딥러닝 스터디

밑바닥 부터 시작하는 딥러닝

김제우

딥러닝스터디

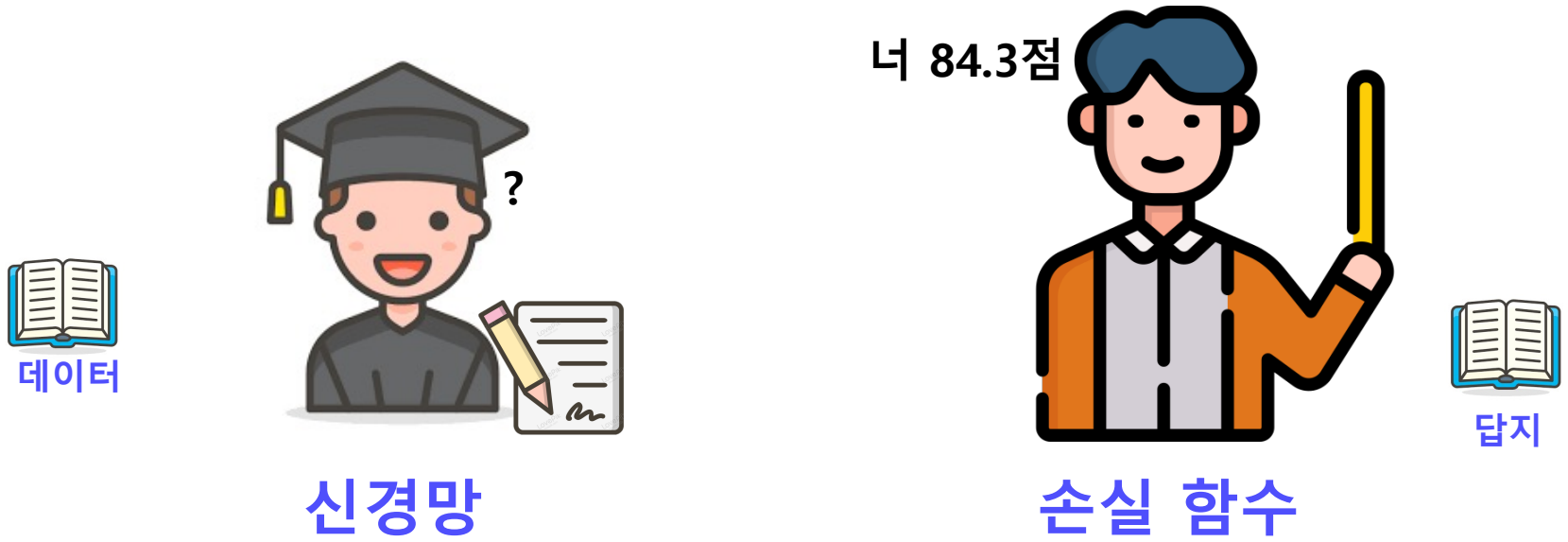
목차

4. 신경망 학습

신경망 학습

신경망 학습

- '학습'이란 데이터로부터 가중치 값들을 자동으로 획득하는 것이다.
- 신경망이 '학습'을 하기 위한 지표 : **손실 함수**



신경망 학습

데이터에서 학습한다!

- 앞서 우리가 배웠던 매개변수는 3개였다. -> 이것도 직접 다루기 힘들다.
- 실제 신경망은 매개변수가 수억개까지도 늘어난다.
- 이걸 전부 직접 정해주는 것은 불가능!

bert-base-cased

12-layer, 768-hidden, 12-heads, 110M parameters.

Trained on lower-cased English text.

roberta-base

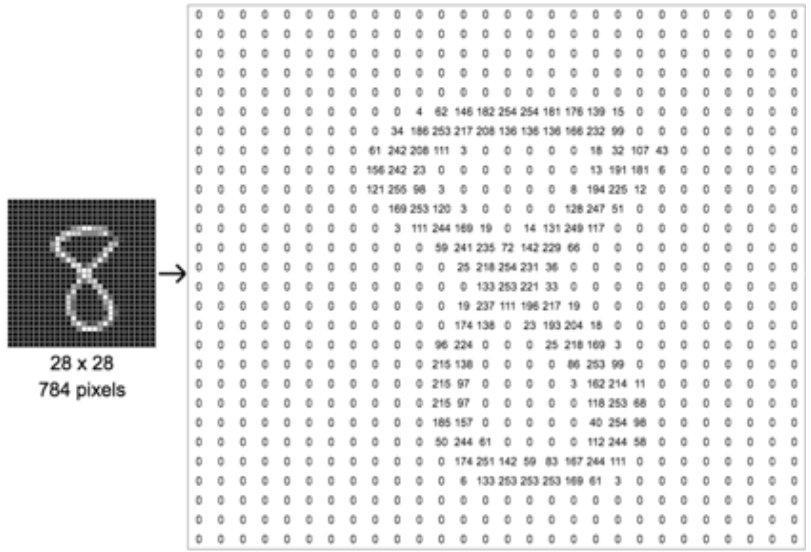
12-layer, 768-hidden, 12-heads, 125M parameters

RoBERTa using the BERT-base architecture

신경망 학습

데이터 주도 학습

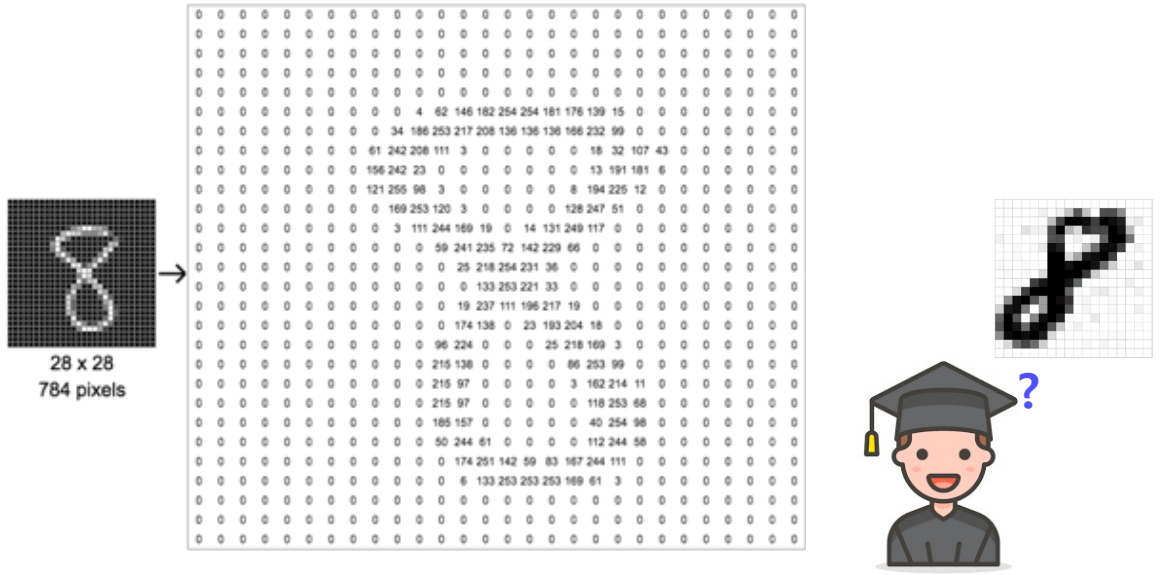
- 기계학습의 생명은 데이터!
- 데이터에서 답을 찾고 패턴을 발견하는 것이 바로 기계학습.
- 사람이 패턴을 찾아서 넣어주는 것에는 예외사항이 너무 많다.



신경망 학습

데이터 주도 학습

- 기계학습의 생명은 데이터!
- 데이터에서 답을 찾고 패턴을 발견하는 것이 바로 기계학습.
- 사람이 패턴을 찾아서 넣어주는 것에는 예외사항이 너무 많다.



신경망 학습

데이터 주도 학습

- 알고리즘을 밑바닥부터 '설계'하는 대신 주어진 데이터의 특징(feature)를 추출하고 그 특징의 패턴을 기억한다.

신경망 학습

훈련 데이터와 시험 데이터

- 기계학습 문제는 데이터를 훈련 데이터와 시험 데이터로 나눠 학습과 실험을 수행한다.

신경망 학습

훈련 데이터와 시험 데이터

- 기계학습 문제는 데이터를 훈련 데이터와 시험 데이터로 나눠 학습과 실험을 수행한다.

신경망 학습

훈련 데이터와 시험 데이터

- 한가지 데이터셋에 지나치게 최적화된 상태를 오버피팅(overfitting)이라고 합니다.

