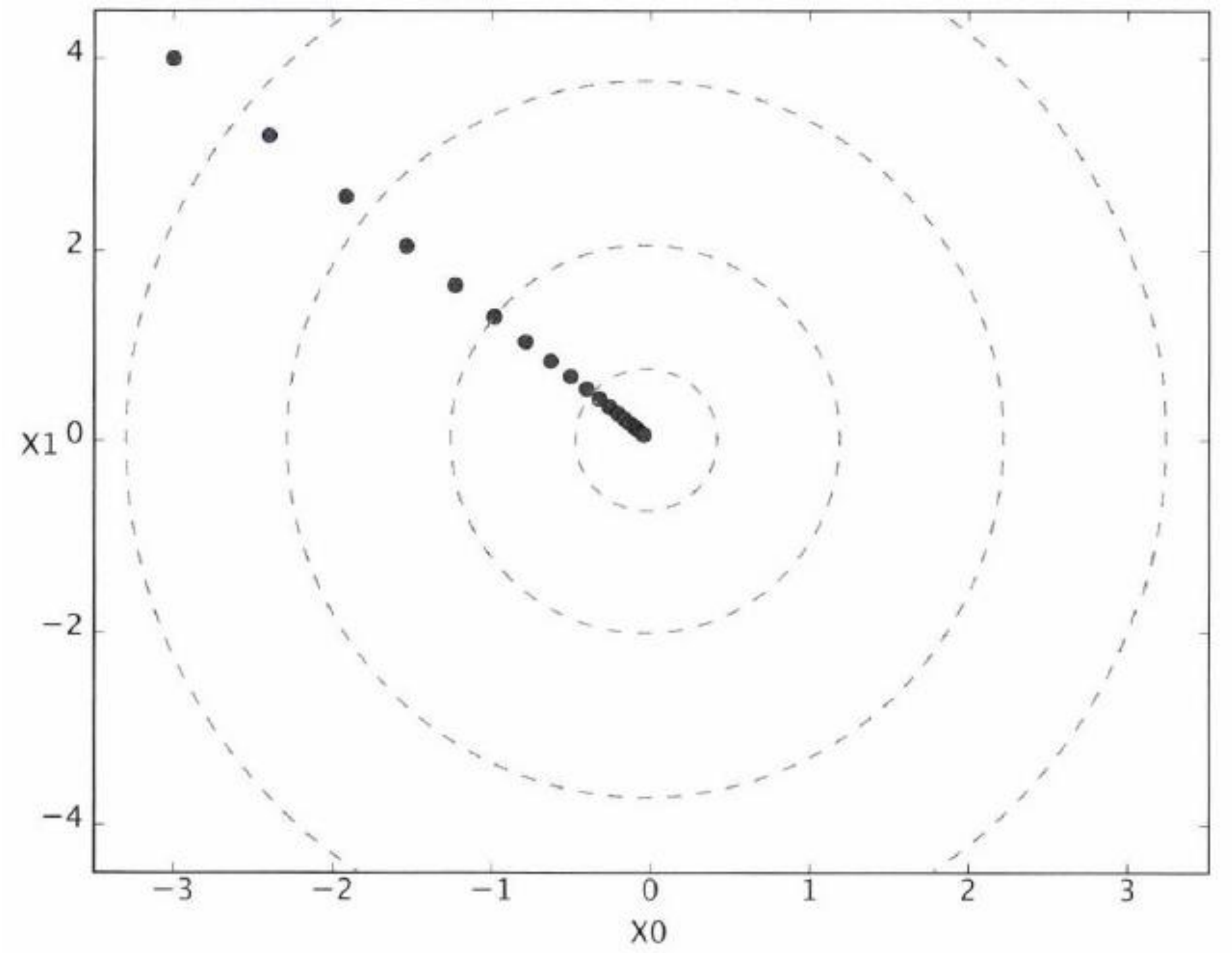
```
array([ -6.11110793e-10,   8.14814391e-10])
```

f : 최적화하려는 함수
Init_x : 초기값
lr(learning rate) : 학습률
step_num : 경사법 반복 회수

numerical_gradient(f,x) : 함수의 기울기
x = (학습률 * 기울기) 갱신을 step_num번 반복함으로써 함수의 극솟값/최솟값을 구한다.

