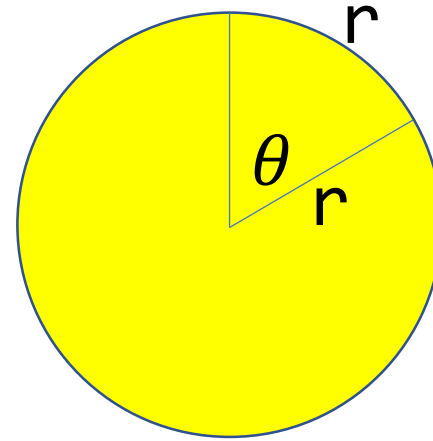


```
import numpy as np
```

```
t = np.arange(0,12,0.01)
```

```
# print(len(t))
```

```
y = np.sin(t)
```



$$\theta = 57.295779513082320876$$

$$2\pi r \times \frac{\theta}{360} = r$$

$$\theta = 180 / \pi$$

$$1 : 180 / \pi = x : 90$$

$$90 = 180 * x / \pi$$

$$\frac{\pi}{180} * degree = x$$

```
plt.figure(figsize=(10,6))
```

```
plt.subplot(221)
```

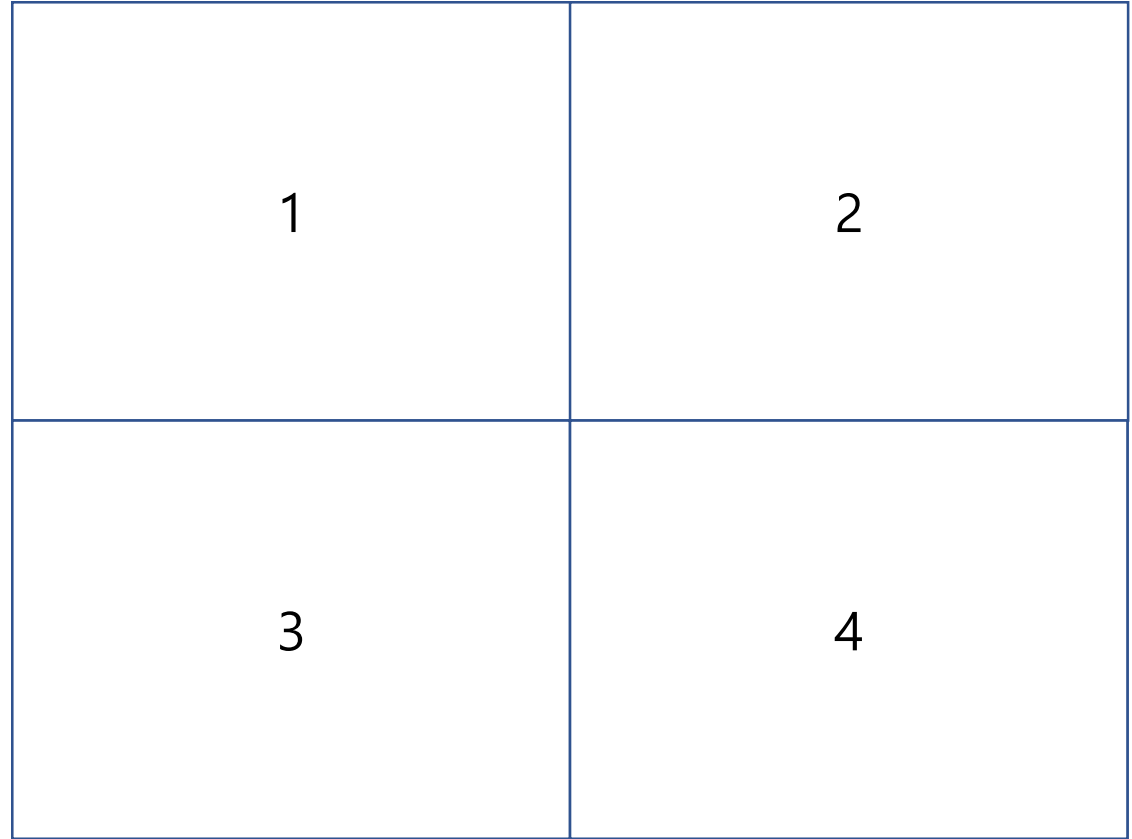
```
plt.subplot(222)
```

```
plt.subplot(223)
```

```
plt.subplot(224)
```

```
#plt.subplot(212)
```

```
plt.show()
```



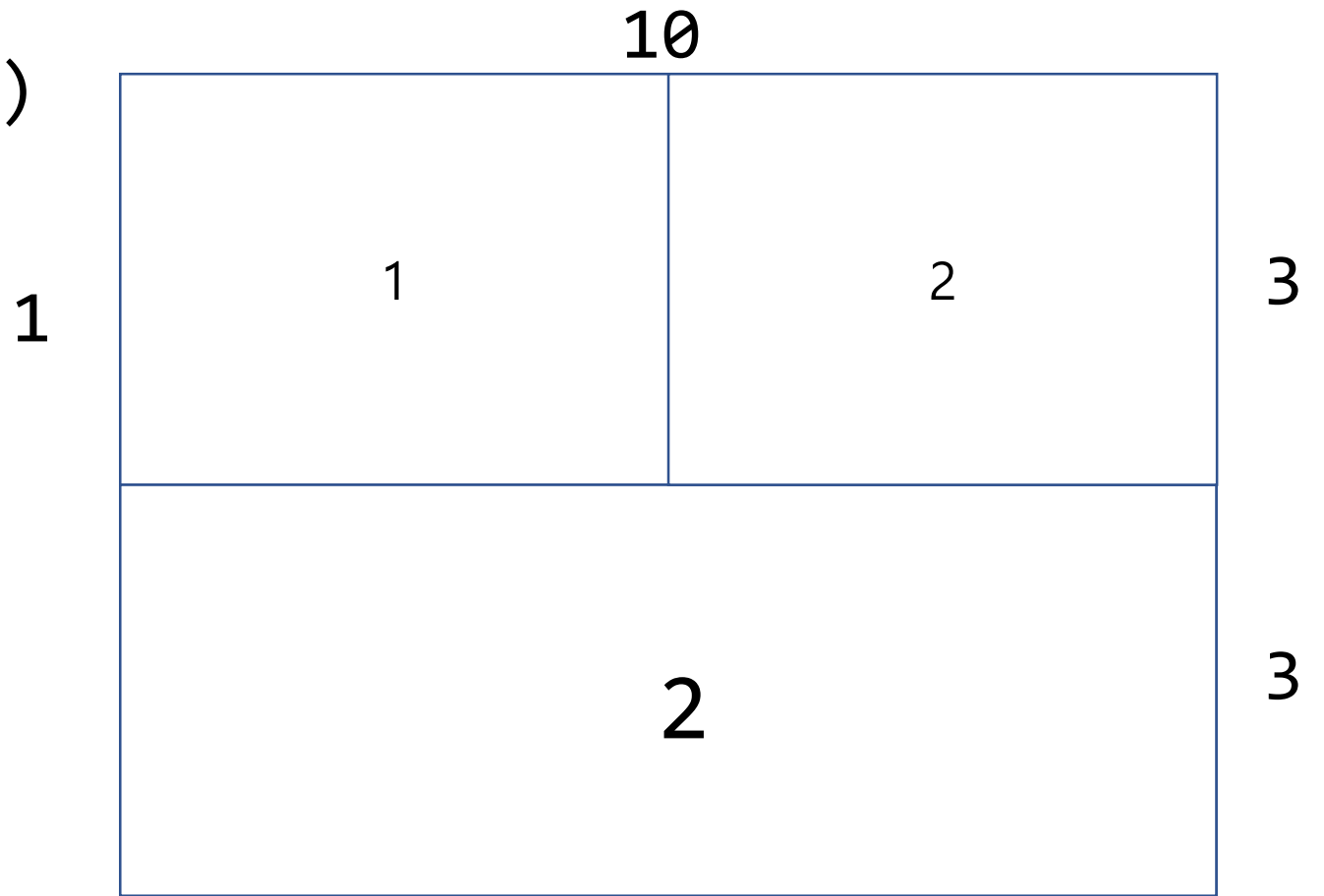
```
plt.figure(figsize=(10,6))
```

```
plt.subplot(221)
```

```
plt.subplot(222)
```

```
plt.subplot(211)
```

```
plt.show()
```



```
plt.figure(figsize=(10,6))
```

```
plt.subplot(411)
```

```
plt.subplot(423)
```

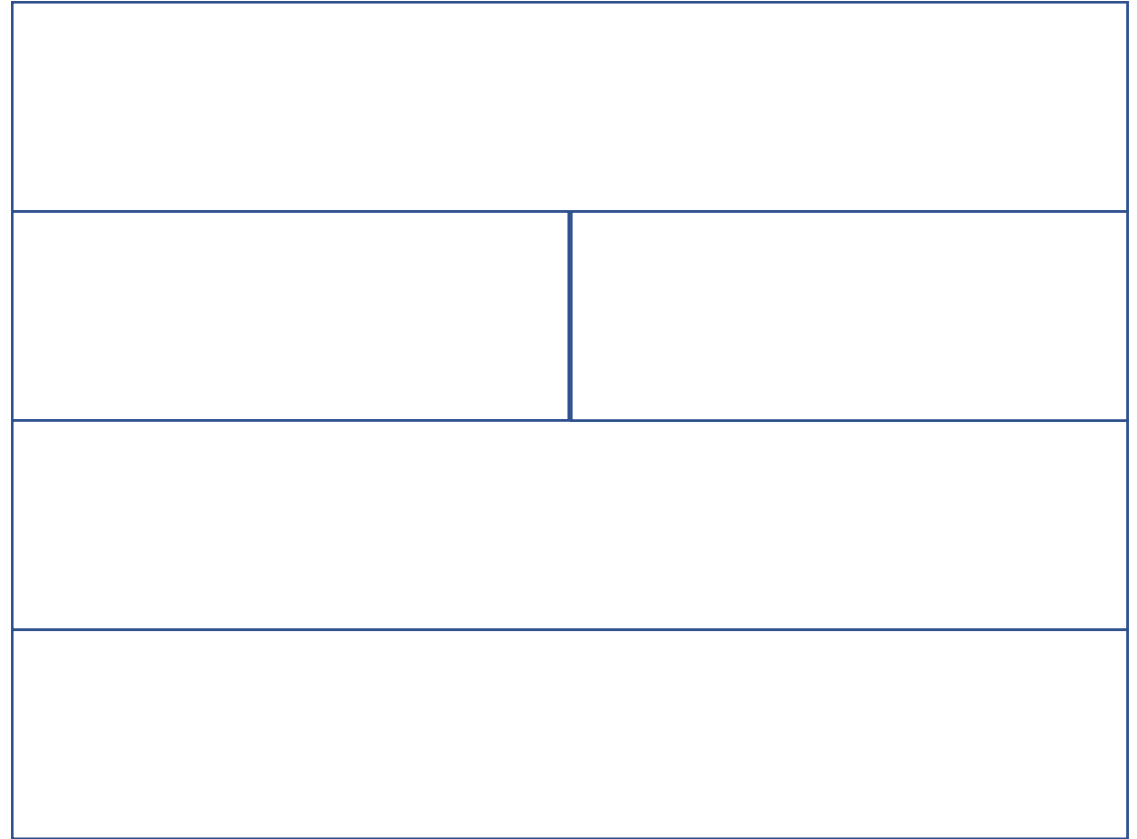
```
plt.subplot(424)
```

```
plt.subplot(413)
```

```
plt.subplot(414)
```

```
plt.show()
```

10



```
gu_name = []
```

```
for name in station_addreess:
```

```
    tmp = name.split()
```

```
    tmp_gu = [gu for gu in tmp if gu[-1] == '구'][0]
```

```
    gu_name.append(tmp_gu)
```

```
crime_anal_police['구별'] = gu_name
```

```
crime_anal_police.head()
```

```
name = '대한민국 서울특별시 중구 을지로동 수표로 27'
```

```
tmp = ['대한민국', '서울특별시', '중구', '을지로동', '수표로', '27']
```

```
gu = '중구'
```

```
['중구'][0]
```

```
'중구'
```

```
gu_name = ['중구', ...]
```

```
def MinMaxScaler( data ):
    min = data.min
    max = data.max
    for i in range(len(data)):
        data[i] = (data[i]-min)/(max-min)
```

max = 3850

min = 1063

$(2366 - 1063) / (3850 - 1063)$

```

for n in pop.index:
    if pop['광역시도'][n][-3:] not in ['광역시', '특별시', '자치시']:
        if pop['시도'][n][: -1]=='고성' and pop['광역시도'][n]=='강원도':
            si_name[n] = '고성(강원)'
        elif pop['시도'][n][: -1]=='고성' and pop['광역시도'][n]=='경상남도':
            si_name[n] = '고성(경남)'
        else:
            si_name[n] = pop['시도'][n][: -1]

            '장안구'
for keys, values in tmp_gu_dict.items():
    if pop['시도'][n] in values:
        if len(pop['시도'][n])==2:
            si_name[n] = keys + ' ' + pop['시도'][n]
        elif pop['시도'][n] in ['마산합포구', '마산회원구']:
            si_name[n] = keys + ' ' + pop['시도'][n][2: -1]
        else:
            si_name[n] = keys + ' ' + pop['시도'][n][: -1]
            수원 장안

```

```
for n in pop.index:
    ...
    elif pop['광역시도'][n] == '세종특별자치시':
        si_name[n] = '세종'

    else:
        if len(pop['시도'][n])==2:
            si_name[n] = pop['광역시도'][n][:2] + ' ' + pop['시도'][n]
        else:
            si_name[n] = pop['광역시도'][n][:2] + ' ' + pop['시도'][n][: -1]
```


sidonames_values

```
['서울특별시',  
'부산광역시',  
'대구광역시',  
'인천광역시',  
'광주광역시',  
'대전광역시',  
'울산광역시',  
'세종특별자치시',  
'경기도',  
'강원도',  
'충청북도',  
'충청남도',  
'전라북도',  
'전라남도',  
'경상북도',  
'경상남도',  
'제주특별자치도']
```

```
import re
```

```
(41.59)'
```

```
[0]
```

```
def get_num(tmp='102,566'):
```