신경망 학습

손실 함수(loss function)

- 신경망 학습에서 매개변수 업데이트를 위해 사용하는 **지표**

손실 함수

오차제곱합

- 많이 쓰이는 손실 함수 (SSE)sum of squares for error

2로 나누는 이유는 미분을 했을때 2를 삭제해주기 위해서

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

오차제곱합

1+1 = 2
1+2 = 3
1+3 = 3
1+4 = 5

시험 데이터 답지

1+1 = 2
1+2 = 3
1+3 = 4
1+4 = 5

$$\frac{(2-2)^2+(3-3)^2+(3-4)^2+(5-5)^2}{2} = 0.5$$

http://www.datamarket.kr/xe/index.php?mid=board_LCmL04&document_srl=25685&listStyle=viewer

신경망 학습

오차제곱합

- 원핫 인코딩
- one-hot encoding

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

[2] ✓ 0.2s

```
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]
print(sum_squares_error(np.array(y), np.array(t)))
```

[4] ✓ 0.2s

점수 → ... 0.09750000000000003

얘가 정답일 확률이 높다.

```
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]
print(sum_squares_error(np.array(y), np.array(t)))
```

[5] ✓ 0.2s

점수 → ... 0.5975

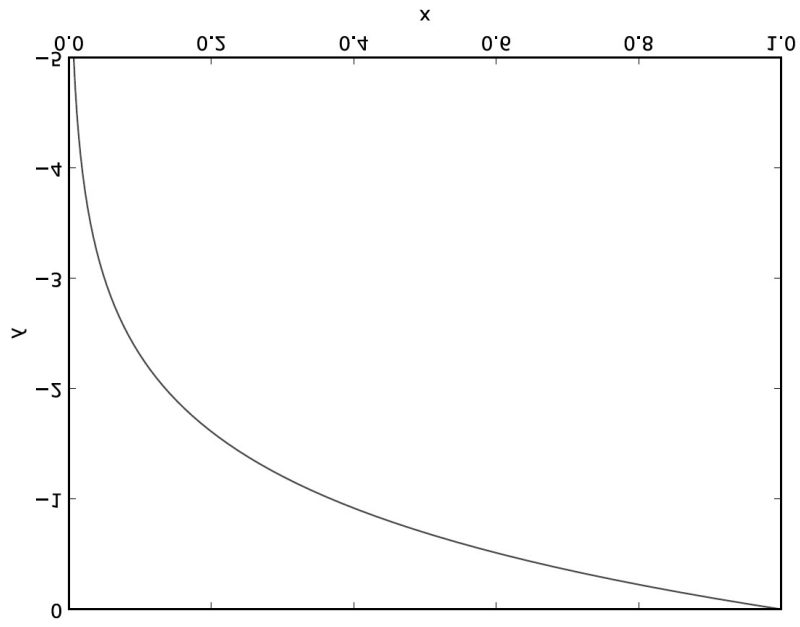
교차 엔트로피 오차

- cross entropy error

$$E = -\sum_{k} t_k \log y_k$$

- t는 정답일때만 1이고 나머지는 0이다.(원-핫)
- 실제로는 t_k가 1일 때의 결과값에만

y_k의 자연 로그를 계산하는 것이 된다.



y = logx의 그래프

그래프에서보면 x(확률)가 1일때는 y는 0에 가까워지고 x가 0.4보다 작아지면 급격히 커진다. -를 붙여주는 것은 전부다 음수의 결과값이 때문에 양수로 바꿔준는 것이다.

교차 엔트로피 오차

구현

```
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y + delta)) # delta는 y가 0일때 결과가 마이너스 무한대가 나오는 것을 방지함.
```

[6] ✓ 0.2s

```
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]
print(cross_entropy_error(np.array(y), np.array(t)))
```

[7] ✓ 0.2s

점수 → `0.510825457099338`

얘가 더 작음

```
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]
print(cross_entropy_error(np.array(y), np.array(t)))
```

[8] ✓ 0.2s

점수 → `2.302584092994546`

미니배치 학습

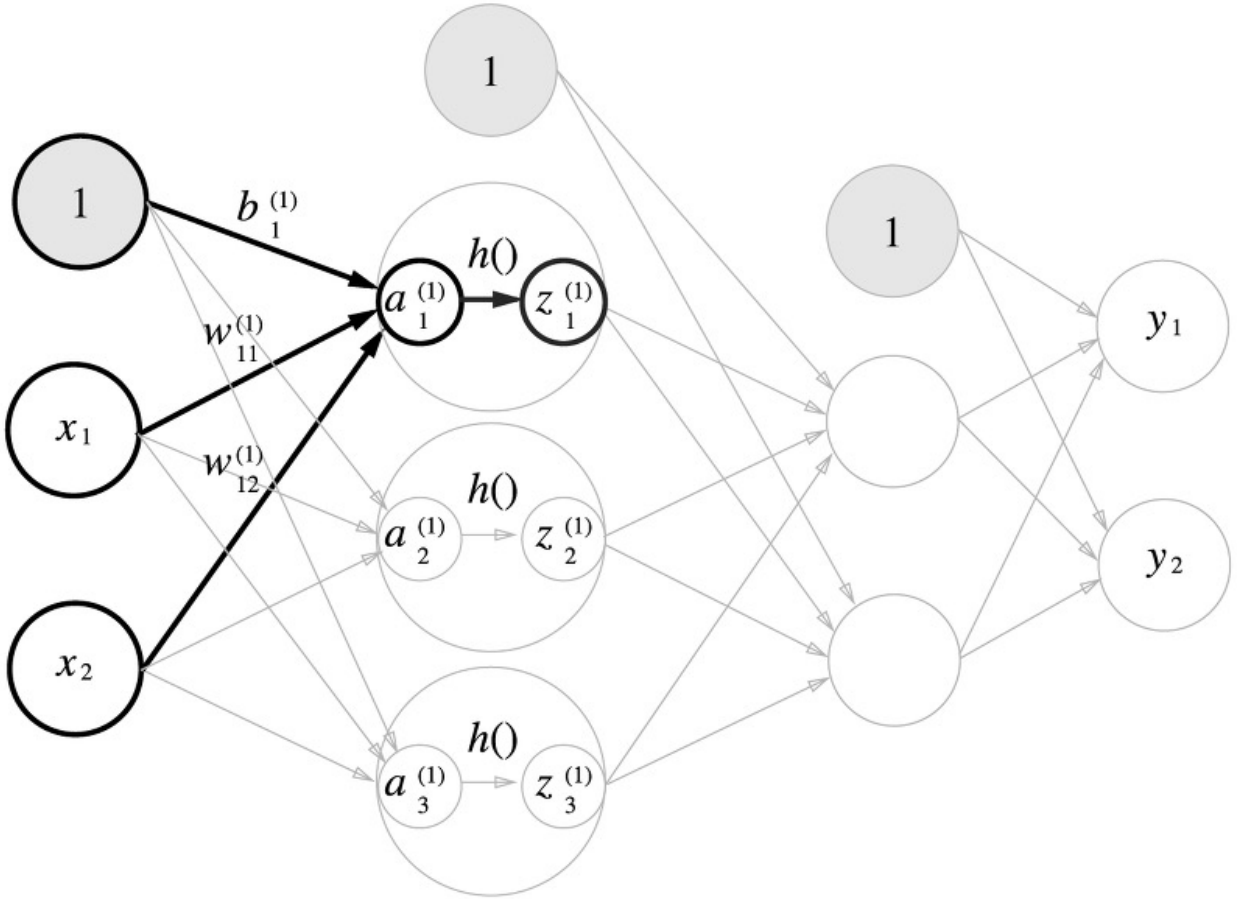
모든 배치를 한번에 연산하기에는 컴터 리소스가 부족하니 미니 배치 단위로 나눠서 loss를 구함.

loss를 배치의 갯수 만큼 다 더해서 평균낸다.

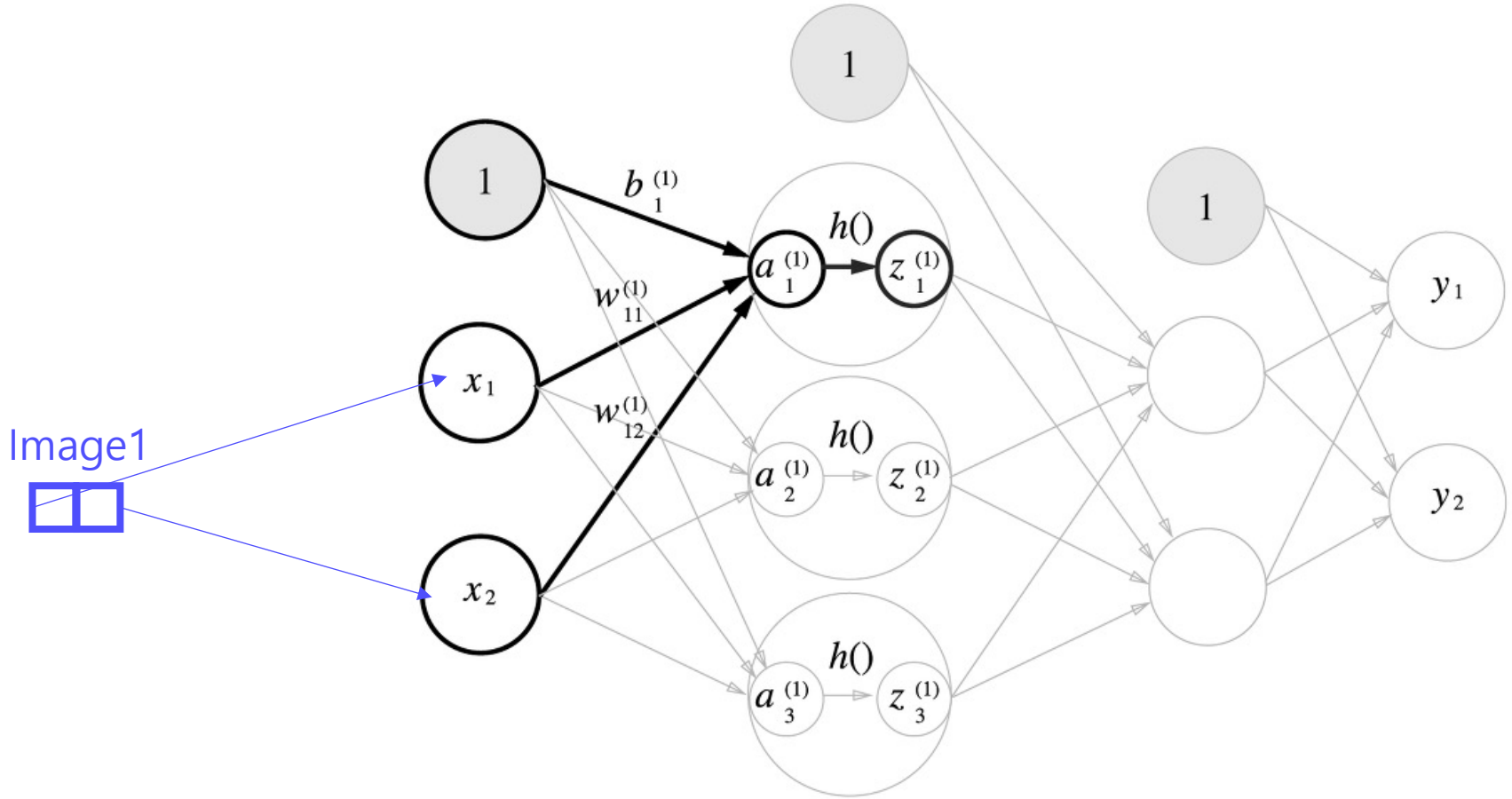
$$E = -\frac{1}{n}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