4. 기울기

경사법을 이용한 갱신과정그래프



4. 기울기

신경망에서의 기울기 : 가중치 매개변수 W에 관한 손실 함수의 기울기. 각각 원소에 대한 편미분

$$W = \begin{vmatrix} w_{11} w_{21} w_{31} \\ w_{12} w_{22} w_{32} \end{vmatrix}$$

$$\frac{\partial L}{\partial W} = \begin{vmatrix} \frac{\partial L}{\partial W_{11}} & \frac{\partial L}{\partial W_{21}} & \frac{\partial L}{\partial W_{31}} \\ \frac{\partial L}{\partial W_{12}} & \frac{\partial L}{\partial W_{22}} & \frac{\partial L}{\partial W_{32}} \end{vmatrix}$$

```python
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
```

4. 기울기

```
net = simpleNet()
print(net.W) # 가중치 매개변수
```

```
[[-0.48896906 -0.43767281  0.94069236]
 [ 1.56181584 -0.6269286  -2.09184833]]
```

```
x = np.array([0.6, 0.9])
p = net.predict(x)
print(p)
```

```
[ 1.11225282 -0.82683942 -1.31824808]
```

```
np.argmax(p) # 최대값의 인덱스
```

```
0
```

```
t = np.array([1, 0, 0]) # 정답 레이블
net.loss(x,t)
```

```
0.20849859791009334
```

```
def f(W):
    return net.loss(x, t)
dW = numerical_gradient(f, net.W)
print(dW)
```

```
[[-0.1129187   0.07005907  0.04285962]
 [-0.16937804  0.10508861  0.06428943]]
```

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

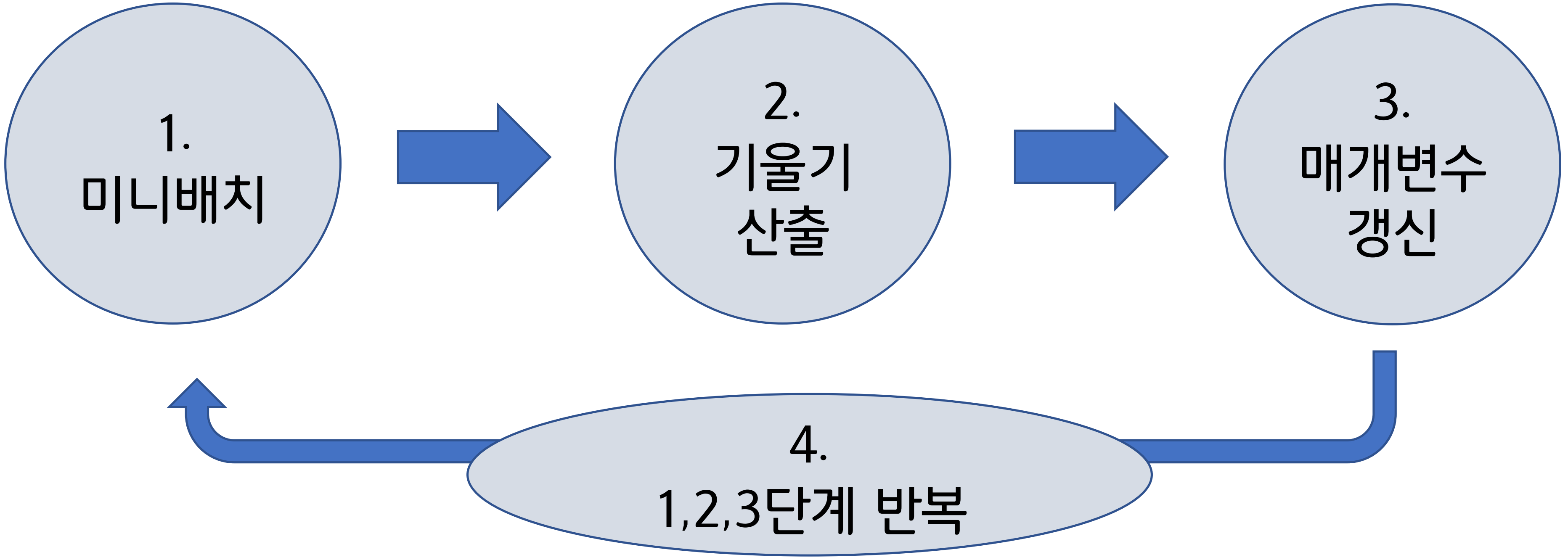
```
[[-0.1129187   0.07005907  0.04285962]
 [-0.16937804  0.10508861  0.06428943]]
```

