

딥러닝 기초 스터디 C조



## CH4. 신경망 학습

DSL 9기 임선민

# Table of Contents

## 4.1 / 데이터에서 학습한다!

- 데이터 주도 학습
- 훈련 데이터와 시험 데이터



## 4.2 / 손실 함수

- 평균 제곱 오차
- 교차 엔트로피 오차
- 미니배치 학습



## 4.3 / 수치 미분

- 미분 & 편미분



## 4.4 / 기울기

- 경사법 (경사하강법)
- 신경망에서의 기울기

## 4.5 / 학습 알고리즘 구현하기

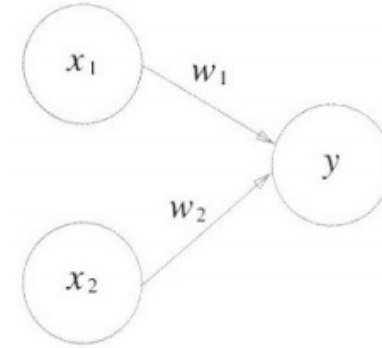
- 2층 신경망 클래스 구현하기
- 미니배치 학습 구현하기
- 시험 데이터로 평가하기

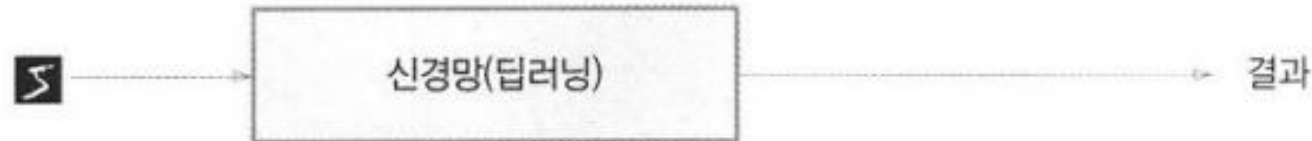
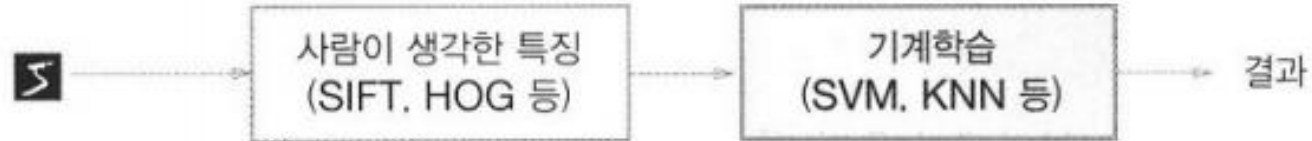
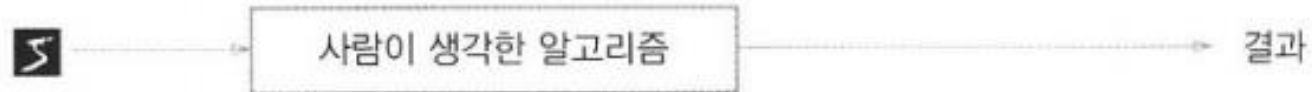
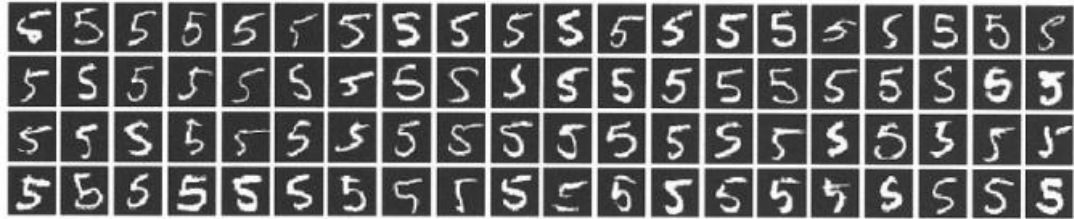
## 4.6 / 정리

## CH4. 신경망 학습

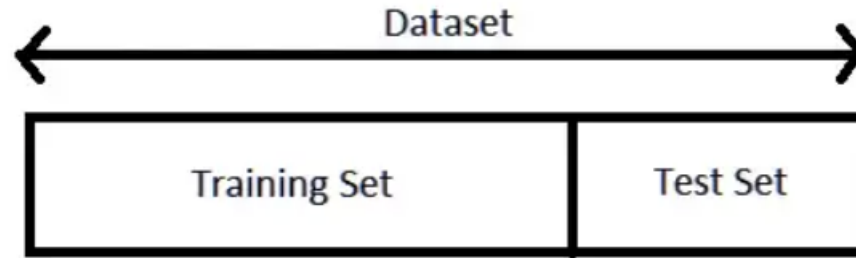
### 4.1 데이터에서 학습한다!

훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 과정





➡ end-to-end machine learning



- ✓ ☐ 범용 능력을 평가하기 위해 데이터를 나눔
- ✓ ☐ 범용 능력: 훈련 데이터에 포함되지 않은 데이터로도 문제를 올바르게 풀어내는 능력

CH4. 신경망 학습

## 4.2 손실 함수

- 신경망 학습에서 사용하는 지표
  - 현재의 신경망이 훈련 데이터를 얼마나 잘 처리하지 '못' 하느냐
- 손실함수가 최소가 되는 매개변수 값이 최적인 것임!!



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

$y_k$ : 신경망의 출력  
 $t_k$ : 정답 레이블  
(원-핫 인코딩)  
 $k$ : 데이터의 차원 수

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

```
>>> # 정답은 '2'  
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]  
>>>  
>>> # 예1 : '2'일 확률이 가장 높다고 추정함 ( 0.6 )  
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]  
>>> mean_squared_error(np.array(y), np.array(t))  
0.0975000000000000031  
>>>  
>>> # 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))  
0.597500000000000003
```



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

$y_k$ : 신경망의 출력  
 $t_k$ : 정답 레이블  
 (원-핫 인코딩)  
 $k$ : 데이터의 차원 수

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

```
>>> 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
```

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





```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=True, one_hot_label=True)
```

```
print(x_train.shape) # (60000, 784)
print(t_train.shape) # (60000, 10)
```

```
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]
```

60000장의 훈련 데이터 중  
무작위로 10장만 뽑아 사용

→ 데이터의 일부: 전체의 '근사치'

(미니배치용) 교차 엔트로피 오차 구현하기

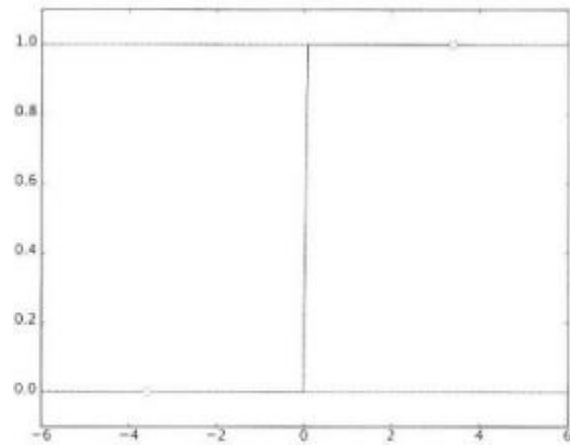
```
def cross_entropy_error(y, t):
    if y.ndim == 1: y가 1차원: reshape 함수로 데이터 형상 바꿔줌
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)
```

```
    batch_size = y.shape[0]
    return -np.sum(t * np.log(y + 1e-7)) / batch_size
```