여러 데이터들의 평균적인 loss를 낮추는 방향으로 학습된다.
이렇게 하면 훈련 데이터 개수와 상관없이 언제든 통일된 지표를 얻을 수 있다.

미니배치 학습



미니배치 학습

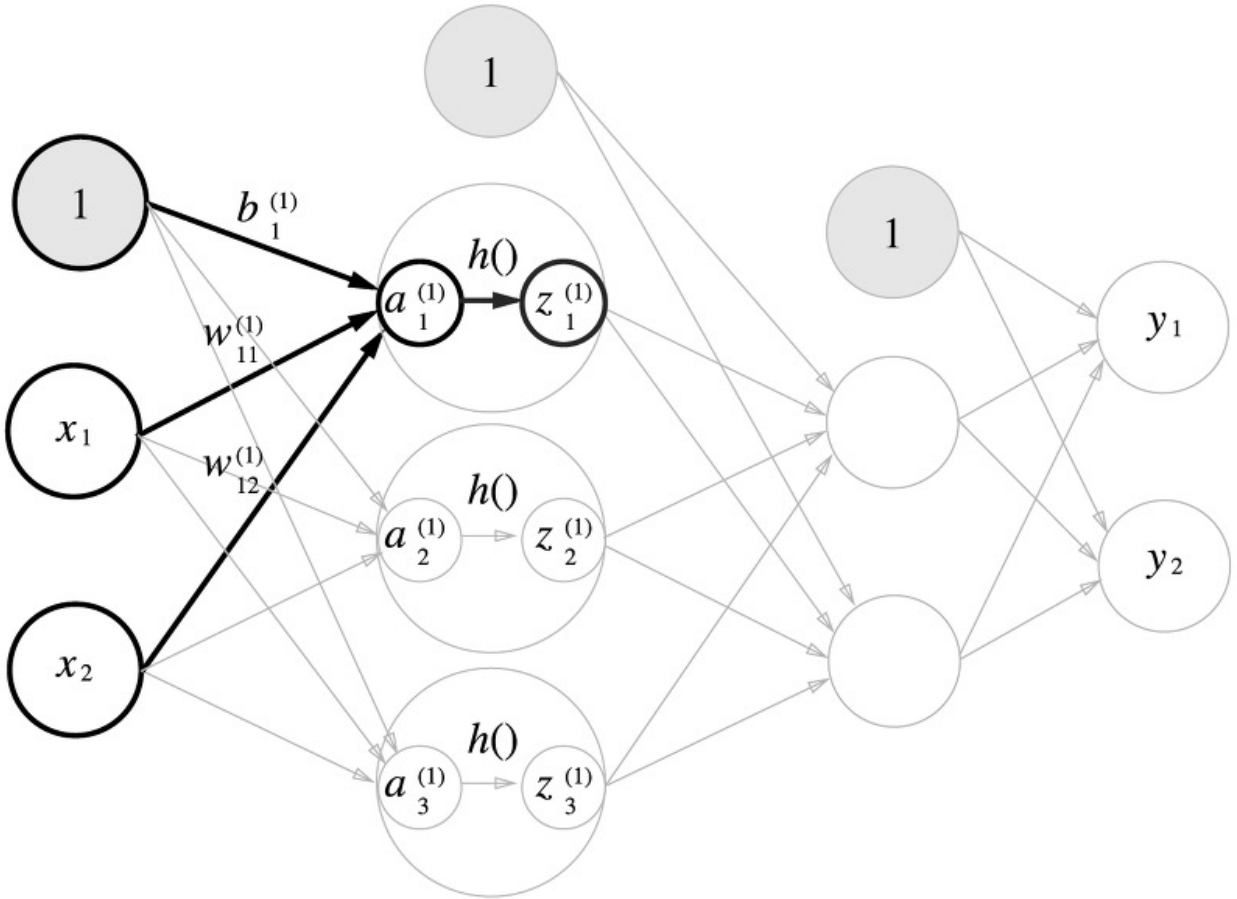


미니배치 학습

Image1

Image2

Image3



미니배치 학습

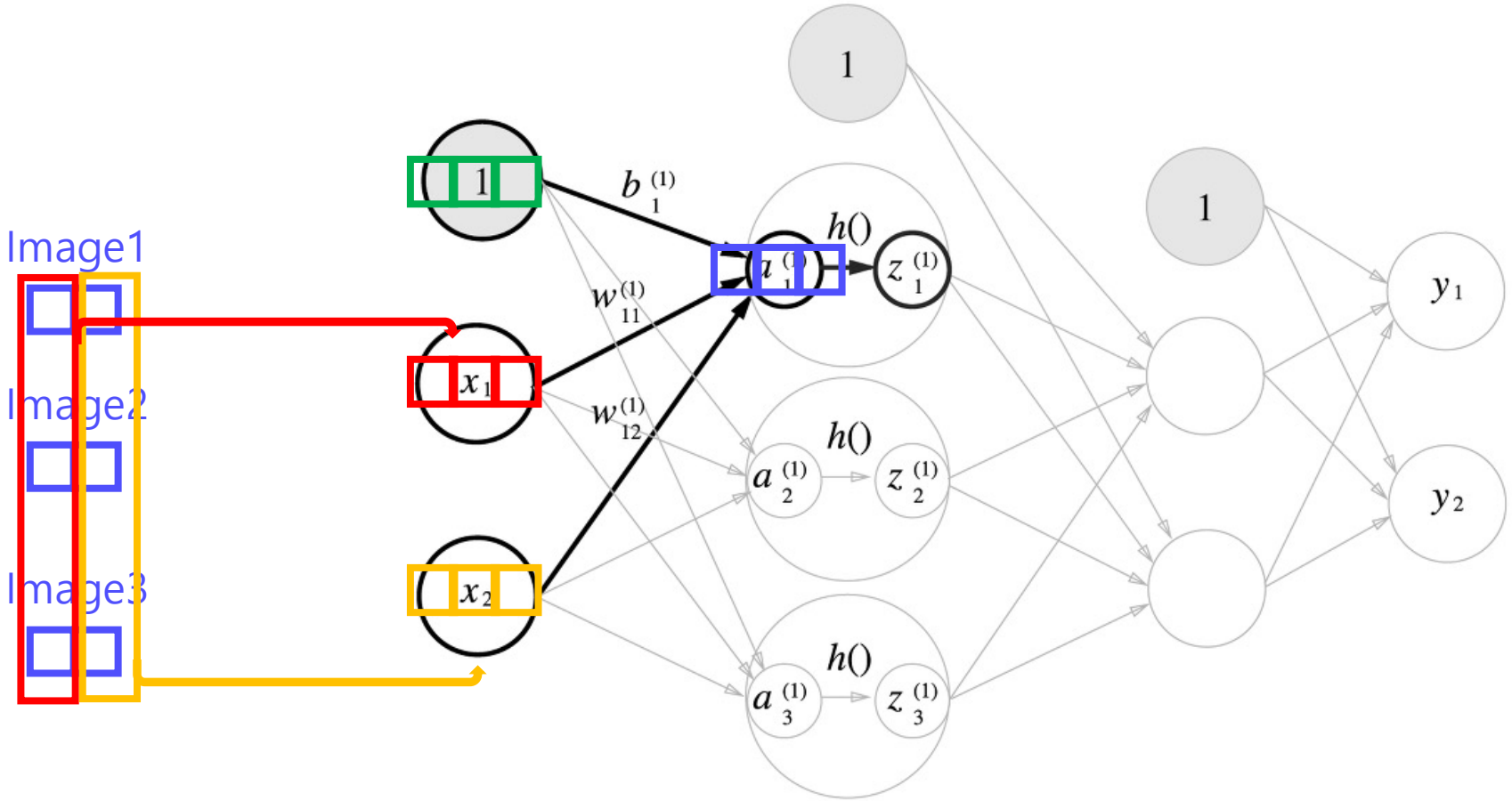


Image1
Image2
Image3
1
x_1
x_2
$b^{(1)}_1$
$w^{(1)}_{11}$
$w^{(1)}_{12}$
h()
$a^{(1)}_1$
$z^{(1)}_1$
$a^{(1)}_2$
$z^{(1)}_2$
$a^{(1)}_3$
$z^{(1)}_3$
1
y_1
y_2