5. 학습 알고리즘 구현하기

5. 학습 알고리즘 구현하기

신경망에는 적응 가능한 가중치 & 편향이 있고, 이를 데이터에 적응하도록 조정하는 것이 ‘학습’

<신경망 학습 절차>



⇒ 경사하강법으로 매개변수 갱신.

⇒ 데이터를 미니배치로 무작위 선정하기 때문에 확률적 경사하강법(SGD)라고 부른다.

5. 학습 알고리즘 구현하기

<2층 신경망 클래스 구현하기>
TwoLayerNet 클래스는 딕셔너리인 params와 grads를 인스턴스 변수로 갖는다.

- Params 변수

Params['W1'] 은 1번째 층의 가중치 매개변수로 넘파이 배열로 저장된다.
예측 처리(순방향 처리)에서 사용된다.

```
x = np.random.rand(100, 784) # 더미 입력 데이터(100장 분량)
y = net.predict(x)
```

- Grads 변수

Numerical_gradient() 메서드를 사용해 기울기를 계산하면 grabs에 기울기 정보 저장된다.

5. 학습 알고리즘 구현하기

```python
def sigmoid_grad(x):
    return (1.0 - sigmoid(x)) * sigmoid(x)

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
```

5. 학습 알고리즘 구현하기

```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
```

5. 학습 알고리즘 구현하기

<미니배치 학습 구현하기>

```
# 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)
```

5. 학습 알고리즘 구현하기

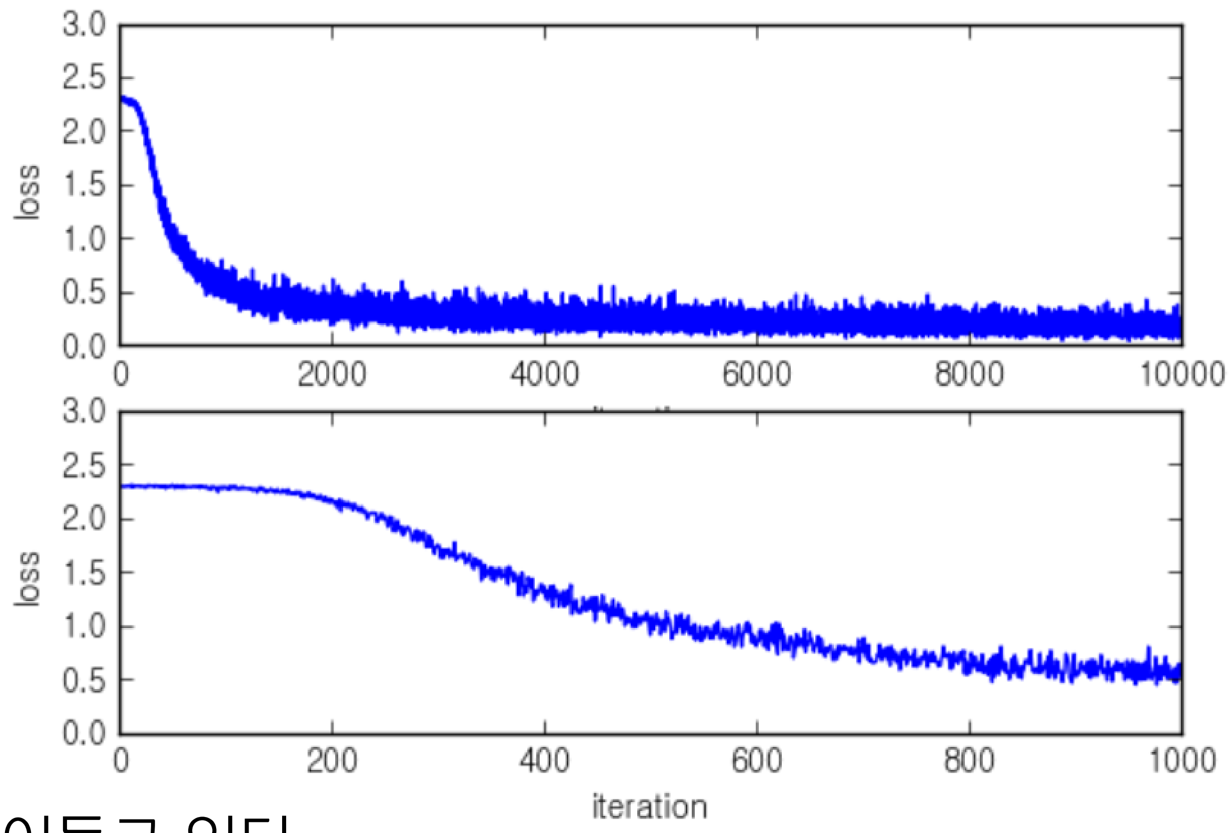
```python
for i in range(iters_num):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

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

```
train acc, test acc | 0.0986166666667, 0.0979
train acc, test acc | 0.77295, 0.782
train acc, test acc | 0.874333333333, 0.8762
train acc, test acc | 0.89825, 0.9008
train acc, test acc | 0.907483333333, 0.9119
train acc, test acc | 0.913916666667, 0.9178
train acc, test acc | 0.919116666667, 0.9221
train acc, test acc | 0.923816666667, 0.9258
train acc, test acc | 0.927466666667, 0.9282
train acc, test acc | 0.930966666667, 0.9303
train acc, test acc | 0.934066666667, 0.9325
train acc, test acc | 0.936083333333, 0.9345
train acc, test acc | 0.939583333333, 0.9377
train acc, test acc | 0.941883333333, 0.9389
train acc, test acc | 0.943916666667, 0.9416
train acc, test acc | 0.945216666667, 0.9427
train acc, test acc | 0.947, 0.9443
```

5. 학습 알고리즘 구현하기