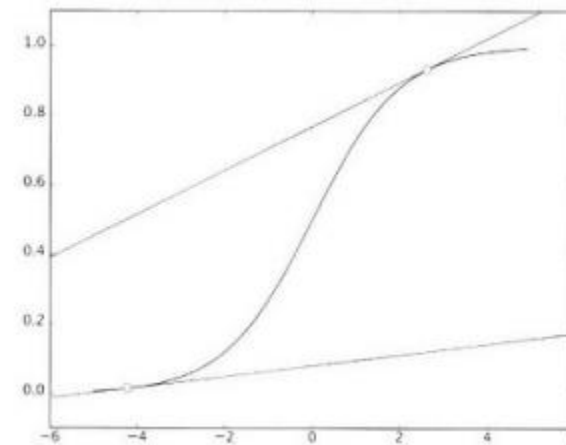
정규화: 배치의 크기로 나눠줌

## 왜 손실함수??

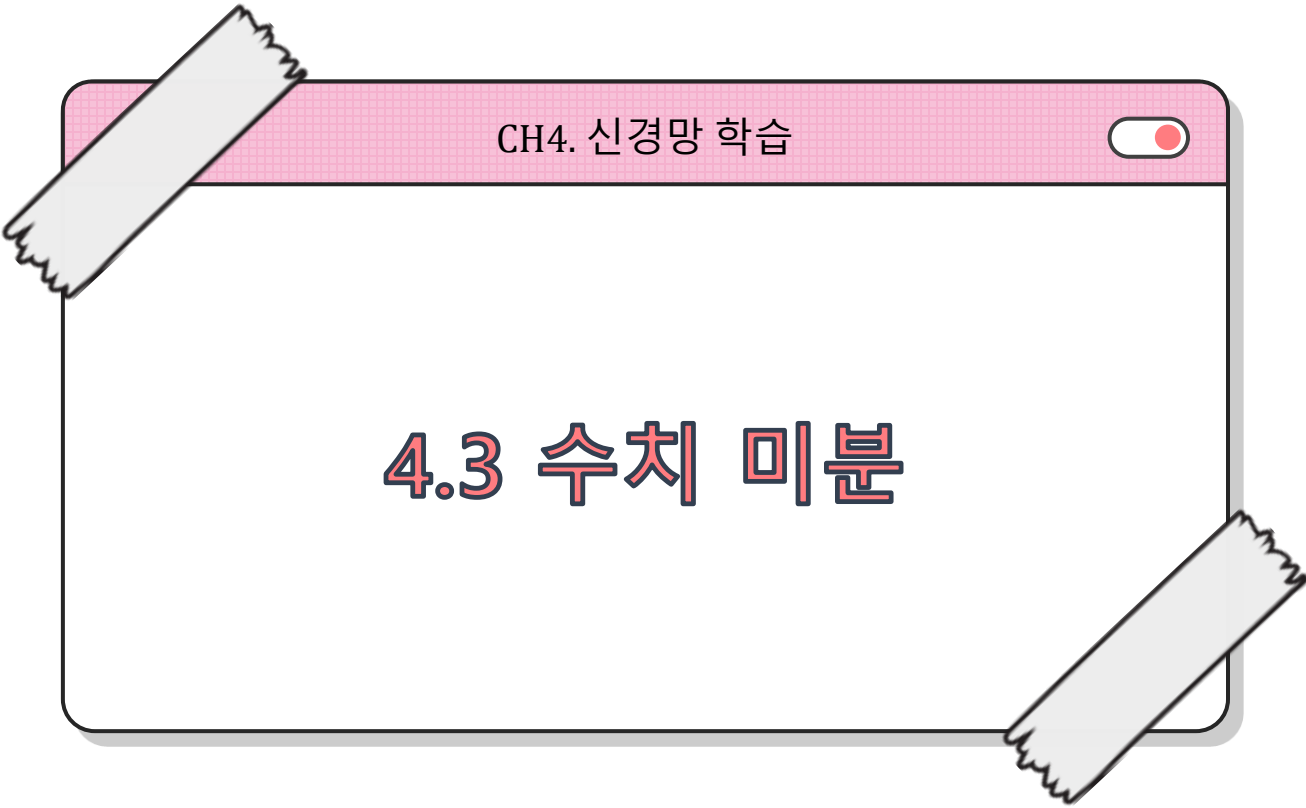
계단 함수



시그모이드 함수



- ✓ ☐ 궁극적인 목적: 높은 '정확도'를 끌어내는 매개변수 값 찾는 것
- ✓ ☐ 정확도: 지표  $X$  ; 미분 값이 대부분의 장소에서 0이 되기 때문