미니배치 학습

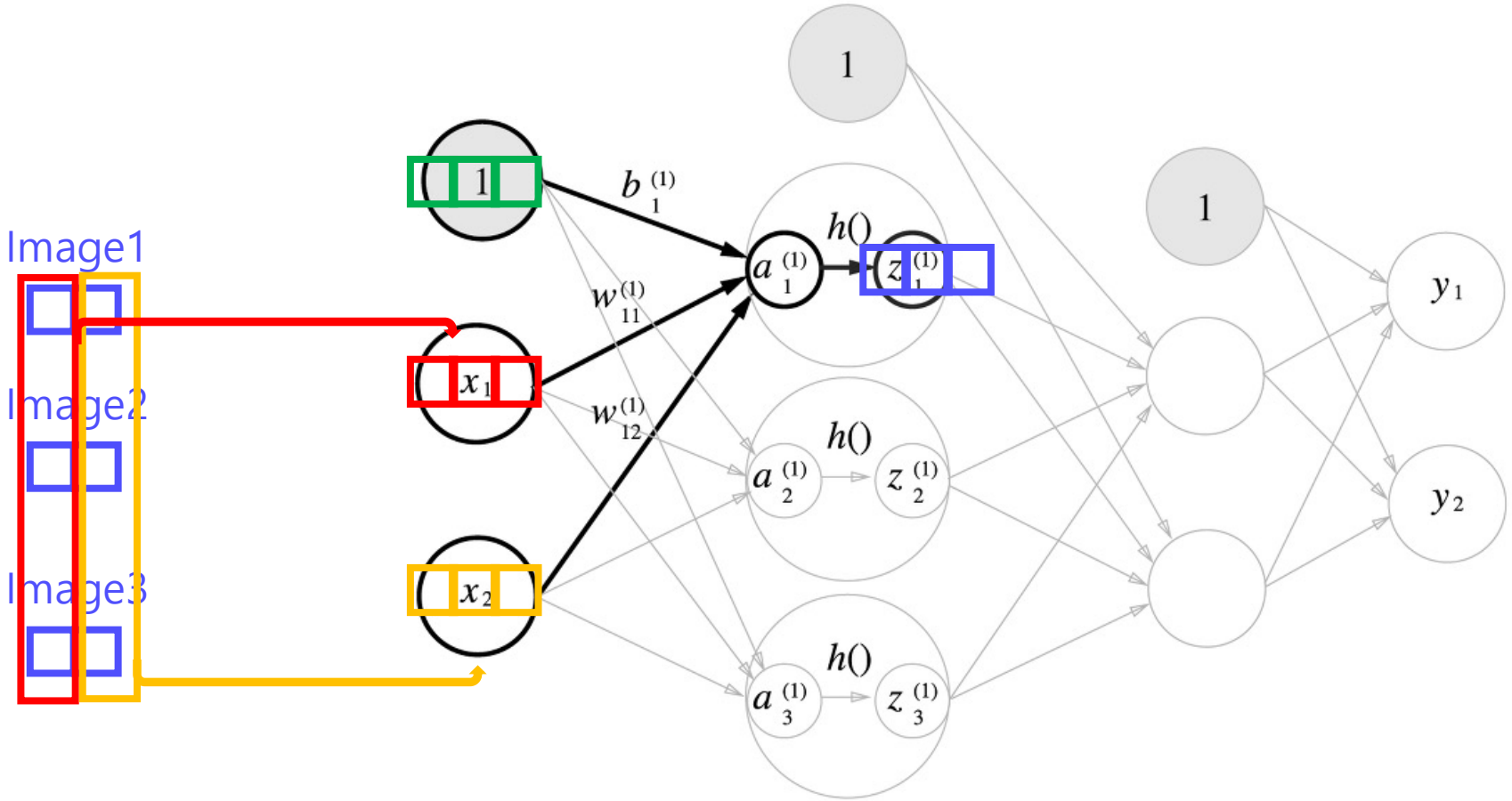


Image1
Image2
Image3
1
x_1
x_2
$b^{(1)}_1$
$w^{(1)}_{11}$
$w^{(1)}_{12}$
h()
$a^{(1)}_1$
$z^{(1)}_1$
$a^{(1)}_2$
$z^{(1)}_2$
$a^{(1)}_3$
$z^{(1)}_3$
1
y_1
y_2

미니배치 학습

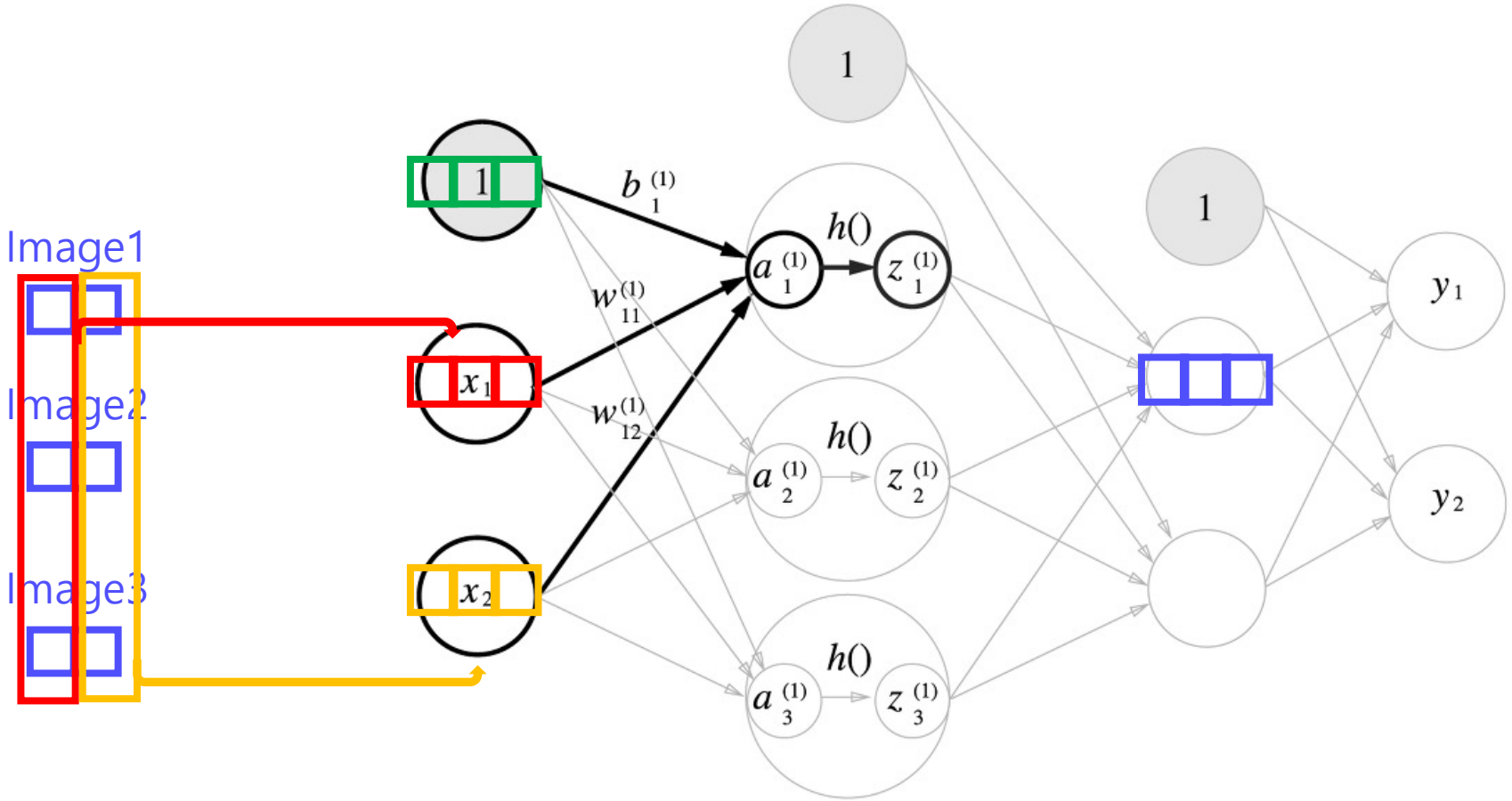


Image1
Image2
Image3
1
x_1
x_2
$b^{(1)}_1$
$w^{(1)}_{11}$
$w^{(1)}_{12}$
h()
$a^{(1)}_1$
$z^{(1)}_1$
$a^{(1)}_2$
$z^{(1)}_2$
$a^{(1)}_3$
$z^{(1)}_3$
1
y_1
y_2