```
f, (ax1, ax2) = plt.subplots(2, 1)
x = np.array(range(iters_num))
ax1.plot(x, train_loss_list, label='loss')
ax1.set_xlabel("iteration")
ax1.set_ylabel("loss")
ax1.set_ylim(0, 3.0)
ax2.plot(x[:1000], train_loss_list[:1000], label='loss')
ax2.set_xlabel("iteration")
ax2.set_ylabel("loss")
ax2.set_ylim(0, 3.0)
```

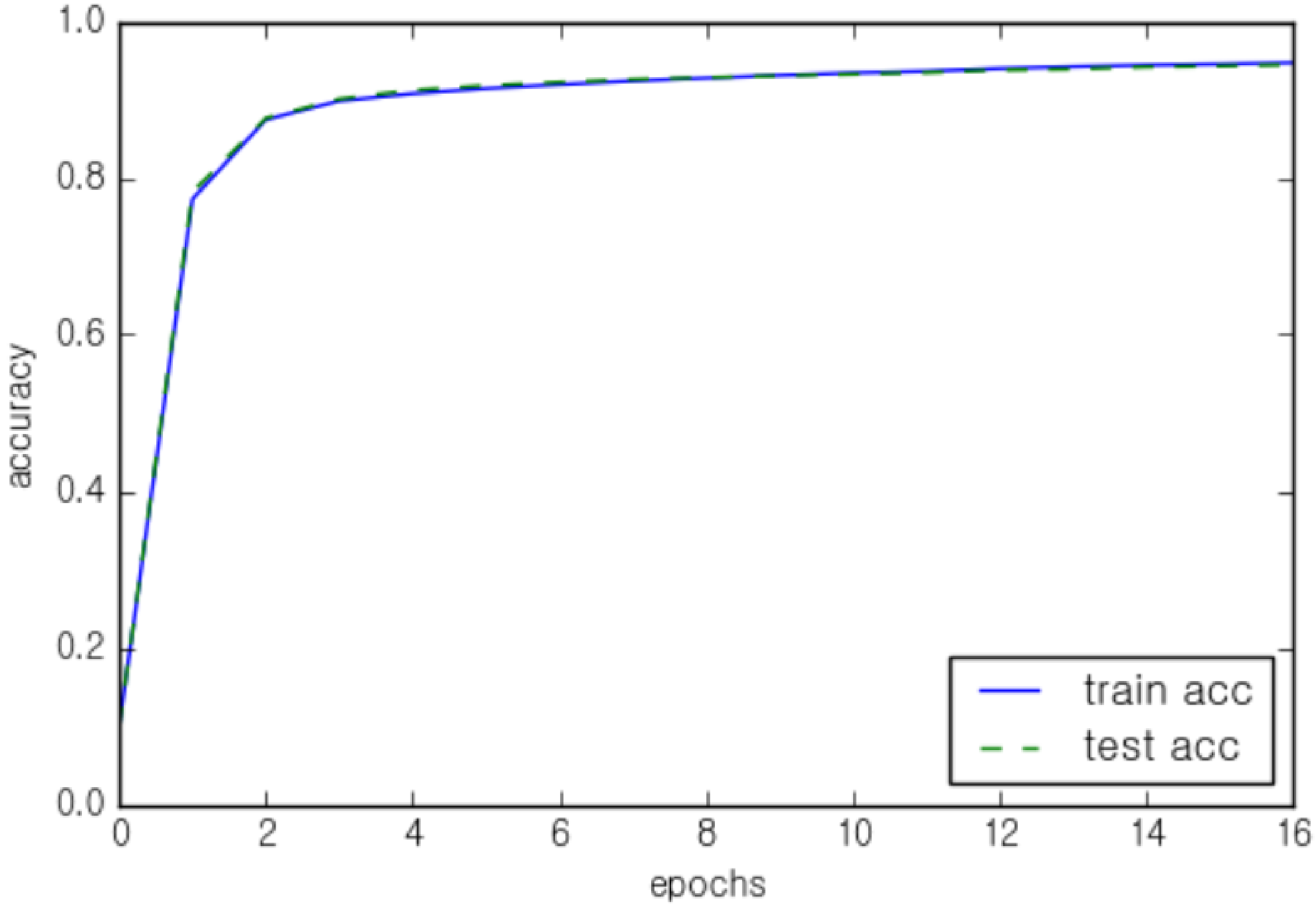
(0, 3.0)



⇒ 학습 회수가 늘어나면서 손실 함수의 값이 줄어들고 있다.

⇒ 학습이 잘 되고 있다 !

5. 학습 알고리즘 구현하기



⇒ train, test 정확도에 차이가 없다. 오버피팅 발생하지 않았다.

⇒ 오버피팅이 발생하면, 조기 종료(early stopping)를 함으로써 오버피팅 예방 가능 !

6. 정리

6. 정리

감사합니다 ☺