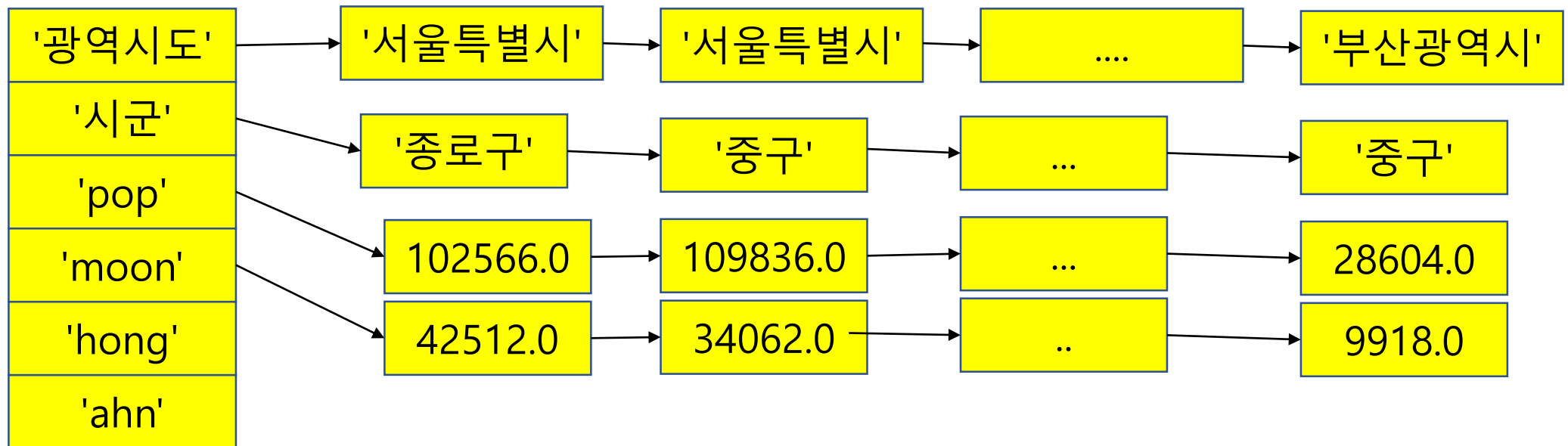
```
    return float(re.split('\(', tmp)[0].replace(
```

```
102566.0
```

```
on_result_raw = {'광역시도' : [],  
                 '시군' : [],  
                 'pop' : [],  
                 'moon' : [],  
                 'hong' : [],  
                 'ahn' : [] }
```

```
def append_data(df, sido_name='부산광역시',  
               for each in df[0].values[1:]:  
    data['광역시도'].append(sido_name)  
    data['시군'].append(each[0])  
    data['pop'].append(get_num(each[2])  
    data['moon'].append(get_num(each[3])  
    data['hong'].append(get_num(each[4])  
    data['ahn'].append(get_num(each[5]))
```

election_result_raw



```
def move_sido(name='서울특별시'):
    element = driver.find_element_by_id("cityCode")
    element.send_keys(name)
    make_xpath = "//*[@id='searchBtn']"
    wait.until(EC.element_to_be_clickable((By.XPATH,make_xpath)))
    driver.find_element_by_xpath(make_xpath).click()
```

```
from bs4 import BeautifulSoup

for each_sido in sido_names_values:
    move_sido(each_sido='부산광역시')

    html = driver.page_source
    soup = BeautifulSoup(html, 'html.parser')
    table = soup.find('table')

    df = pd.read_html(str(table))

    append_data(df, each_sido, election_result_raw)
```

```

sido_candi = election_result['광역시도']
sido_candi = [name[:2] if name[:2]
               in ['서울', '부산', '대구', '광주', '인천', '대전', '울산']
               else '' for name in sido_candi]

```

[illegible]

```
def cut_char_sigu(name):  
    return name if len(name)==2 else name[:-1]
```

```
import re
```

```
sigun_candi = ['']*len(election_result)           '안산시상록구' => '안산', '상록'  
                                                    => '안산 상록'
```

```
for n in election_result.index:  
    each = election_result['시군'][n]  
    if each[:2] in ['수원', '성남', '안양', '안산', '고양',  
                   '용인', '청주', '천안', '전주', '포항', '창원']:  
        sigun_candi[n] = re.split('시', each)[0]+' '+ cut_char_sigu(re.split('시', each)[1])  
    else:  
        sigun_candi[n] = cut_char_sigu(each)
```

```
sigun_candi
```



['서울',
 '서울',
 '서울',
 '서울',
 '서울',
 ...
 '']



['종로',
 '중구',
 '용산',
 '성동',
 '광진',
 ...
 '수원 장안']

'서울 종로'
 '서울 중구'
 '서울 용산'
 '서울 성동'
 '서울 광진'

'수원 장안'

'수원 장안'

'예천'

```
ID_candi = [sido_candi[n]+' '+sigun_candi[n] for n in range(0,len(sigun_candi))]
```

```
ID_candi = [name[1:] if name[0]==' ' else name for name in ID_candi]
```

```
ID_candi = [name[:2] if name[:2]=='세종' else name for name in ID_candi]
```

```
ID_candi
```

```
election_result[['rate_moon', 'rate_hong', 'rate_ahn']] = \
    election_result[['moon', 'hong', 'ahn']].div(election_result['pop'], axis=
election_result[['rate_moon', 'rate_hong', 'rate_ahn']] *= 100
election_result.head()
```


	광역시도	시군	pop	moon	hong	ahn	ID	ra
85	경기도	부천시	543777.0	239697.0	100544.0	128297.0	부천	44
...								
250	경기도	부천시	543777.0	239697.0	100544.0	128297.0	부천	44

```
ahn_tmp = election_result.loc[85, 'ahn']/3
hong_tmp = election_result.loc[85, 'hong']/3
moon_tmp = election_result.loc[85, 'moon']/3
pop_tmp = election_result.loc[85, 'pop']/3
```

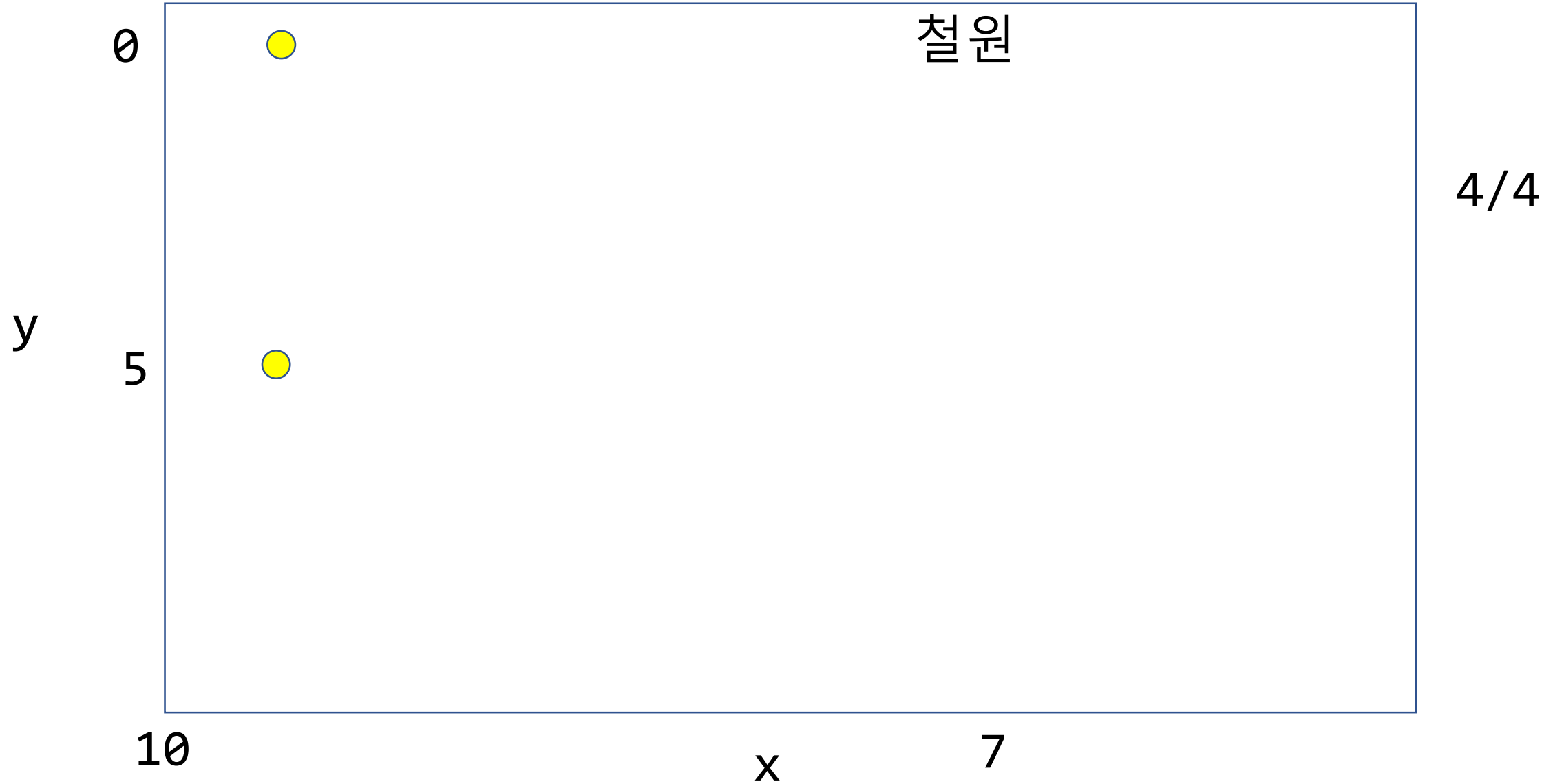
```
rate_moon_tmp = election_result.loc[85, 'rate_moon']
rate_hong_tmp = election_result.loc[85, 'rate_hong']
rate_ahn_tmp = election_result.loc[85, 'rate_ahn']
```

```
election_result.loc[250] = [ahn_tmp, hong_tmp, moon_tmp, pop_tmp,
                             '경기도', '부천시', '부천 소사',
                             rate_moon_tmp, rate_hong_tmp, rate_ahn_tmp]
election_result.loc[251] = [ahn_tmp, hong_tmp, moon_tmp, pop_tmp,
                             '경기도', '부천시', '부천 오정',
                             rate_moon_tmp, rate_hong_tmp, rate_ahn_tmp]
election_result.loc[252] = [ahn_tmp, hong_tmp, moon_tmp, pop_tmp,
                             '경기도', '부천시', '부천 원미',
```

plt.plot()

0,7,철원

1/4



```
def drawKorea(targetData, blockedMap, cmapname):
    gamma = 0.75

    whitelabelmin = 20.

    datalabel = targetData

    tmp_max = max([ np.abs(min(blockedMap[targetData])),
                    np.abs(max(blockedMap[targetData])) ])
    vmin, vmax = -tmp_max, tmp_max

    mapdata = blockedMap.pivot_table(index='y', columns='x', values=targetData)
    masked_mapdata = np.ma.masked_where(np.isnan(mapdata), mapdata)

    plt.figure(figsize=(9, 11))
    plt.pcolor(masked_mapdata, vmin=vmin, vmax=vmax, cmap=cmapname,
               edgecolor='#aaaaaa', linewidth=0.5)

drawKorea('moon_vs_hong', final_elect_data, 'RdBu')
```

2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

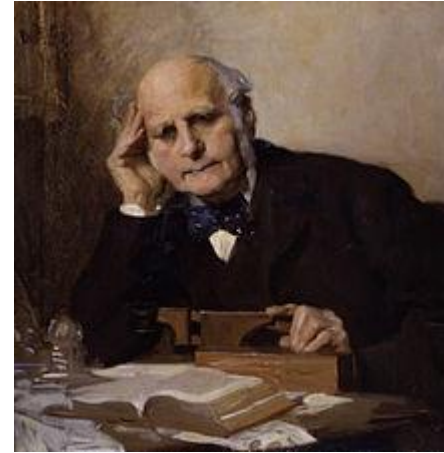
2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

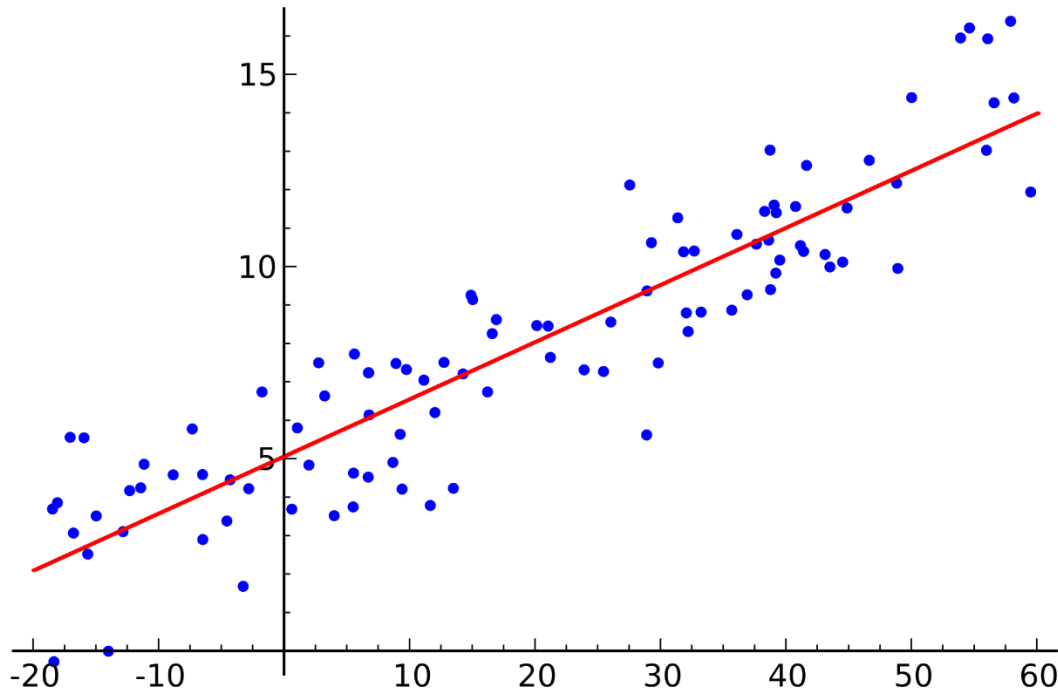
Regression - 회귀

"Regression toward the mean"



Sir Francis Galton
(1822 ~ 1911)