CH4. 신경망 학습

## 4.3 수치 미분

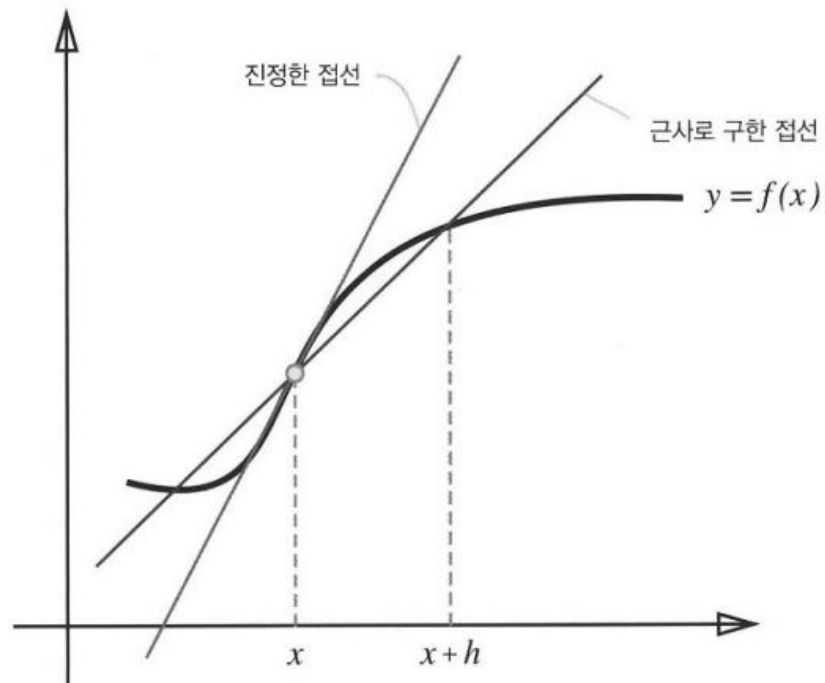
```
def numerical_diff(f, x):  
    h = 10e-50  
    return (f(x + h) - f(x)) / h
```

[개선점 2가지]

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

- 1) 반올림 오차 문제
- 2)  $f$  의 차분 (함수값들의 차이)

```
>>> np.float32(1e-50)  
0.0
```



## 미분의 예 (변수 개수에 따라)

$$y = 0.01x^2 + 0.1x$$

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

$$y' = 0.02x + 0.1$$

```
>>> numerical_diff(function_1, 5)  
0.1999999999999998  
>>> numerical_diff(function_1, 10)  
0.2999999999999998
```

### [편미분]

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

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

문제 1 :  $x_0 = 3, x_1 = 4$  일 때,  $x_0$ 에 대한 편미분  $\frac{\partial f}{\partial x_0}$  를 구하라.

---

```
>>> def function_tmp1(x0):  
...     return x0*x0 + 4.0**2.0  
...  
>>> numerical_diff(function_tmp1, 3.0)  
6.000000000000000
```

---

문제 2 :  $x_0 = 3, x_1 = 4$  일 때,  $x_1$ 에 대한 편미분  $\frac{\partial f}{\partial x_1}$  를 구하라.

---

```
>>> def function_tmp2(x1):  
...     return 3.0**2.0 + x1*x1  
...  
>>> numerical_diff(function_tmp2, 4.0)  
7.999999999999999
```

---

CH4. 신경망 학습



## 4.4 기울기



기울기: 모든 변수의 편미분을 벡터로 정리한 것



기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 크게 줄이는 방향

그림 4-8  $f(x_0, x_1) = x_0^2 + x_1^2$ 의 그래프

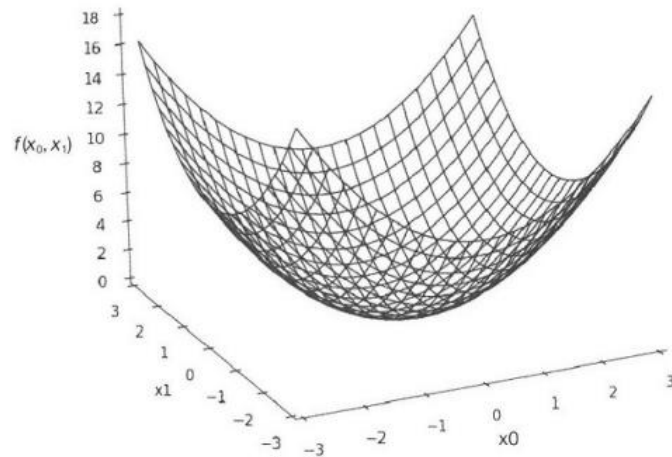
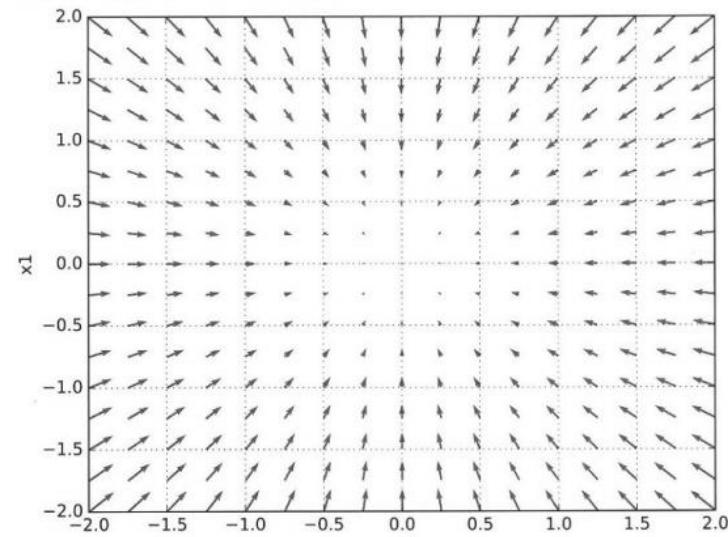