미니배치 학습

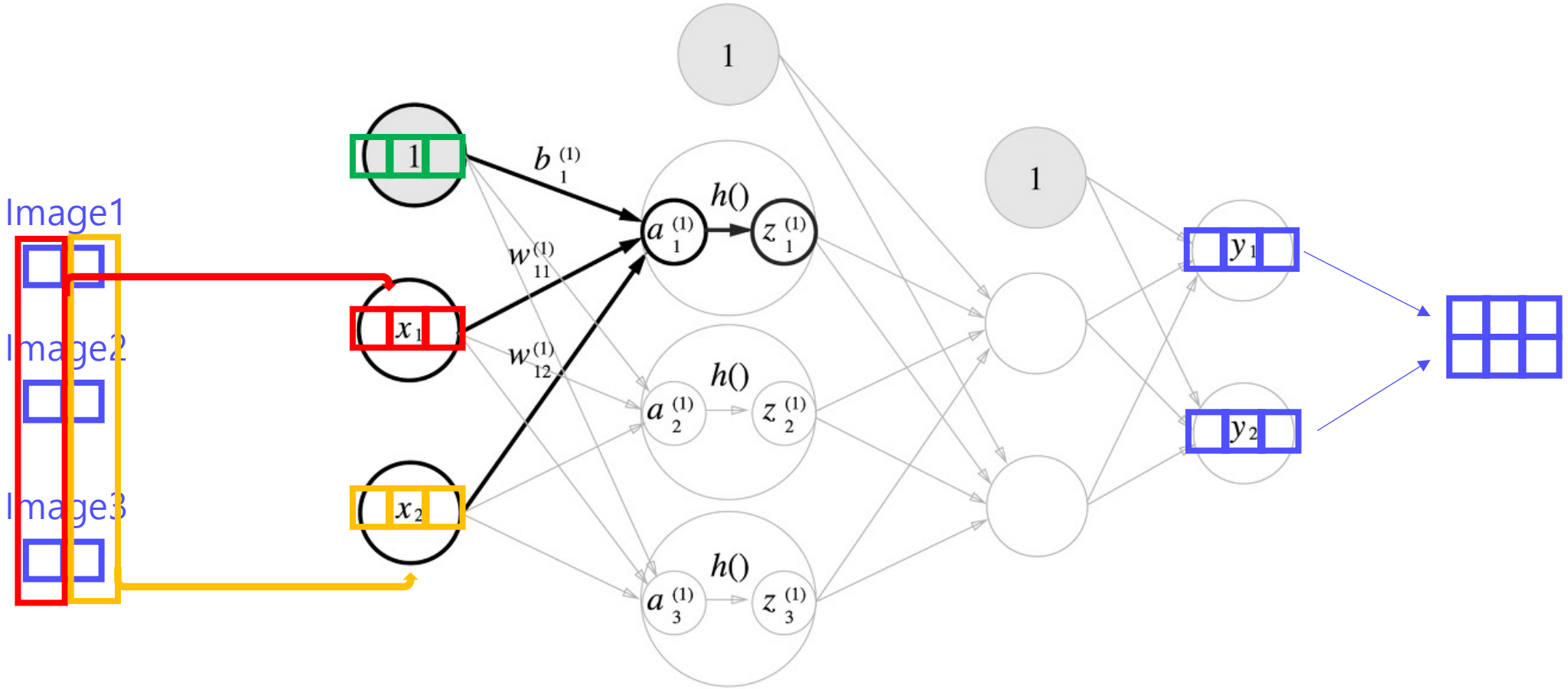


Image1
Image2
Image3
1
1
$b^{(1)}_1$
x_1
x_2
$w^{(1)}_{11}$
$w^{(1)}_{12}$
h()
$a^{(1)}_1$
$z^{(1)}_1$
h()
$a^{(1)}_2$
$z^{(1)}_2$
h()
$a^{(1)}_3$
$z^{(1)}_3$
1
y_1
y_2

신경망 학습

미니배치 학습

신경망이 제출한 답

Image1	Image2	Image3
0.6	0.5	0.9
0.4	0.5	0.1

라벨링 된 답

Image1	Image2	Image3	
1	0	1	개
0	1	0	고양이

미니배치 학습

0.6	0.5	0.9
0.4	0.5	0.1

1	0	1
0	1	0

$$E = -\frac{1}{n}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

Image1 Image2 Image3

$$-\frac{1}{3}((1 * \log(0.6)) + (0 * \log(0.4)) + (0 * \log(0.5)) + (1 * \log(0.5)) + (1 * \log(0.6)) + (0 * \log(0.4)))$$

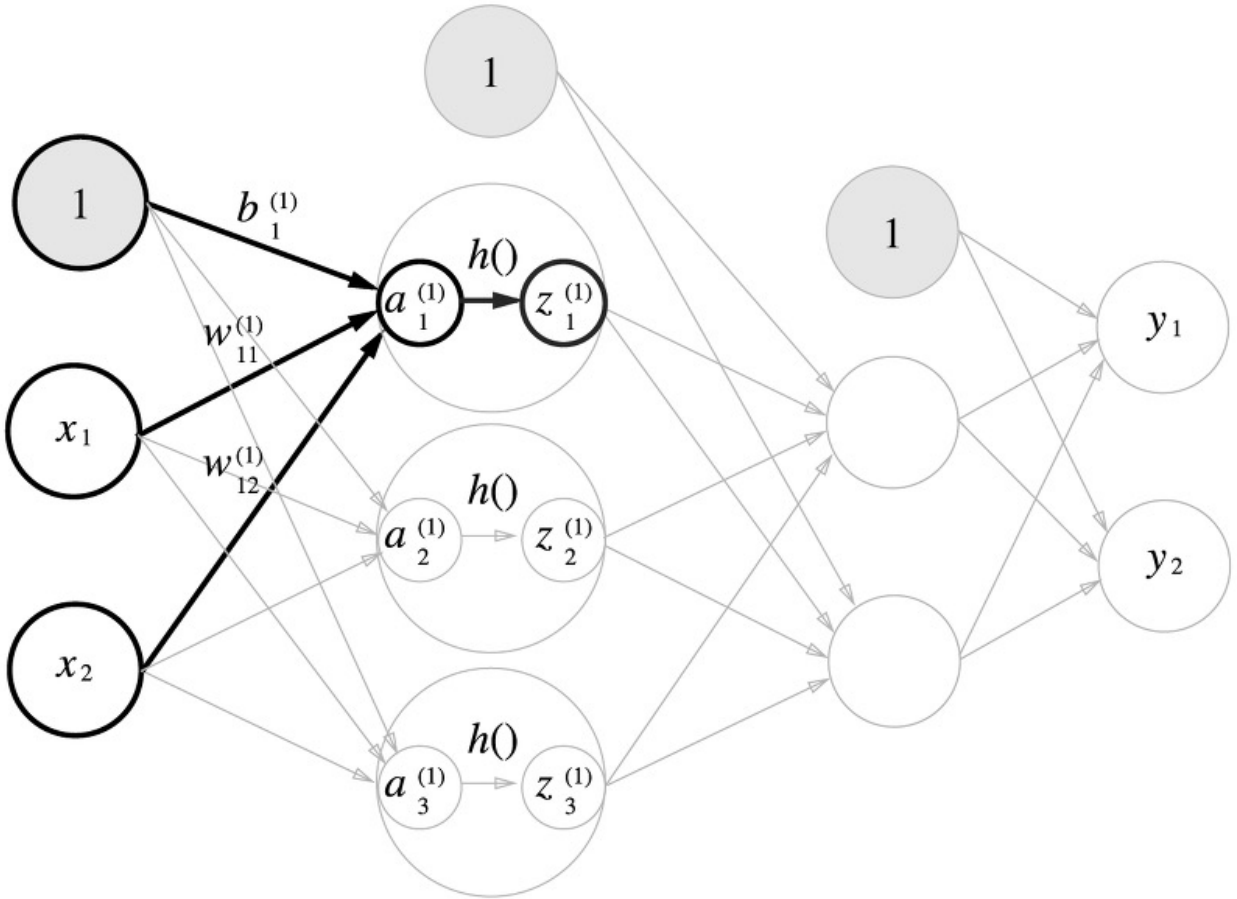
신경망 학습

미니배치 학습

Image1

Image2

Image3



loss = 0.6774

미니배치 학습

```
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)
print(t_train.shape)
```

미니배치 학습

```
[9]
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]
```

```
[11]
np.random.choice(60000,10)
```

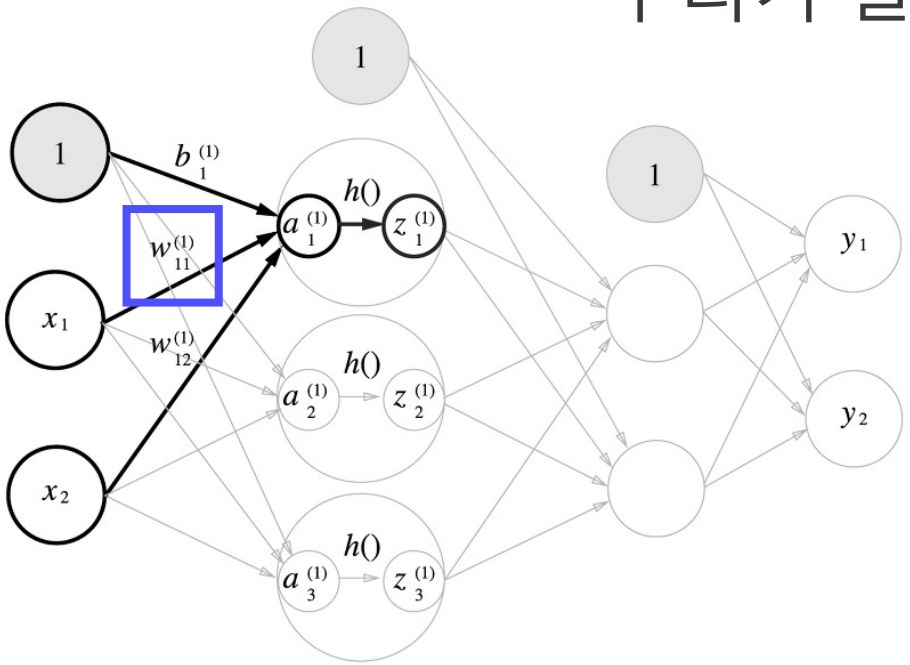
```
... array([22395, 21964, 20880, 54683, 32309, 53400, 55977,  6533, 19250,
           53492])
```

신경망 학습

왜 손실 함수를 설정하는가?