Linear Regression - 선형 회귀

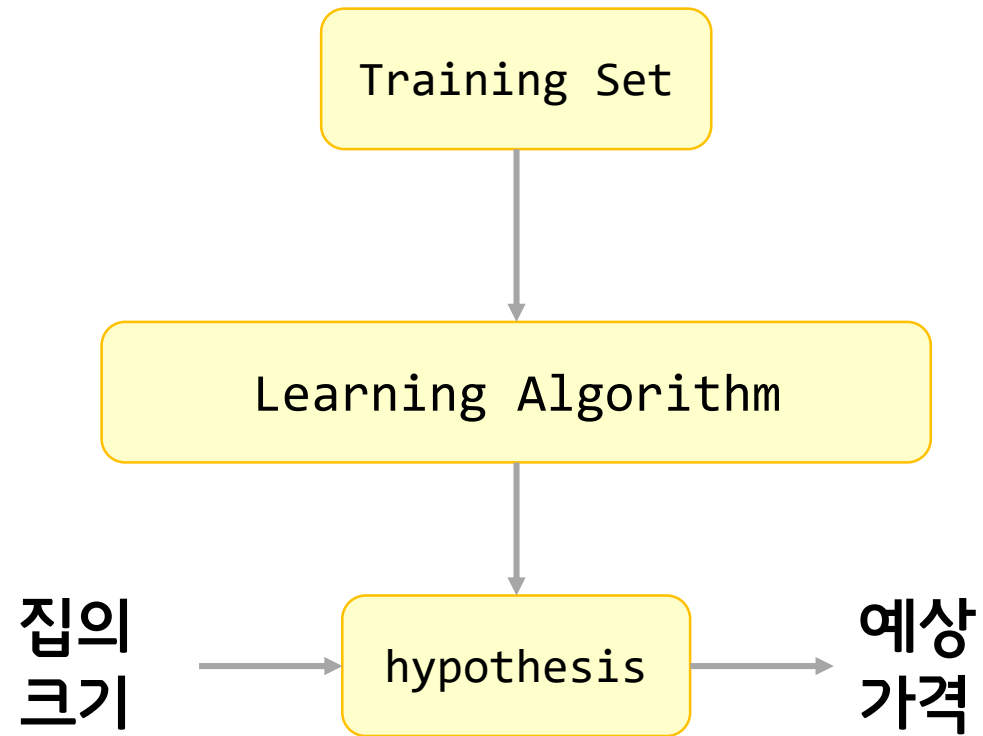


$$y = ax + b$$

$$y = wx + b$$

https://en.wikipedia.org/wiki/Linear_regression

Cost function

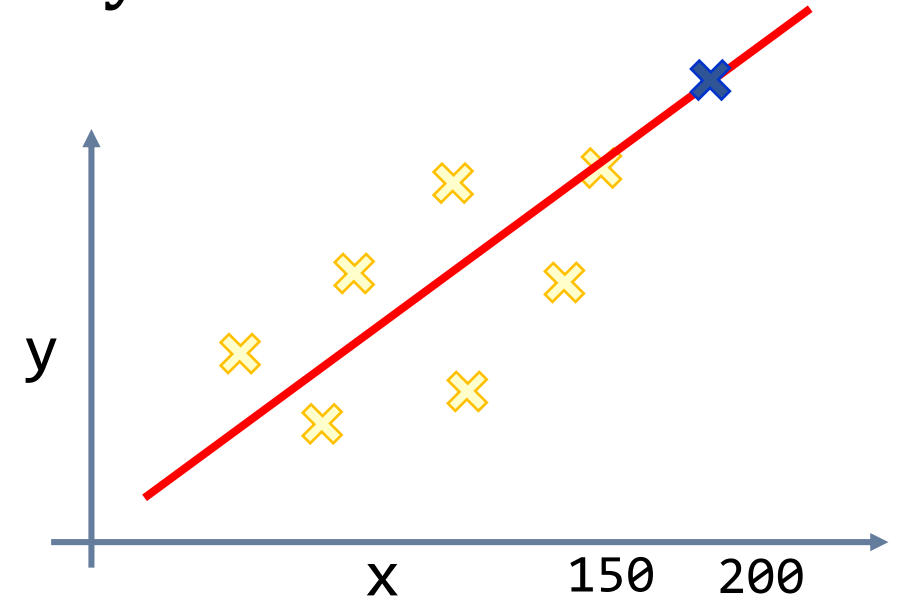


2.1 MSE(Mean Squared Error)

hypothesis ?

$$\hat{y} = wx + b$$

$$\frac{dy}{dx}$$



```
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
import numpy as np
import matplotlib.pyplot as plt

X=np.array([1,2,3,4,5])
Y=np.array([1,2,3,4,5])

model=Sequential()
model.add(Dense(1, input_dim=1, activation='linear'))
sgd=optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd ,loss='mse',metrics=['accuracy'])
model.fit(X,Y, batch_size=1, epochs=10, shuffle=False)

plt.plot(X, Y, 'bo')
plt.plot(X, model.predict(X), 'r-')
plt.show()
```


matplotlib 를 이용한 그래프 그리기

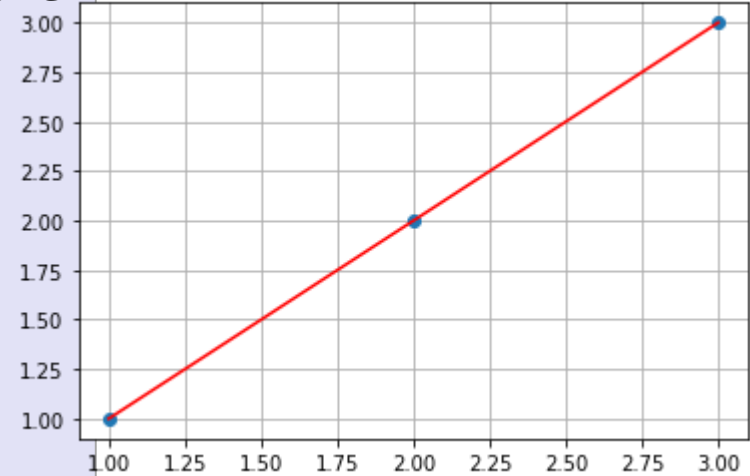
2.1 MSE(Mean Squared Error)

```
import numpy as np    # numerical computing
import matplotlib.pyplot as plt  # plotting core

x = np.array([1,2,3])
y = np.array([1,2,3])

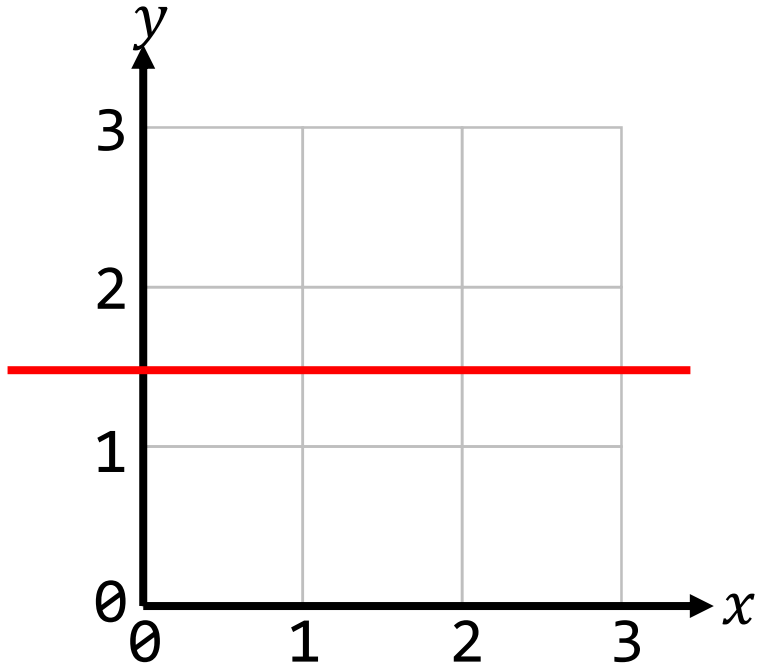
plt.plot(x,y, 'o' )
plt.plot(x,y, 'r-' )

plt.grid(True)
plt.show()
```

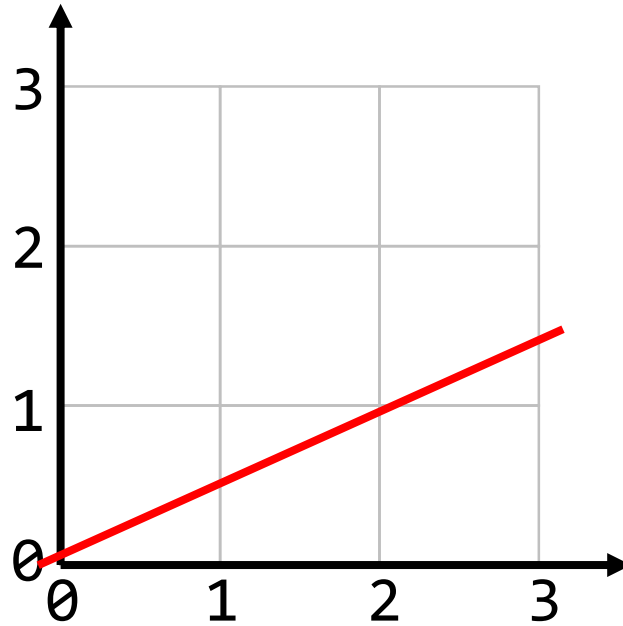


Cost function

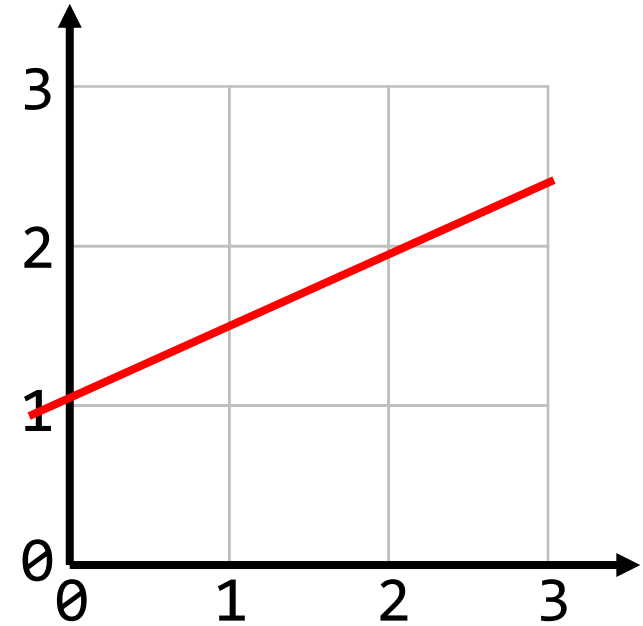
$$\hat{y} = wx + b$$



$$w = 0$$
$$b = 1.5$$



$$w = 0.5$$
$$b = 0$$



$$w = 0.5$$
$$b = 1$$

2.1 MSE(Mean Squared Error)

Hypothesis :

$$\hat{y} = wx + b$$

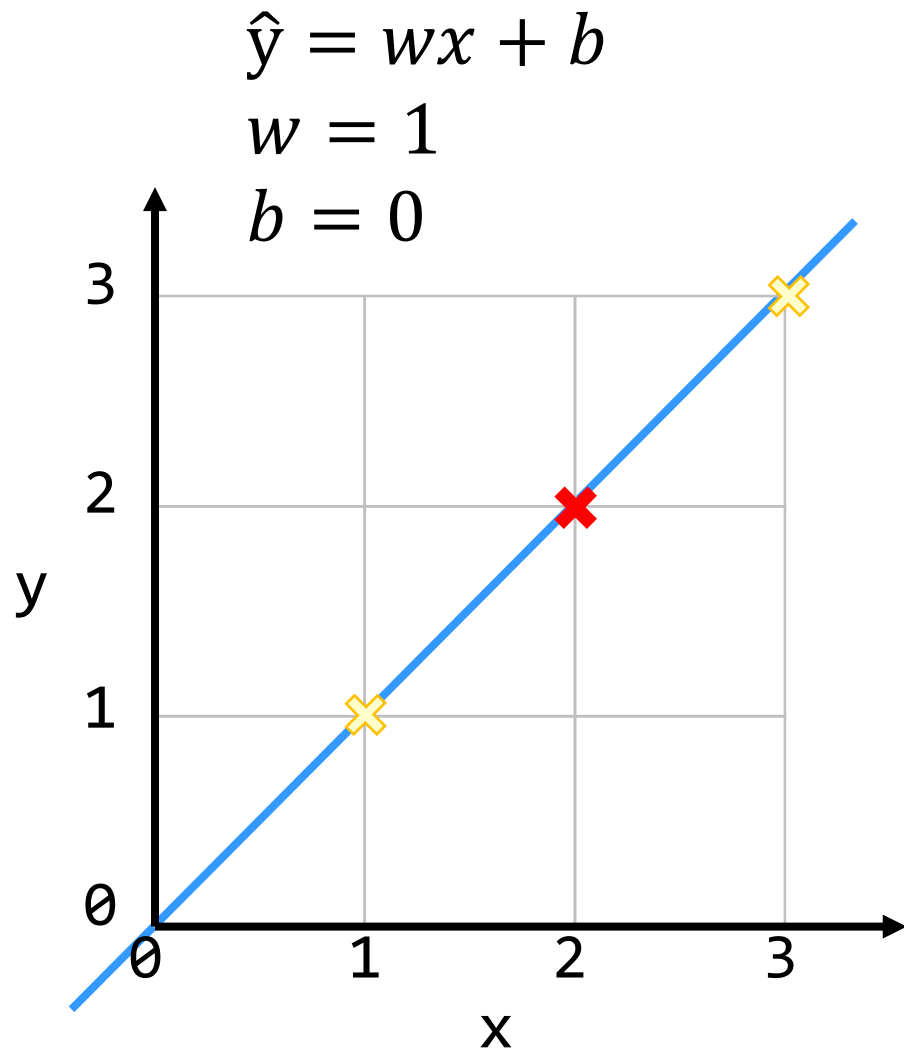
Parameters:

$$w, b$$

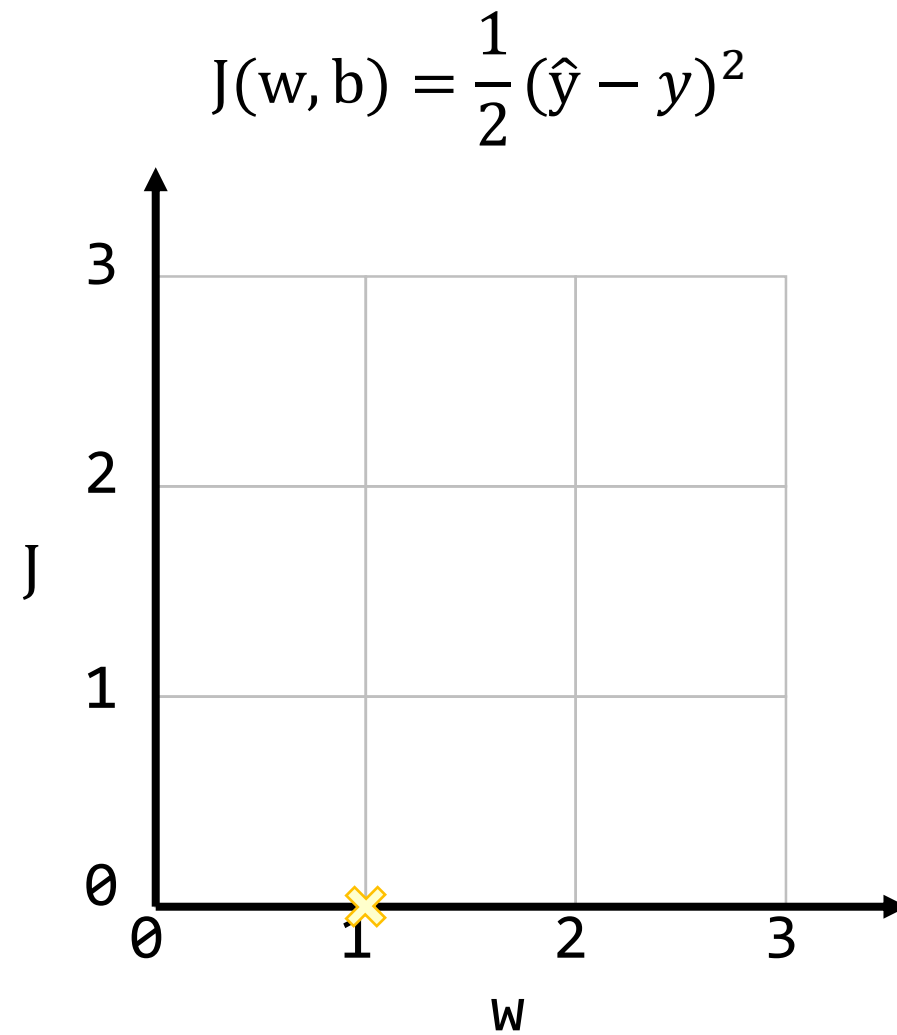
Cost Function

$$J(w, b) = \frac{1}{2} (\hat{y} - y)^2$$

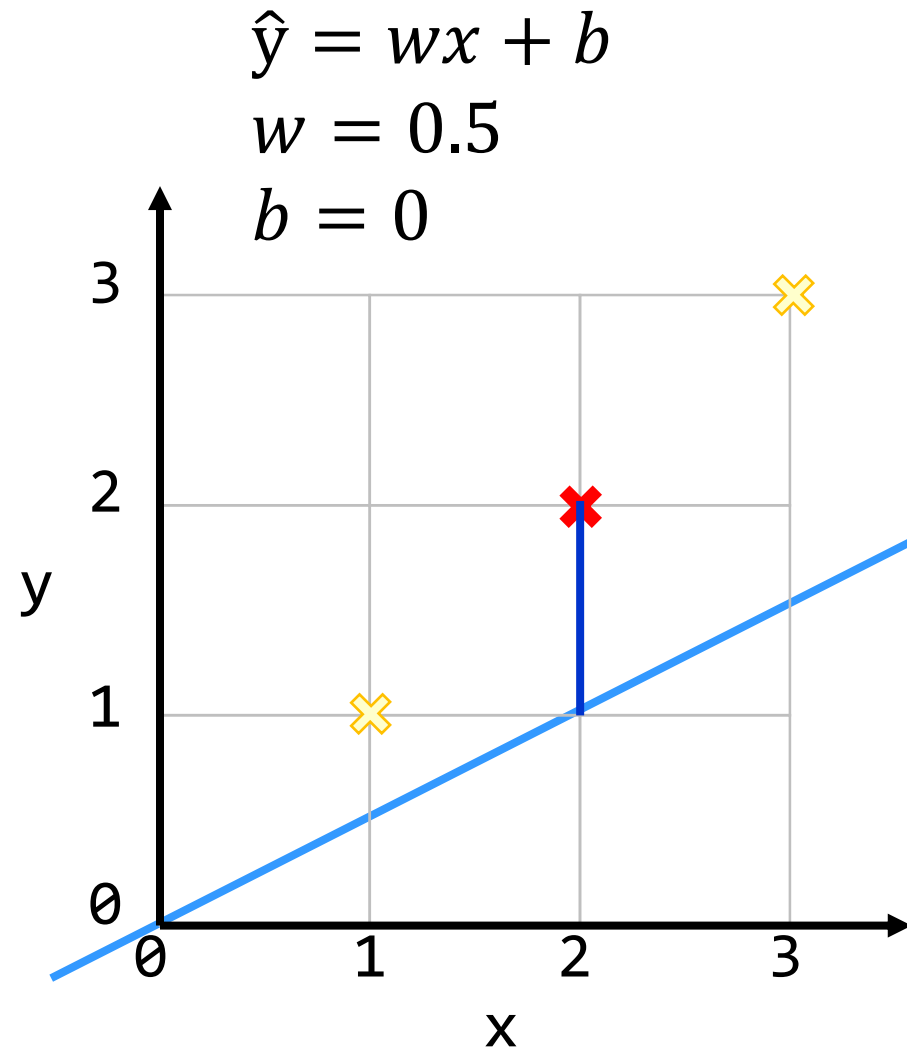
Cost function



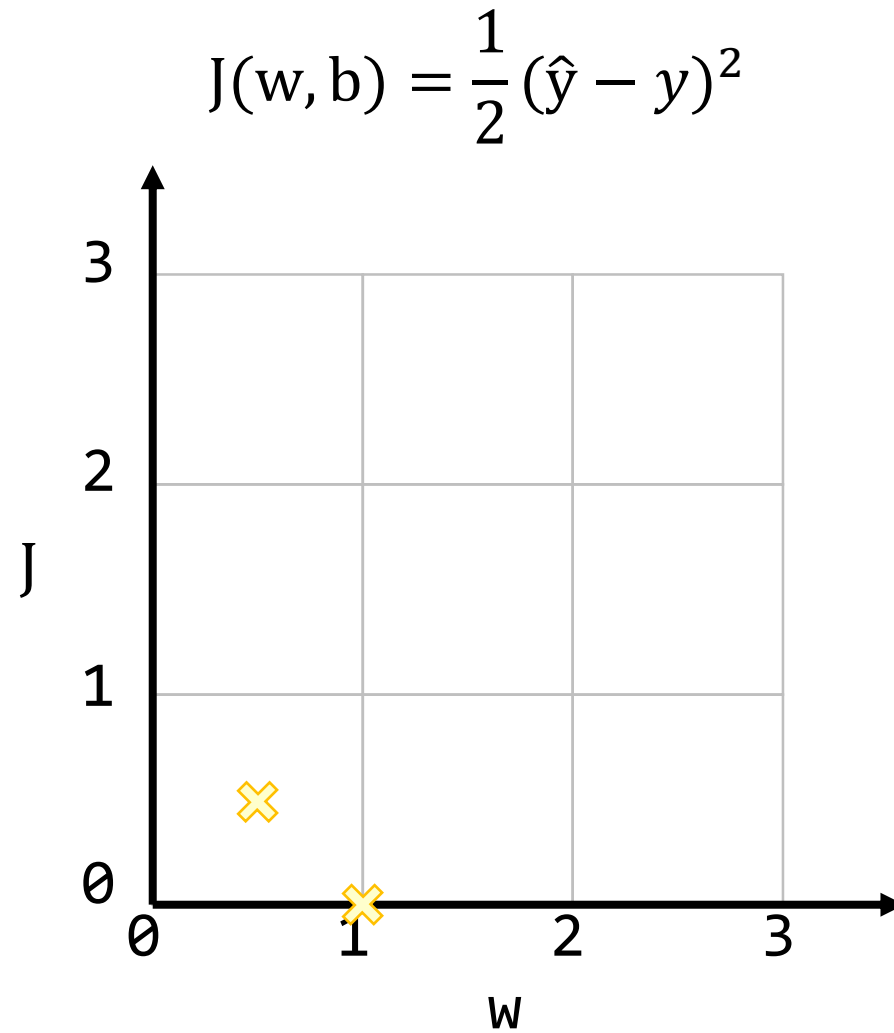
2.1 MSE(Mean Squared Error)



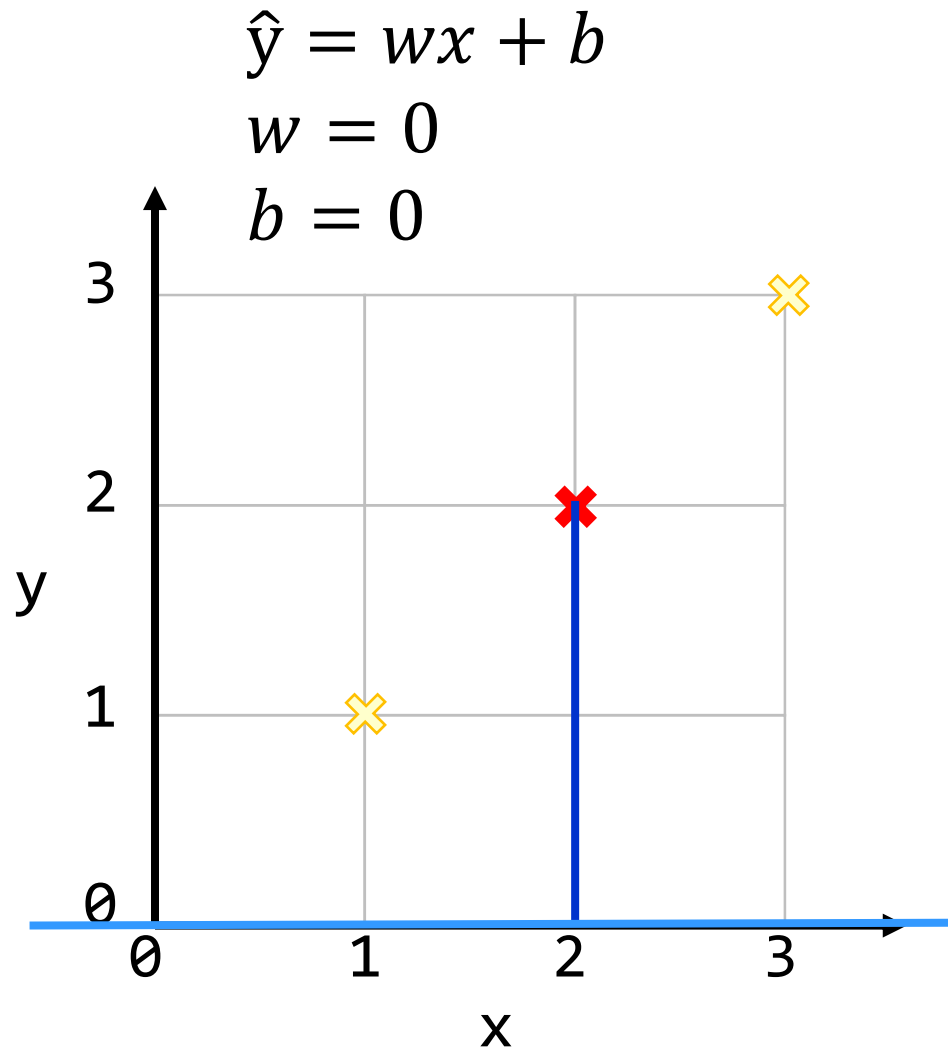
Cost function



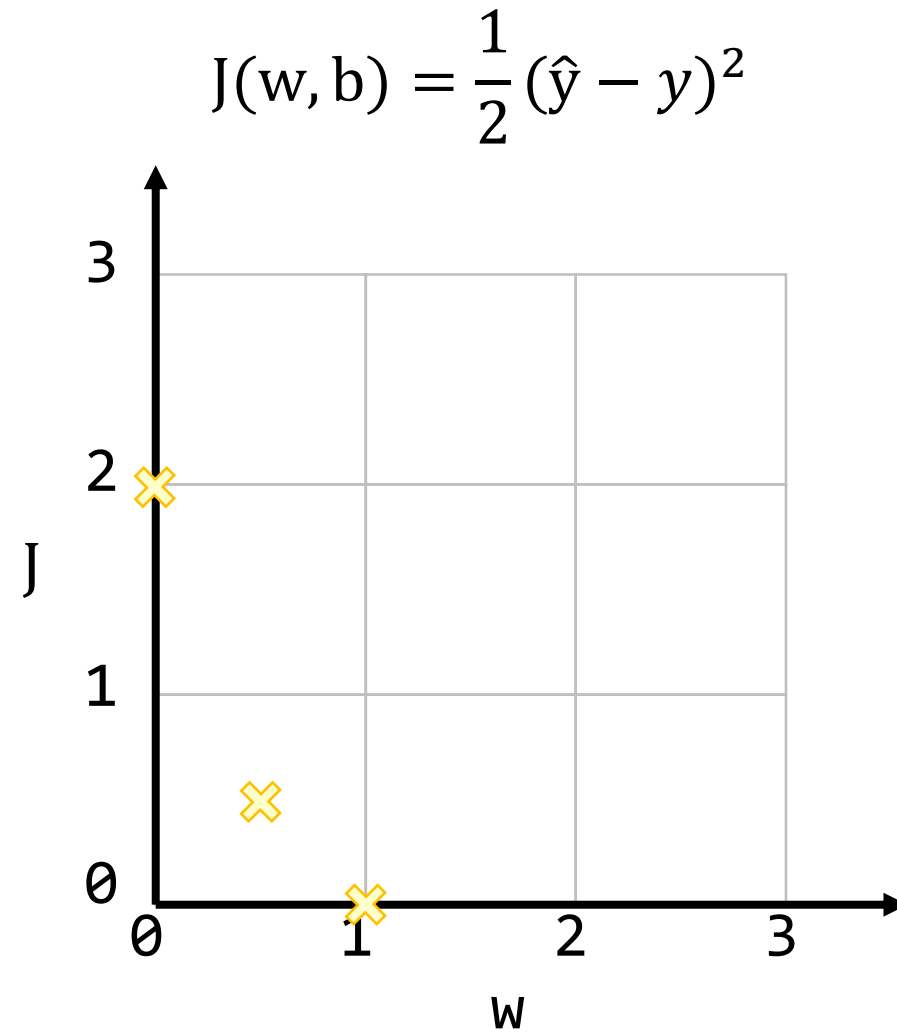
2.1 MSE(Mean Squared Error)