그림 4-9  $f(x_0, x_1) = x_0^2 + x_1^2$ 의 기울기



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

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



갱신하는 양 (학습률)

```
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
```

f: 최적화하려는 함수

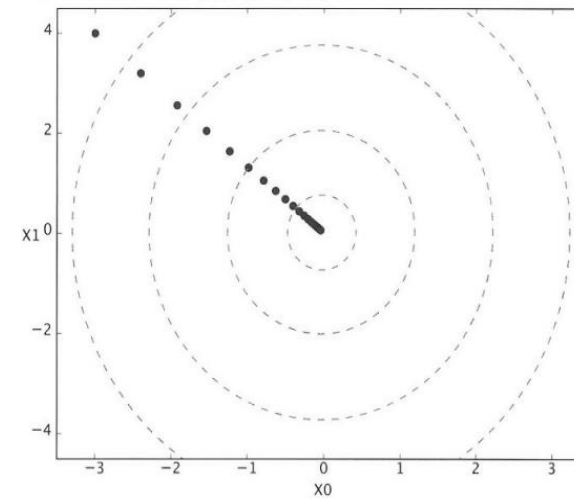
init\_x: 초깃값

lr: 학습률

step\_num: 경사법에 따른 반복 횟수

문제: 경사법으로  $f(x_0, x_1) = x_0^2 + x_1^2$ 의 최솟값을 구하라.

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





CH4. 신경망 학습



## 4.5 학습 알고리즘 구현하기



### 전제

신경망에는 적응 가능한 가중치와 편향이 있고, 이 가중치와 편향을 훈련 데이터에 적응하도록 조정하는 과정을 '학습'이라 합니다. 신경망 학습은 다음과 같이 4단계로 수행합니다.

### 1단계 - 미니배치

훈련 데이터 중 일부를 무작위로 가져옵니다. 이렇게 선별한 데이터를 미니배치라 하며, 그 미니배치의 손실 함수 값을 줄이는 것이 목표입니다.

### 2단계 - 기울기 산출

미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구합니다. 기울기는 손실 함수의 값을 가장 작게 하는 방향을 제시합니다.

### 3단계 - 매개변수 갱신

가중치 매개변수를 기울기 방향으로 아주 조금 갱신합니다.

### 4단계 - 반복

1~3단계를 반복합니다.

CH4. 신경망 학습



## 4.6 정리



## 이번 장에서 배운 내용

- 기계학습에서 사용하는 데이터셋은 훈련 데이터와 시험 데이터로 나눠 사용한다.
- 훈련 데이터로 학습한 모델의 범용 능력을 시험 데이터로 평가한다.
- 신경망 학습은 손실 함수를 지표로, 손실 함수의 값이 작아지는 방향으로 가중치 매개변수를 갱신한다.
- 가중치 매개변수를 갱신할 때는 가중치 매개변수의 기울기를 이용하고, 기울어진 방향으로 가중치의 값을 갱신하는 작업을 반복한다.
- 아주 작은 값을 주었을 때의 차분으로 미분하는 것을 수치 미분이라고 한다.
- 수치 미분을 이용해 가중치 매개변수의 기울기를 구할 수 있다.
- 수치 미분을 이용한 계산에는 시간이 걸리지만, 그 구현은 간단하다. 한편, 다음 장에서 구현하는 (다소 복잡한) 오차역전파법은 기울기를 고속으로 구할 수 있다.

감사합니다  
Thank you!