- 가중치를 아주 조금 변화 시켰을 때 손실함수가 어떻게 변하는지를 알아야 가중치를 조절할 수 있음!

우리가 알려고 하는것! $\frac{d\,\text{손실함수}}{dw}$

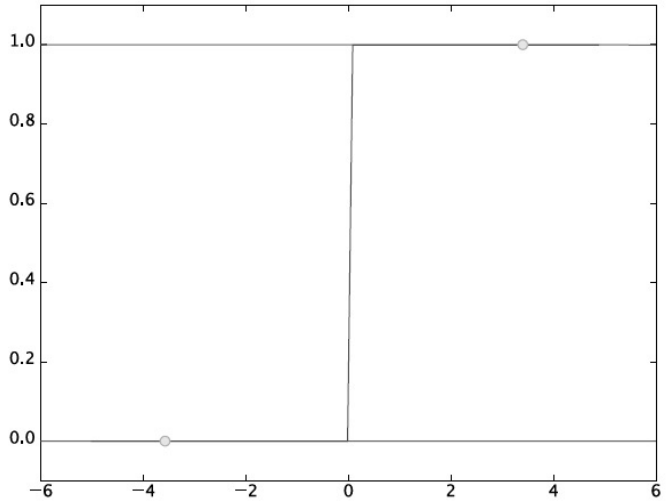


loss = 0.6774

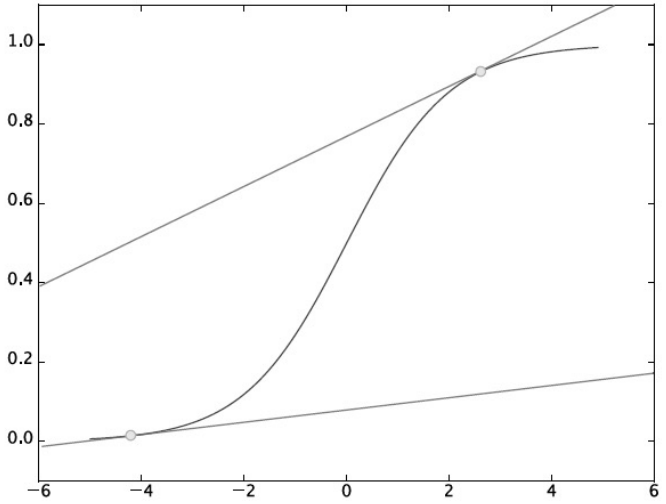
왜 손실 함수를 설정하는가?

- 손실함수를 정확도 같은 걸 쓴다면 변화량의 폭이 너무 커질 수 있음.
- 같은 이유로 시그모이드 함수는 변화량을 한 지점에서의 변화량을 구할 수 있는데 계단 함수는 대부분의 장소에서 기울기가 0이기 때문에 안씀

계단 함수



시그모이드 함수



수치 미분

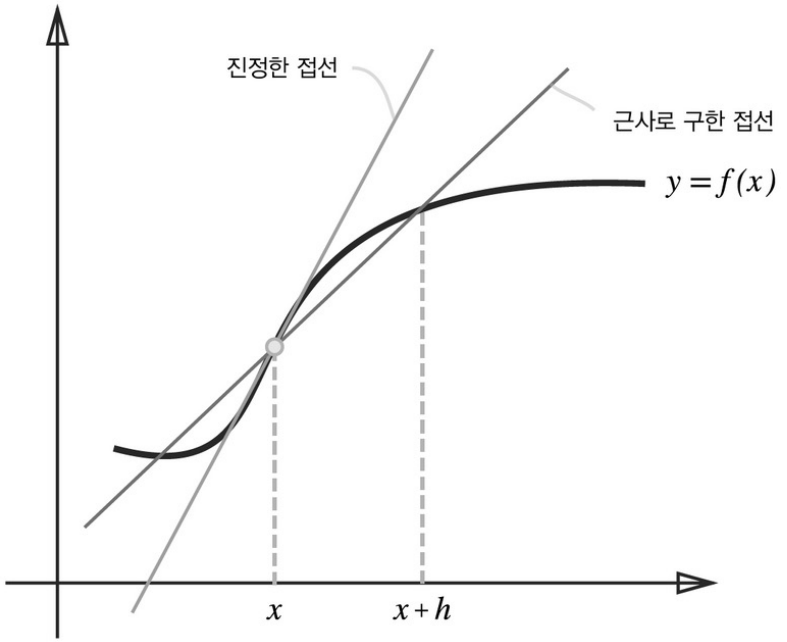
- 미분을 프로그래밍으로 구현할 때 주의사항 1
- 0에 가까운 숫자를 표현할 때 너무 작으면 반올림 오차가 발생할 수 있음
- 10^{-4}정도 추천!

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

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

수치 미분

- 미분을 프로그래밍으로 구현할 때 주의사항 2
- 도함수 표현식 $$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



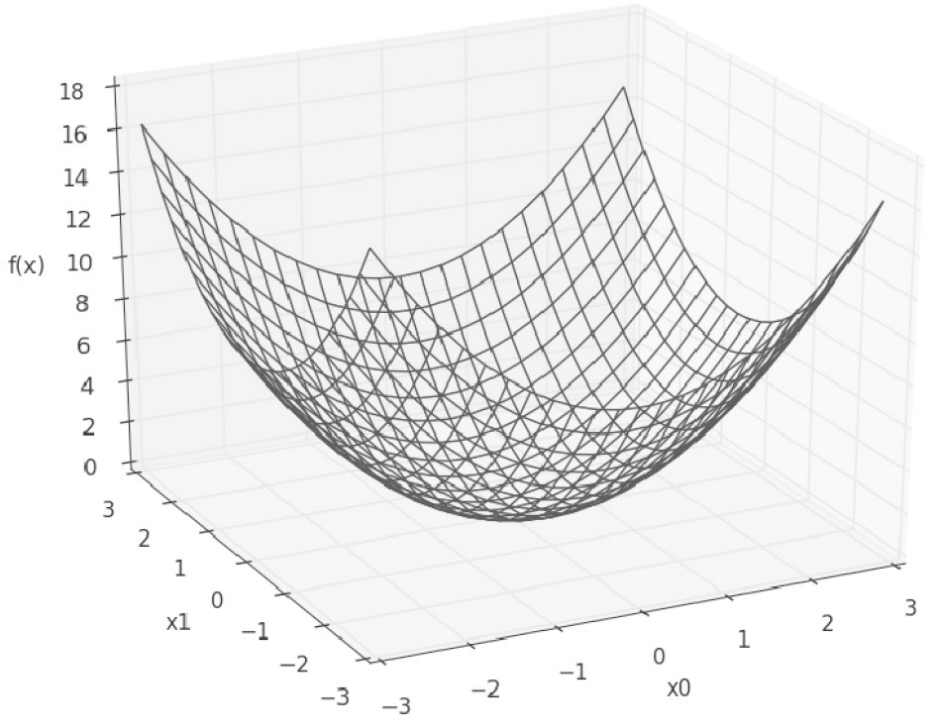
- 이렇게 하면 근사 접선이 제대로 생기지 않음
- 이걸 전방 차분이라고 한다.
- 프로그래밍에서는 x-h부터 x+h까지의 변화량을 구함
- 이걸 중심 차분 혹은 중앙 차분이라고함