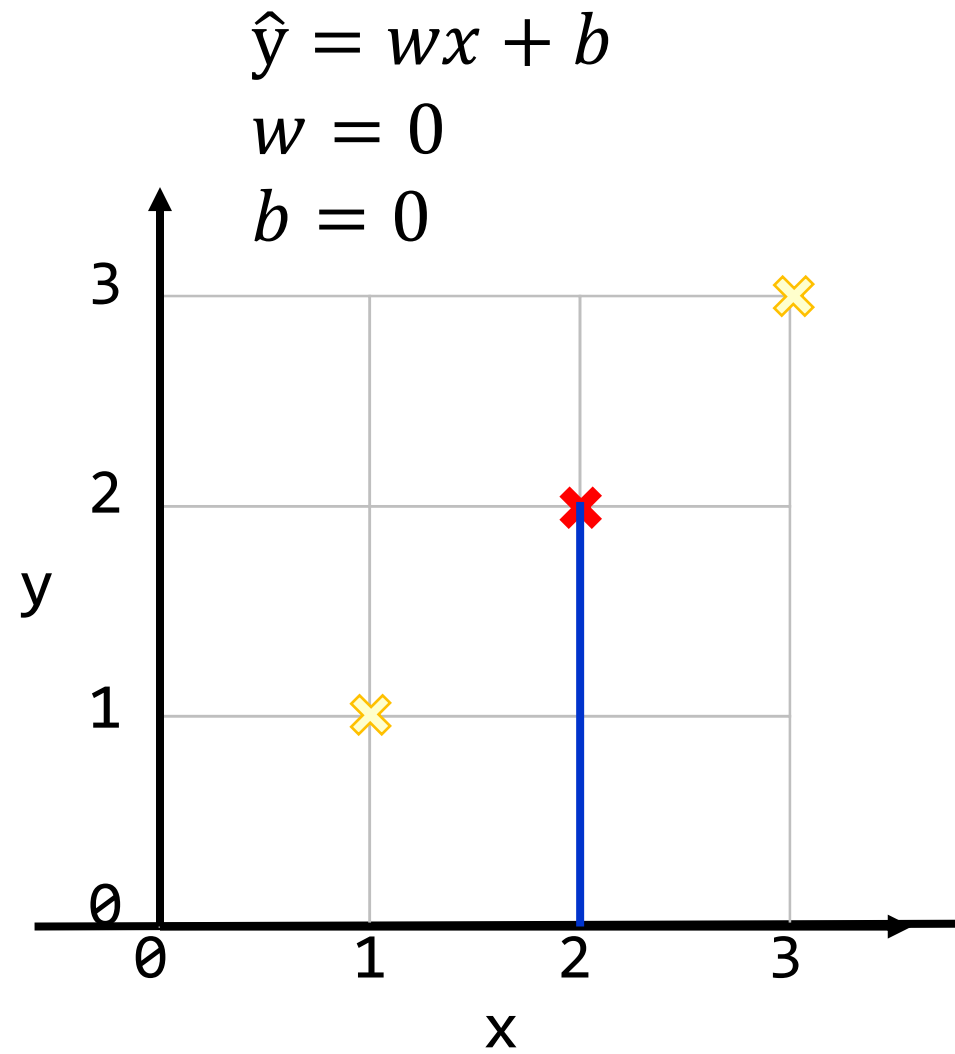
Cost function



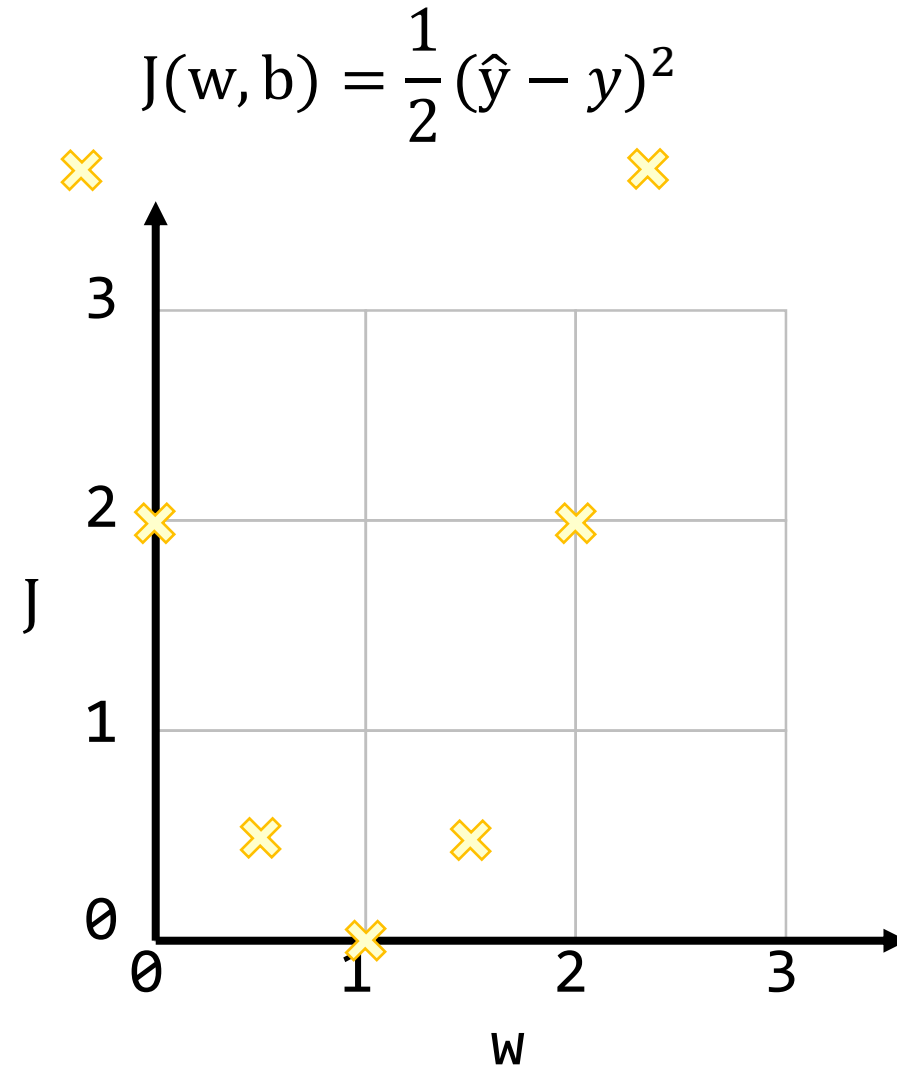
2.1 MSE(Mean Squared Error)



Cost function

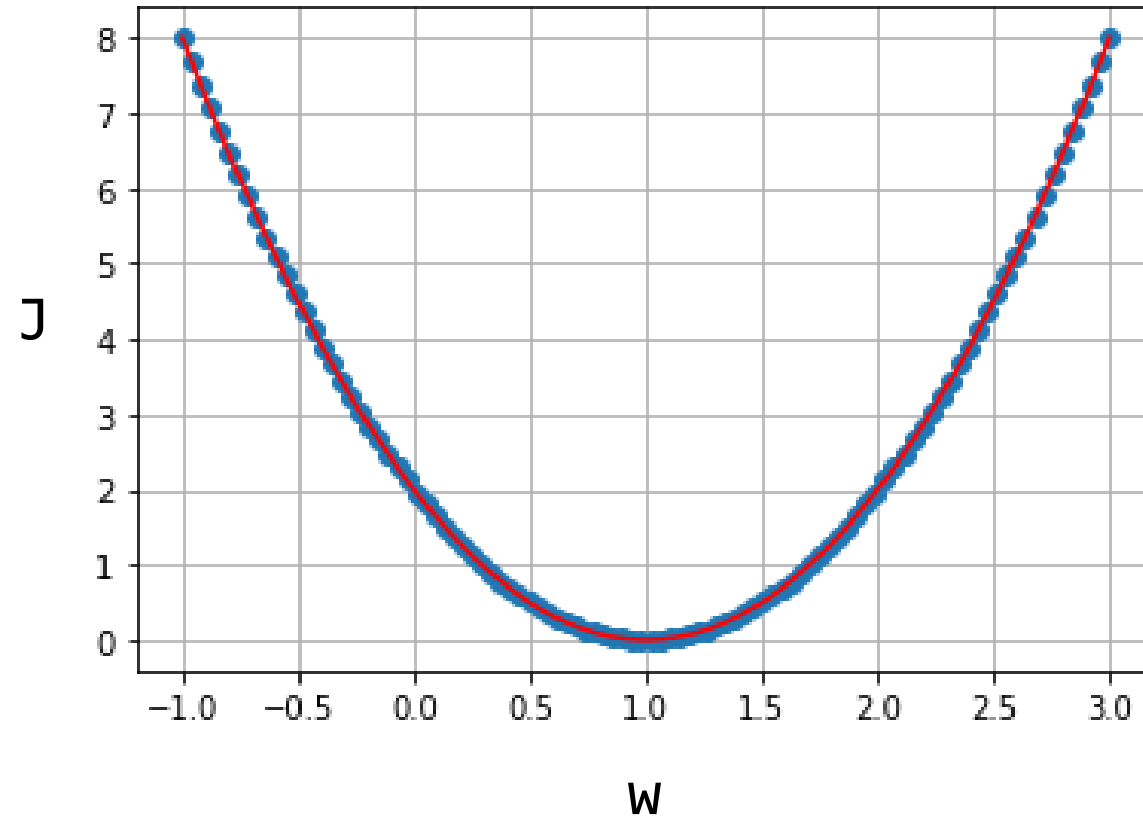


2.1 MSE(Mean Squared Error)

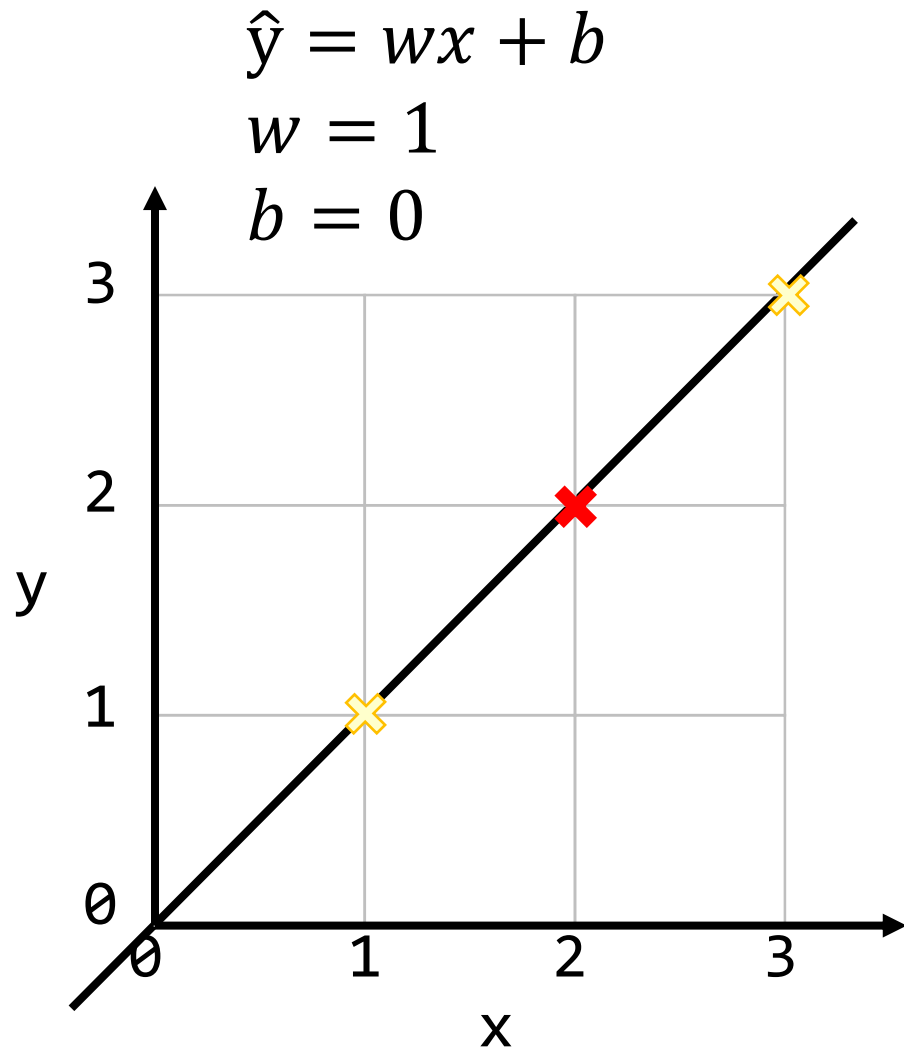


Cost function

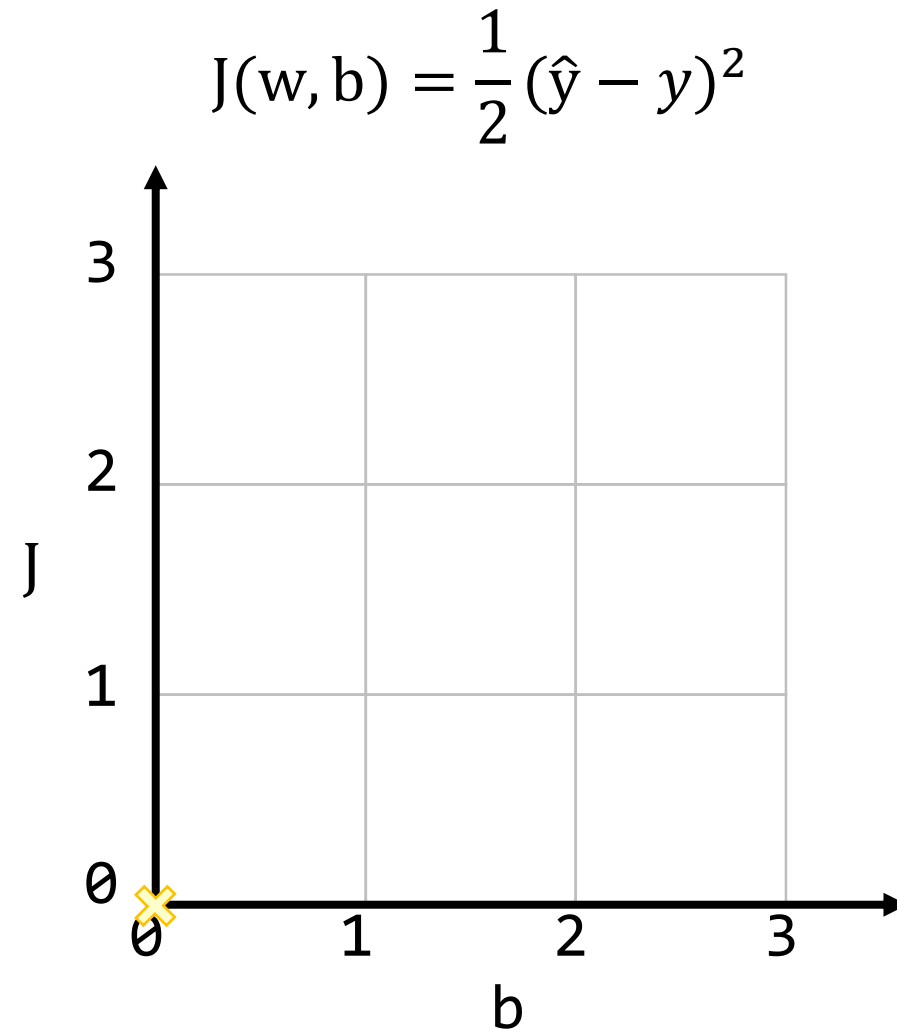
2.1 MSE(Mean Squared Error)



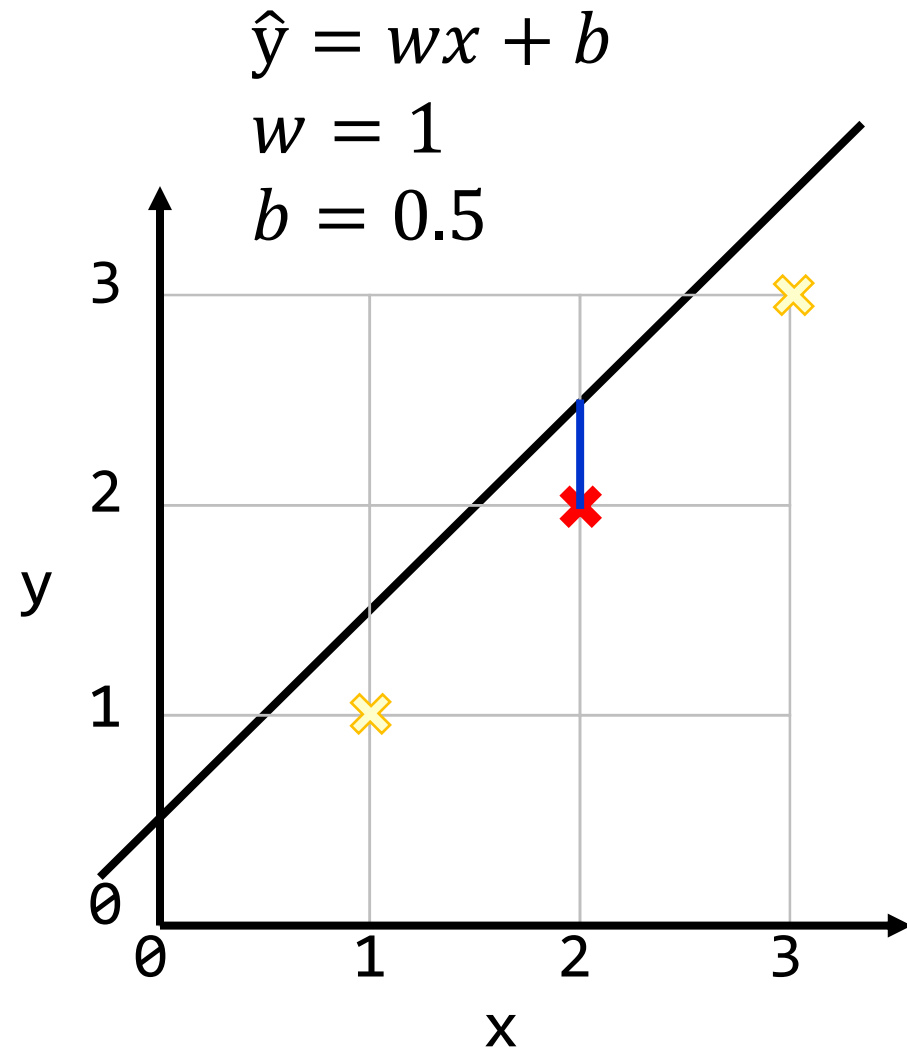
Cost function



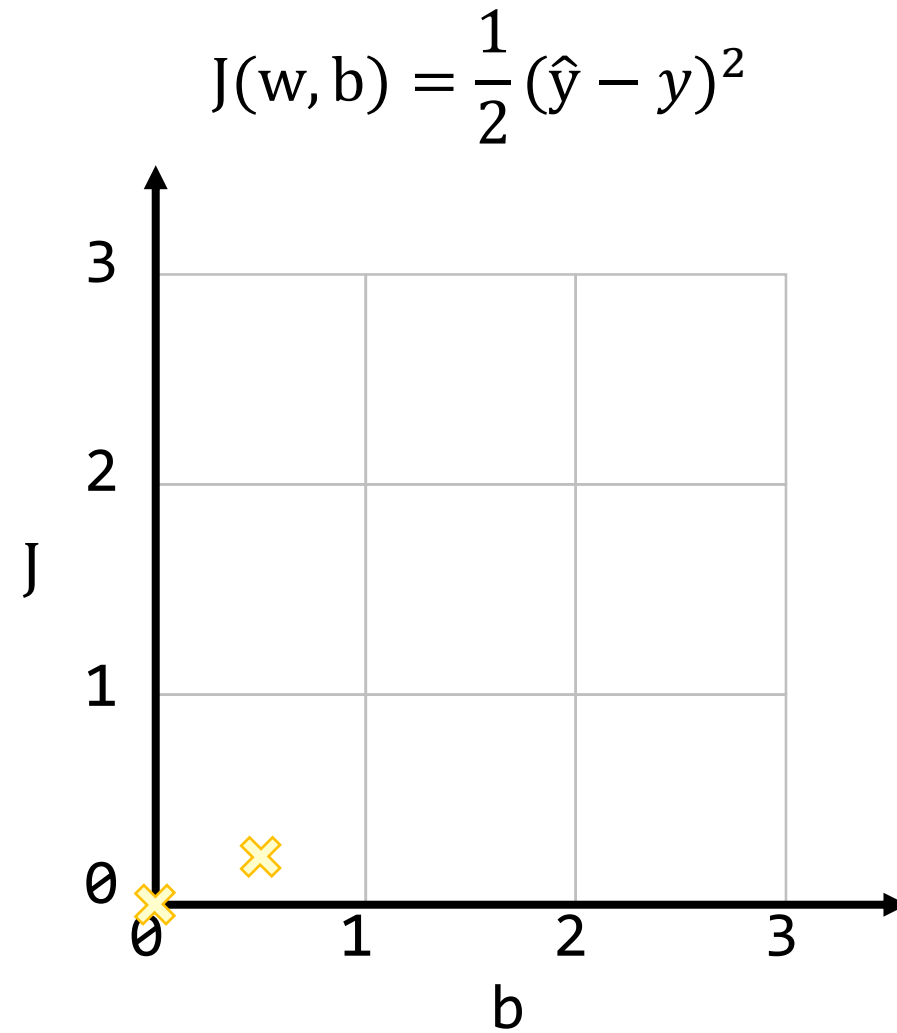
2.1 MSE(Mean Squared Error)



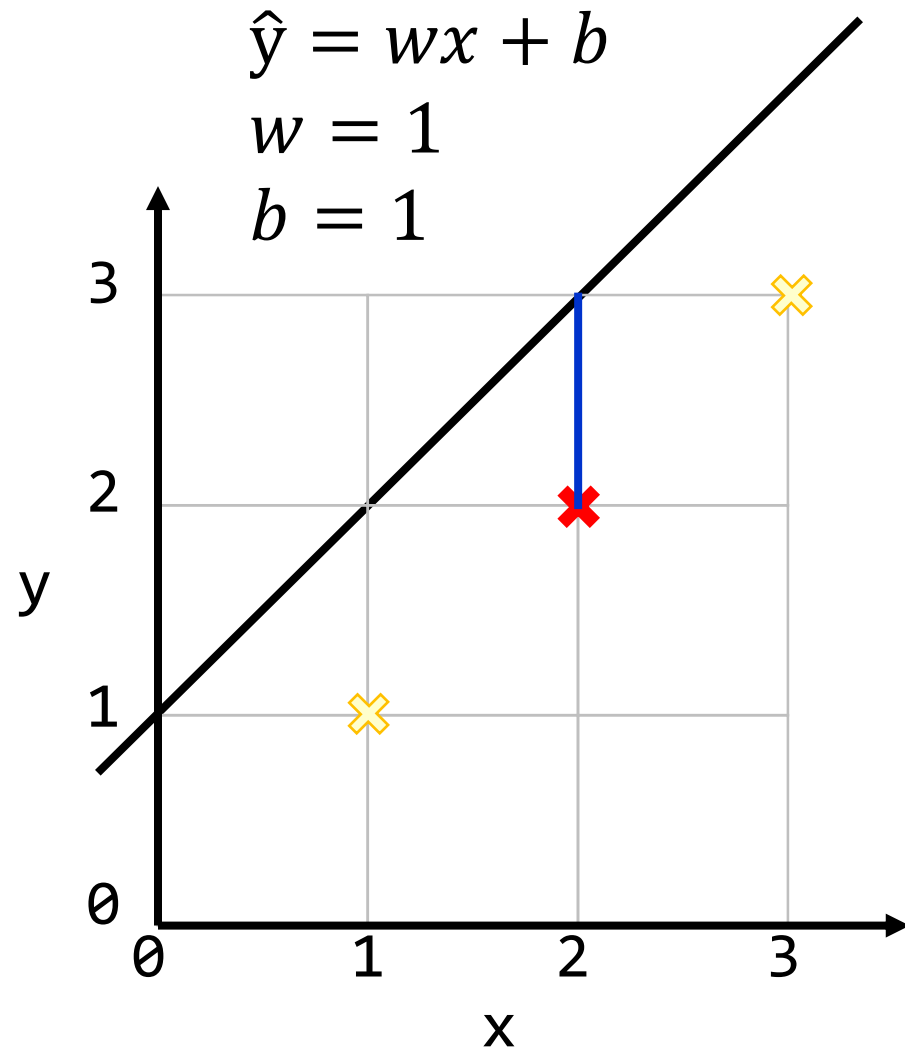
Cost function



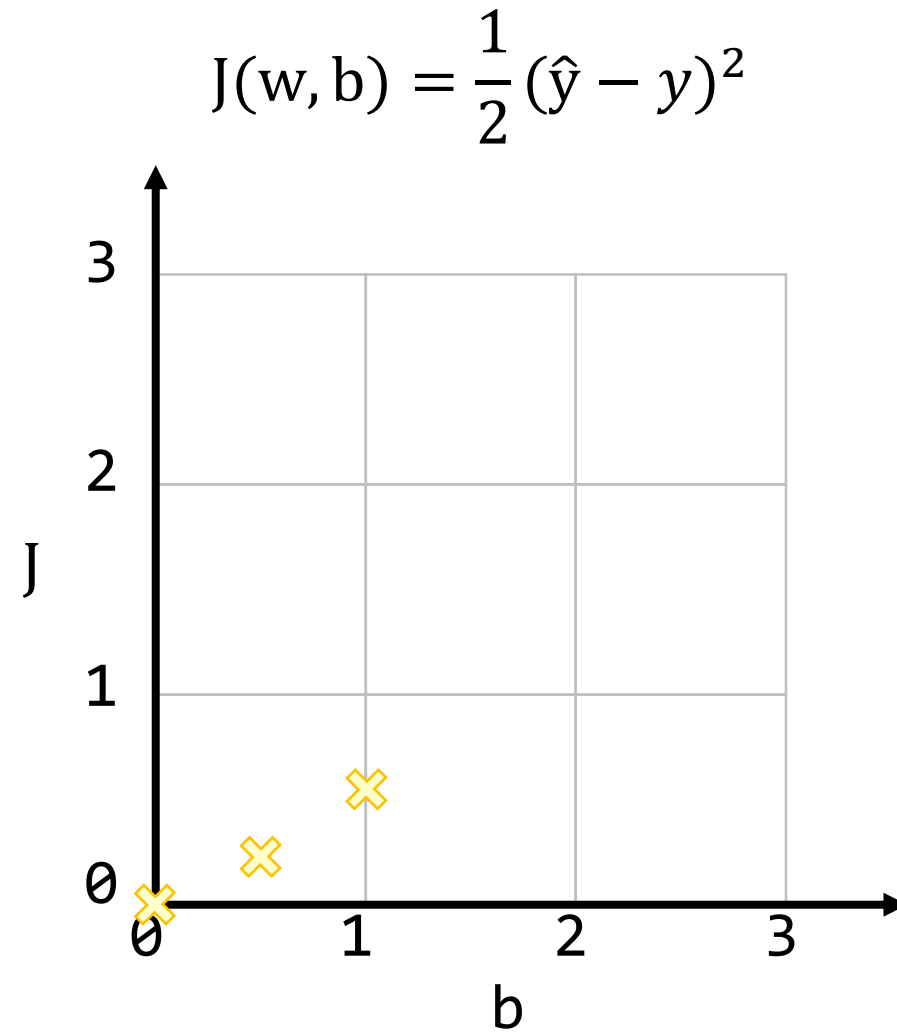
2.1 MSE(Mean Squared Error)