수치 미분

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

편미분

- 한가지 변수에 대해서 미분 & 다른 변수는 상수로 취급



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

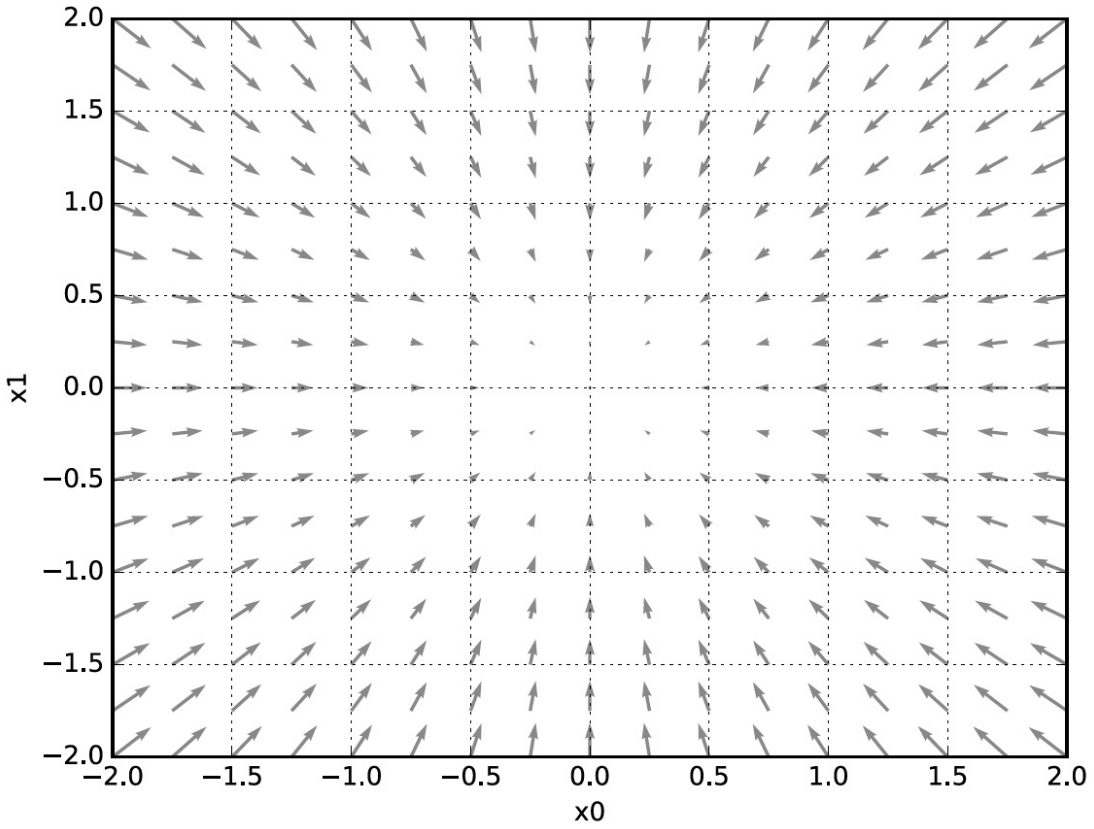
편미분

- 모든 변수에 대한 편미분 값을 저장한 벡터를 gradient vector라고 부름

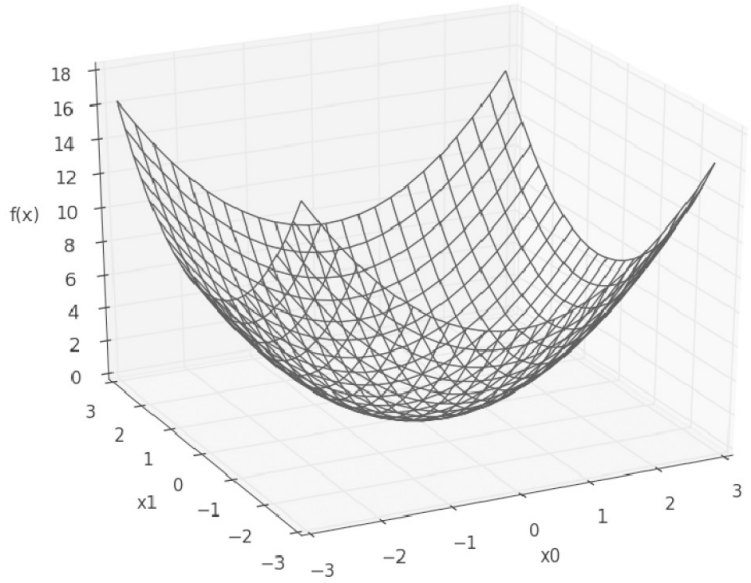
```python
def numerical_gradient(f, x):
    h = 1e-4
    grad = np.zeros_like(x)

    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = tmp_val + h
        fxh1 = f(x)

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

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

    return grad
```

편미분

- 기울기(gradient)가 가리키는 쪽은 함수의 출력을 가장 크게 줄이는 방향!



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

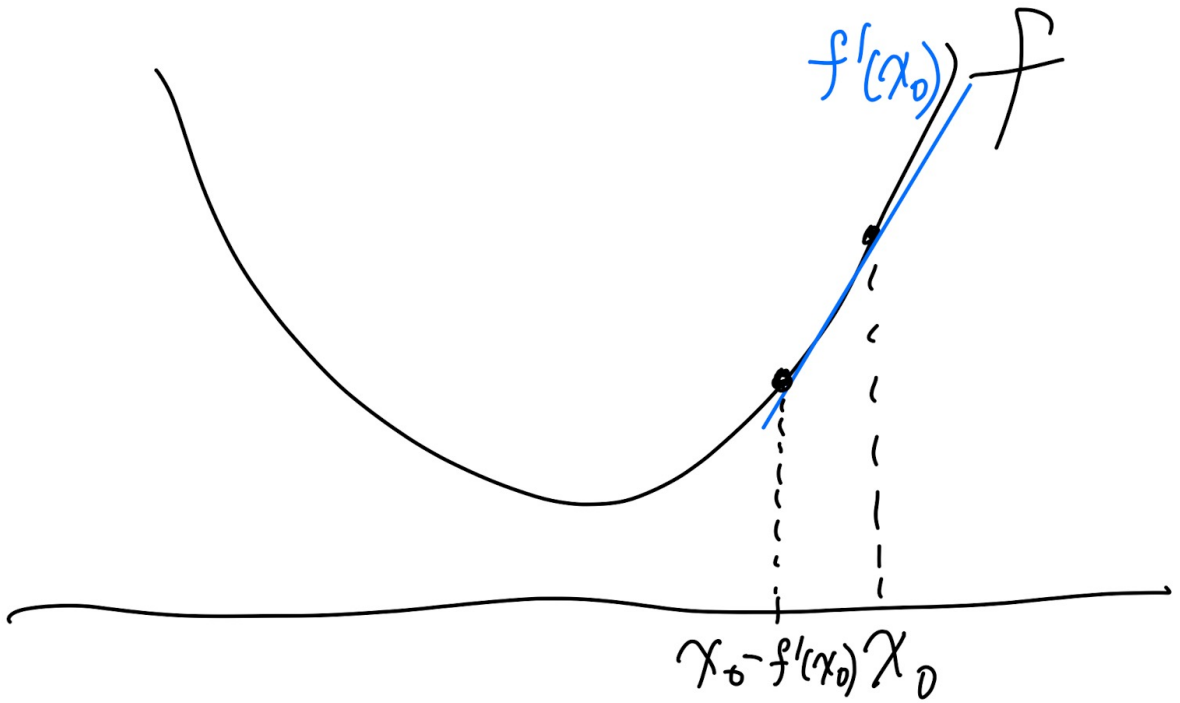


경사 하강법

- 함수값에서 미분값을 빼서 함수의 최소값을 찾는 것
- η 는 학습률 (한 번에 얼만큼 갱신 할 건지)

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

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

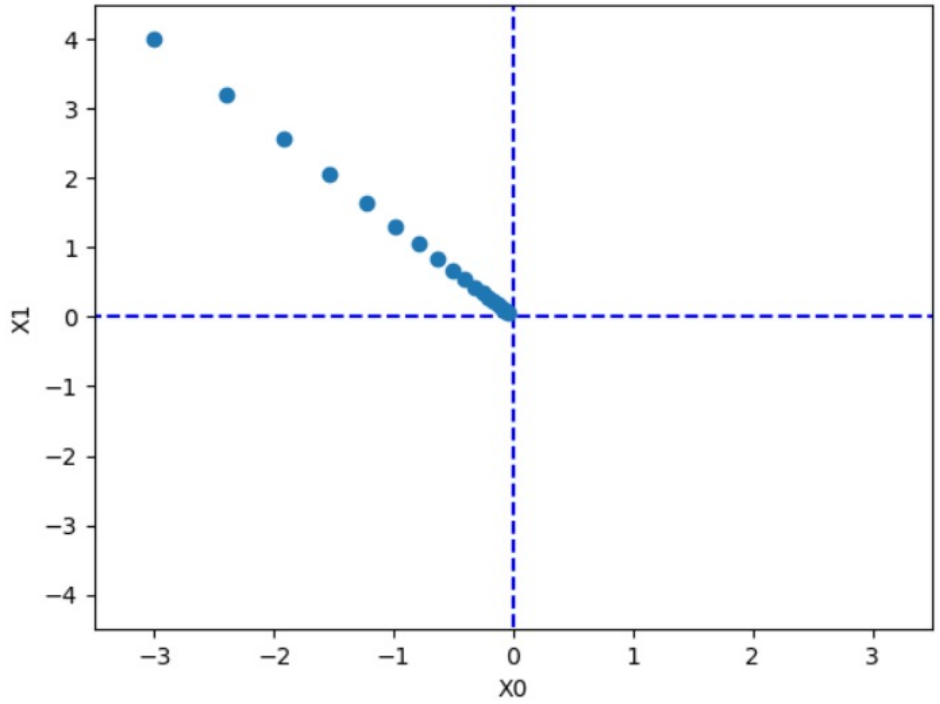


경사 하강법

- 함수값에서 미분값을 빼서 함수의 최소값을 찾는 것
- η 는 학습률 (한 번에 얼만큼 갱신 할 건지)