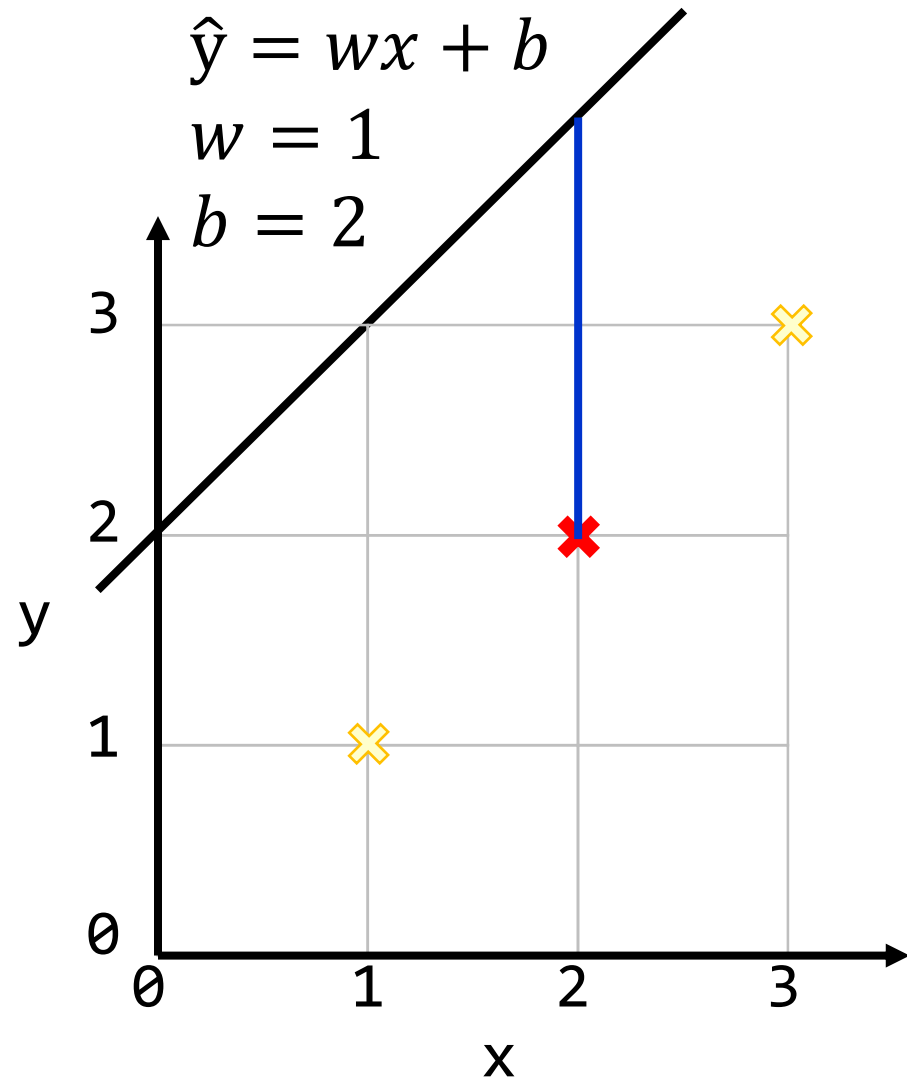
Cost function



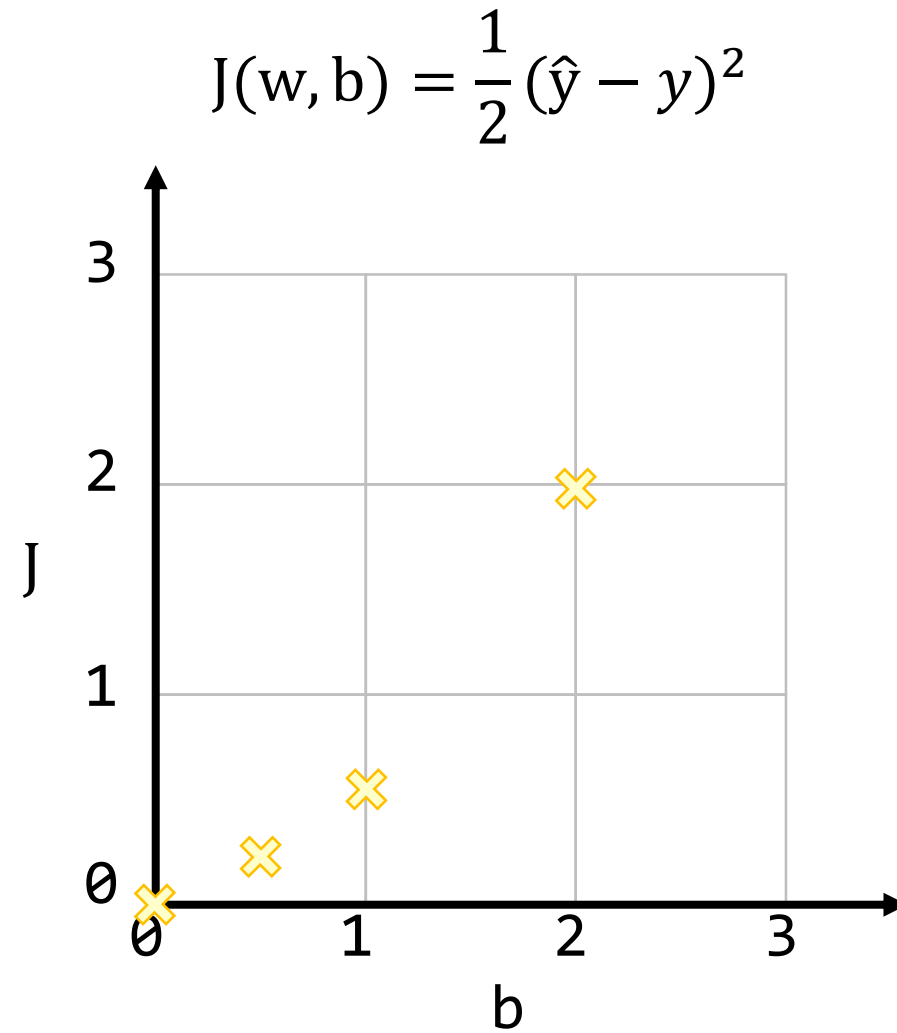
2.1 MSE(Mean Squared Error)



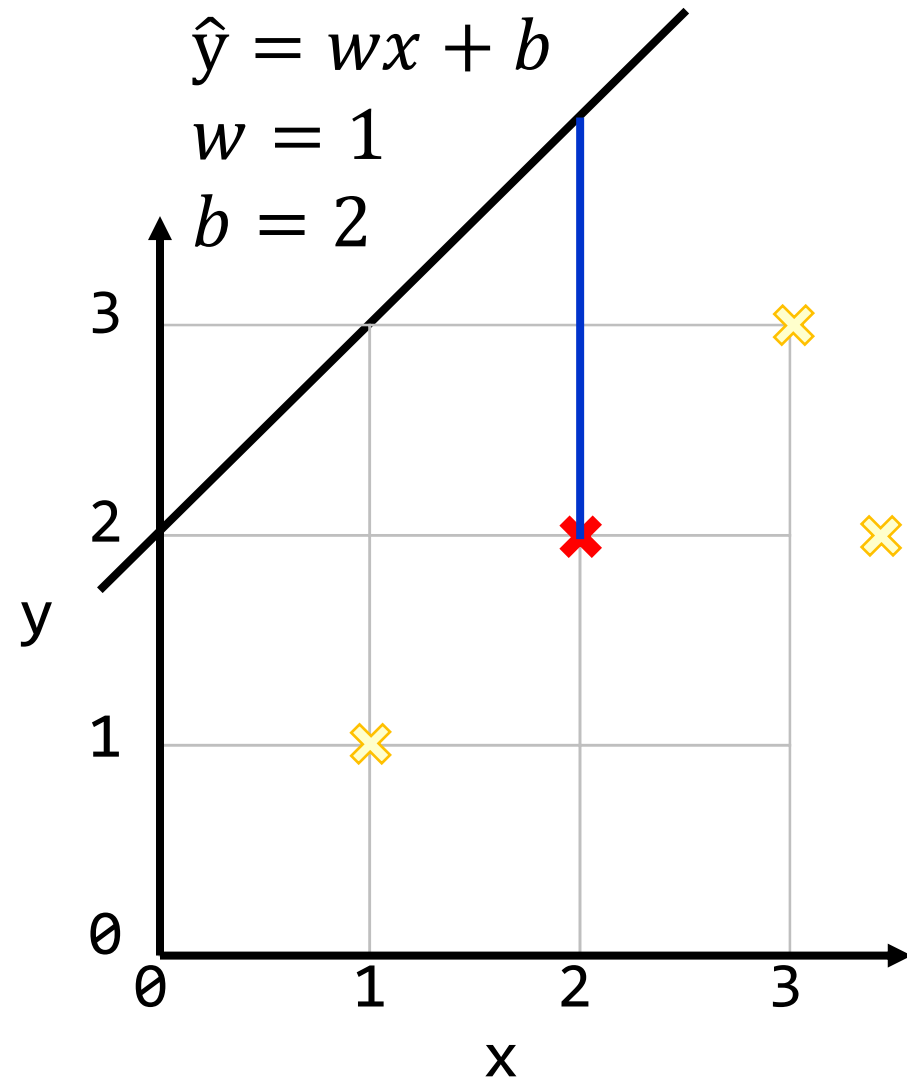
Cost function



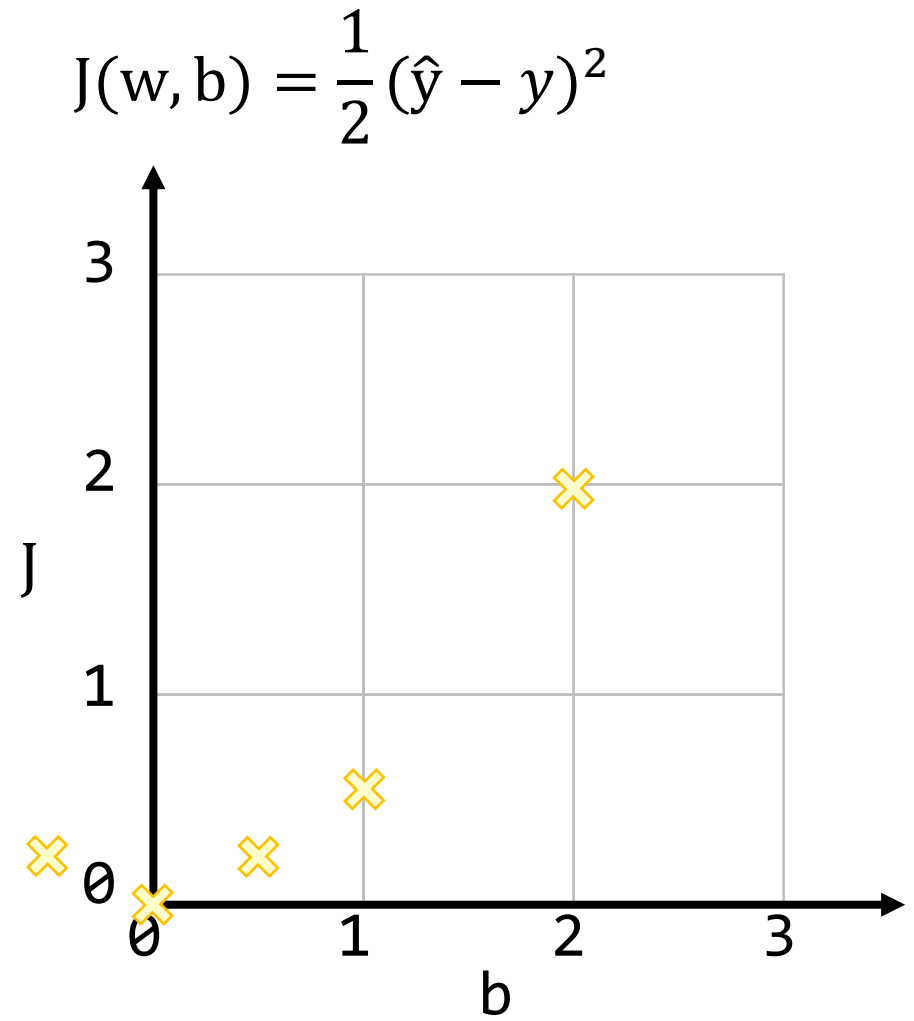
2.1 MSE (Mean Squared Error)