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

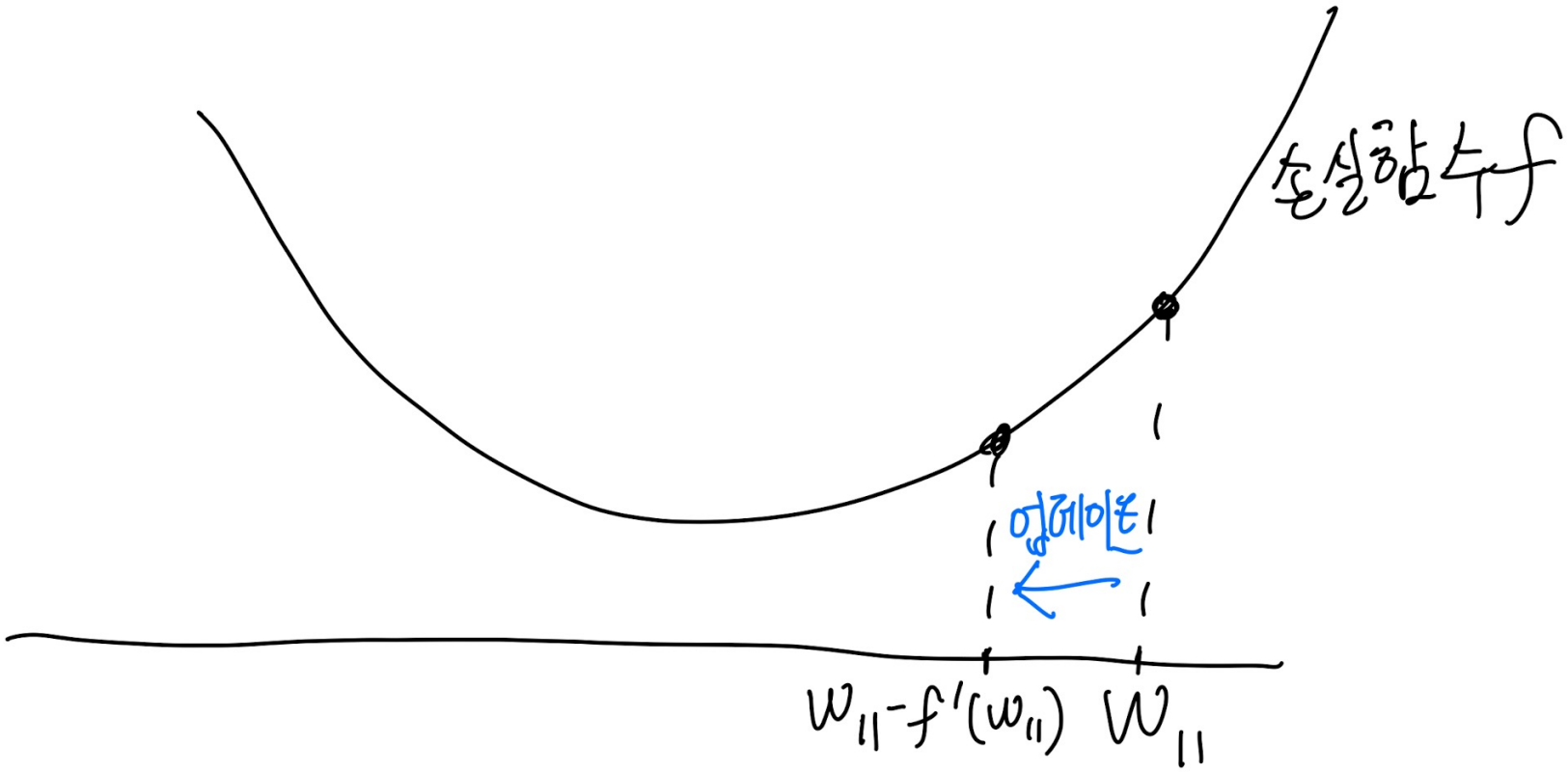
$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$



신경망 학습

경사하강법

- 신경망에서의 기울기



$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

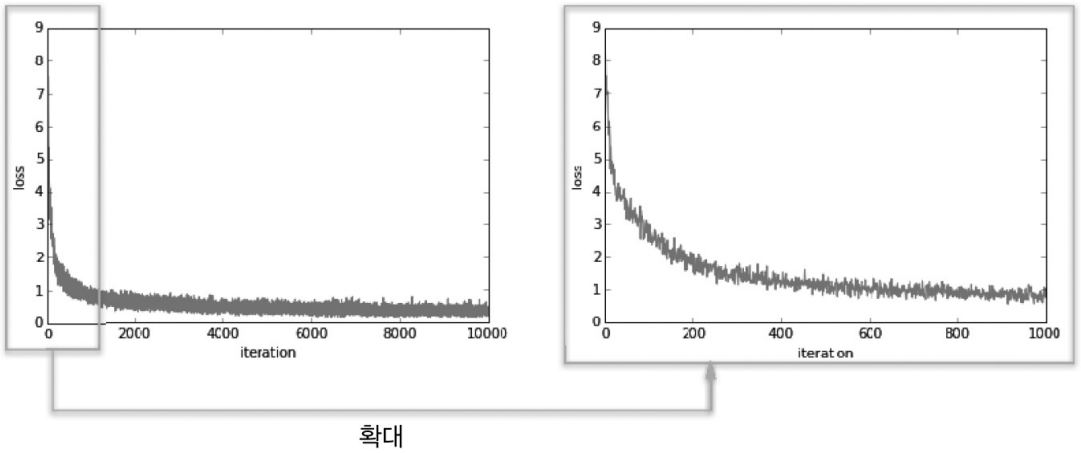
$$\frac{\partial L}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{31}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{32}} \end{pmatrix}$$

학습 알고리즘 구현하기

- 전제
  - 신경망에는 적응 가능한 가중치와 편향이 있고, 가중치와 편향을 훈련 데이터에 적응하도록 조정하는 과정을 '학습'이라 합니다.
- 1단계 미니배치
  - 훈련 데이터 중 일부를 가져와서 미니배치를 만들고 미니배치의 손실 함수 값을 줄이는 것이 목표입니다.
- 2단계 기울기 산출
  - 미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구합니다.
  - 기울기는 손실 함수 값을 가장 작게 하는 방향을 제시합니다.
- 3단계 매개변수 갱신
  - 가중치 매개변수를 기울기 방향으로 아주 조금 갱신합니다.
- 4단계 반복
  - 1~3단계 반복

학습 알고리즘 구현하기

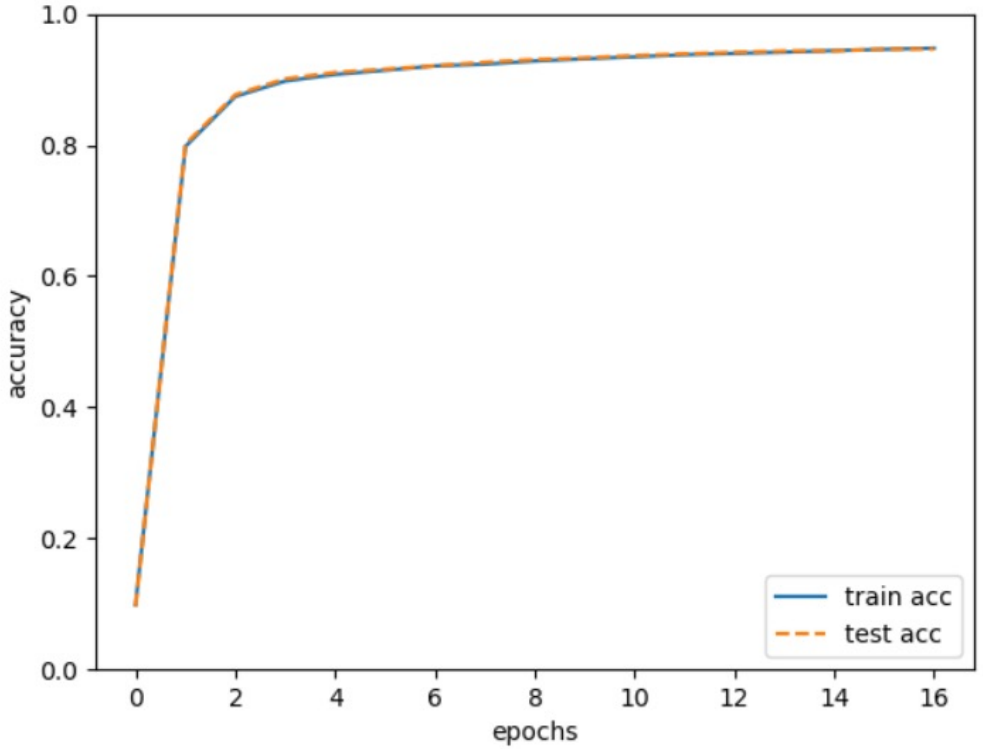
- 학습 결과




```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from 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 = []

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)
```

학습 알고리즘 구현하기

- 시험데이터로 평가



에폭(epoch)은 하나의 단위이다.

1에폭은 학습에서 훈련 데이터를
모두 소진했을 때의 횟수에 해당한다.

정리

- 기계 학습에서 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, 구현은 간단하다. 다음장에서는 기울기를 빠르게 구하는 오차역전파법을 학습한다.