Cost function

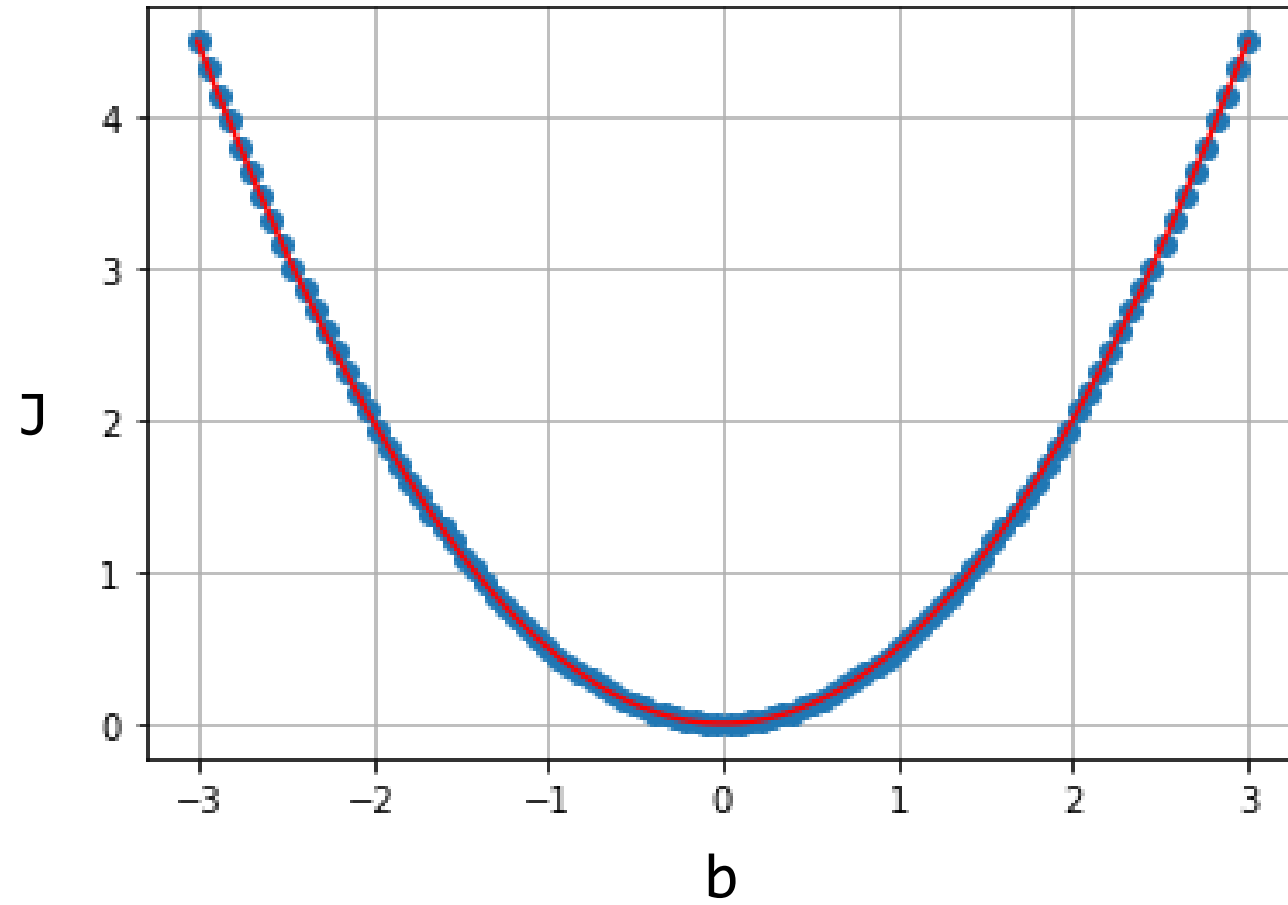


2.1 MSE (Mean Squared Error)



Cost function

2.1 MSE(Mean Squared Error)



2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

Gradient descent algorithm

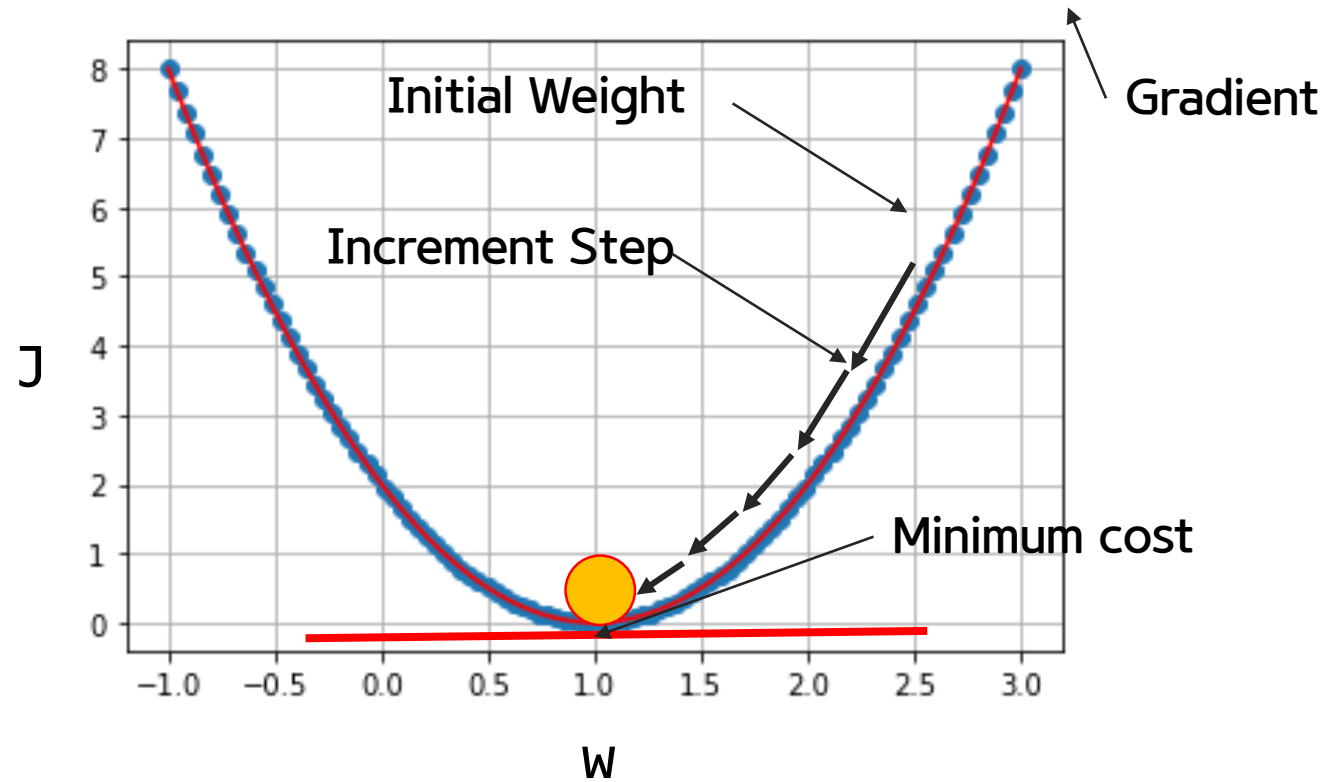
- 비용 함수 최소화
- 경사 하강은 많은 최소화 문제에 사용된다.
- 주어진 비용 함수, 비용 (W, b)에 대해 비용을 최소화하기 위해 W, b 를 찾는다.
- 일반적인 함수 : 비용 (w_1, w_2, \dots)에 적용 가능

작동 방식

- 초기 추측으로 시작
 - 0,0 (또는 다른 값)에서 시작
 - W 와 b 를 약간 변경하여 $\text{cost}(W, b)$ 의 비용을 줄이려고 노력
- 매개 변수를 변경할 때마다 가능한 가장 낮은 $\text{cost}(W, b)$ 을 감소시키는 기울기를 선택
- 반복
- 최소한의 지역으로 수렴 할 때까지 수행

Gradient descent algorithm

$$W = W - G$$



수렴까지 반복

{

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

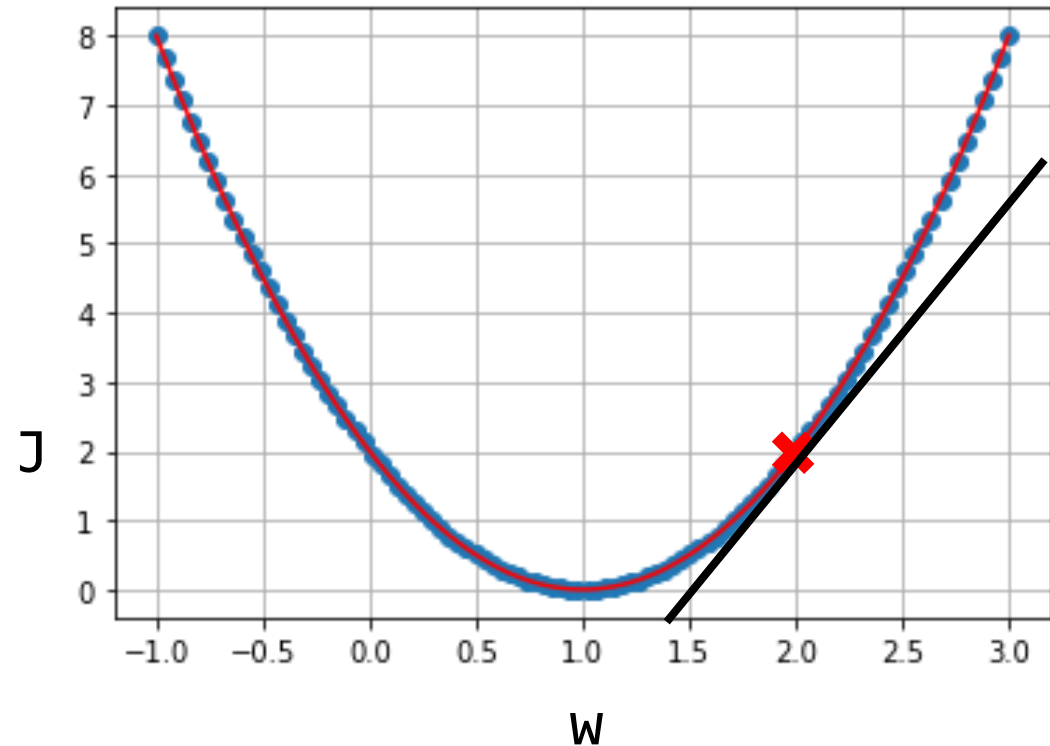
}

learning
rate

derivativ
e

$$J(w, b) = \frac{1}{2} (\hat{y} - y)^2$$

$\frac{\partial}{\partial w} J(w, b)$ 는 $\frac{1}{2} (\hat{y} - y)^2$ 을 w 에 대해서 편미분 하면 된다.



Gradient descent algorithm

2.2 SGD (Stochastic Gradient Descent)

$$= \frac{1}{2} ((wx + b) - y)^2$$

$$= \frac{1}{2} ((wx + b)^2 - 2(wx + b)y + y^2)$$

$$= \frac{1}{2} ((wx + b)^2 - 2ywx - 2by + y^2)$$

=

$$= \frac{1}{2} (w^2x^2 + 2wxb + b^2 - 2ywx - 2by + y^2)$$

$$= \frac{1}{2} (2wx^2 + 2xb - 2yx)$$

$$= x(wx + b - y)$$

$$= x(\hat{y} - y)$$

$$\hat{y} = wx + b$$

$$J(w, b) = \frac{1}{2} (\hat{y} - y)^2$$

Gradient descent algorithm

$$\frac{\partial J}{\partial \hat{y}} = \frac{1}{2} (\hat{y} - y)^2$$

$$= \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial w} = wx + b$$

$$= x$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w} = x(\hat{y} - y)$$

2.2 SGD (Stochastic Gradient Descent)

Chain Rule 사용

$$\hat{y} = wx + b$$

$$J(w,b) = \frac{1}{2} (\hat{y} - y)^2$$

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w}$$

Gradient descent algorithm

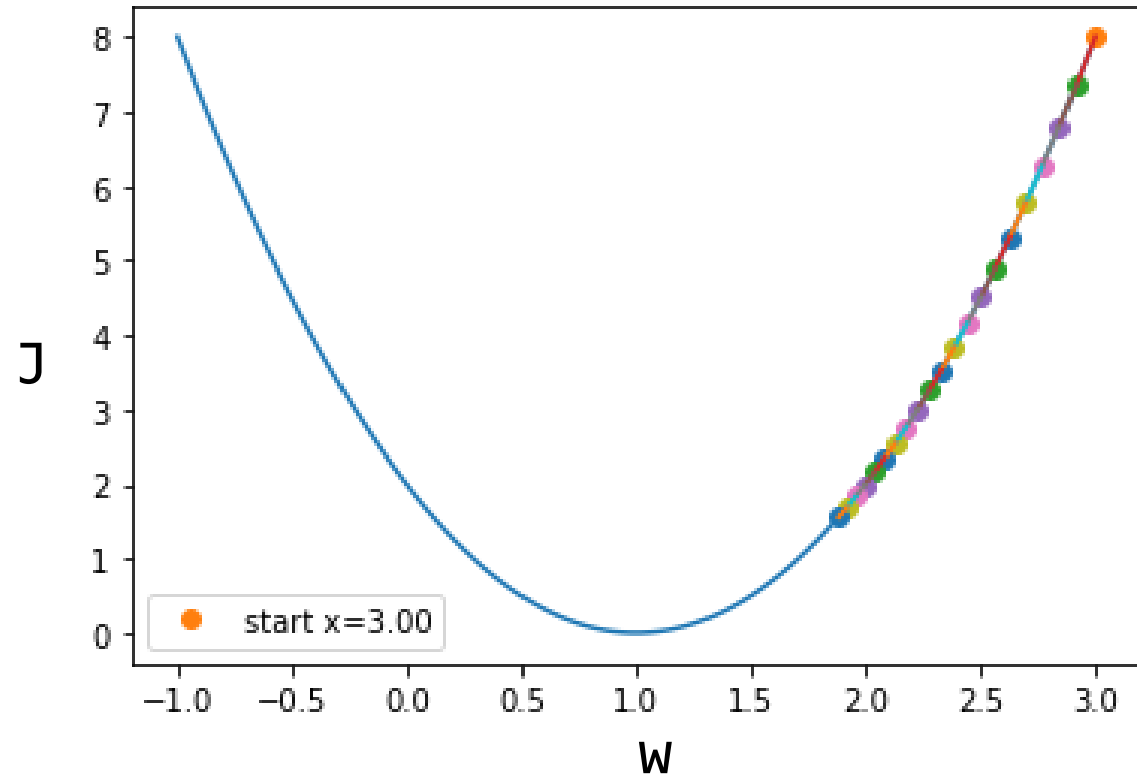
2.2 SGD (Stochastic Gradient Descent)

수렴까지 반복

{
}
}

$$w = w + \alpha * (y - \hat{y}) * x$$

α 가 0.01인 경우



Gradient descent algorithm

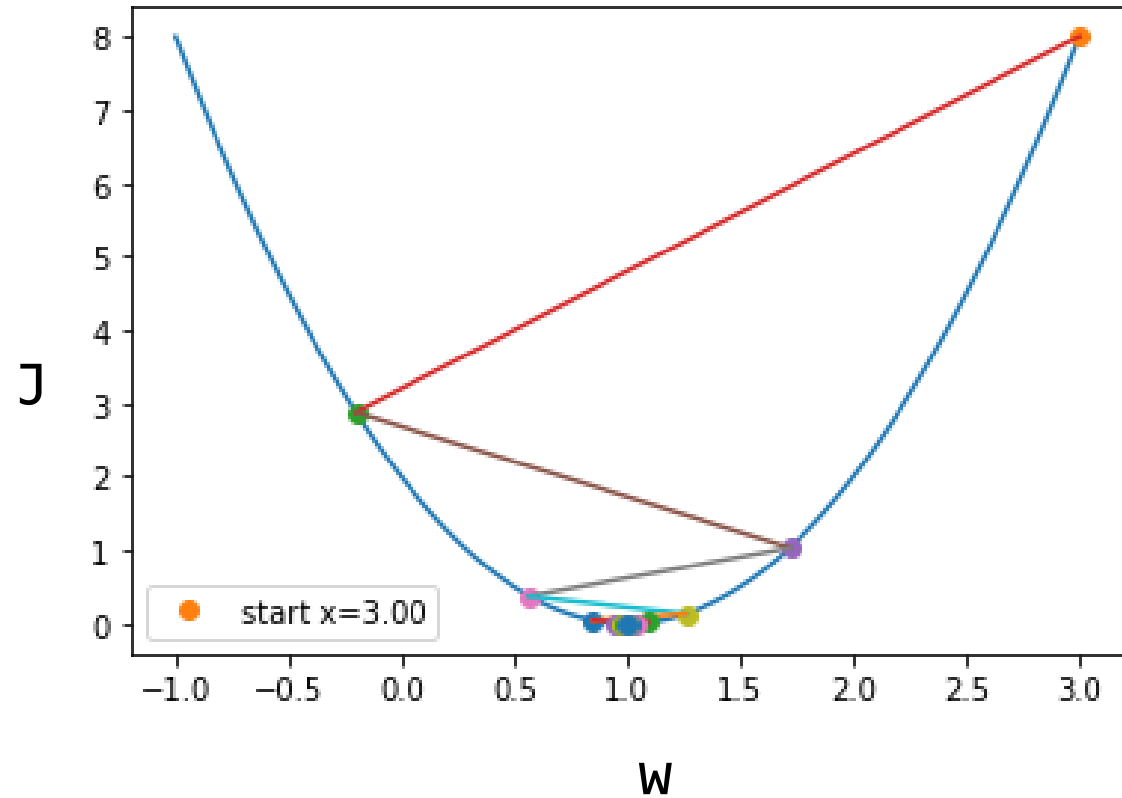
2.2 SGD (Stochastic Gradient Descent)

수렴까지 반복

{
}
}

$$w = w + \alpha * (y - \hat{y}) * x$$

α 가 0.4인 경우



Gradient descent algorithm

2.2 SGD (Stochastic Gradient Descent)

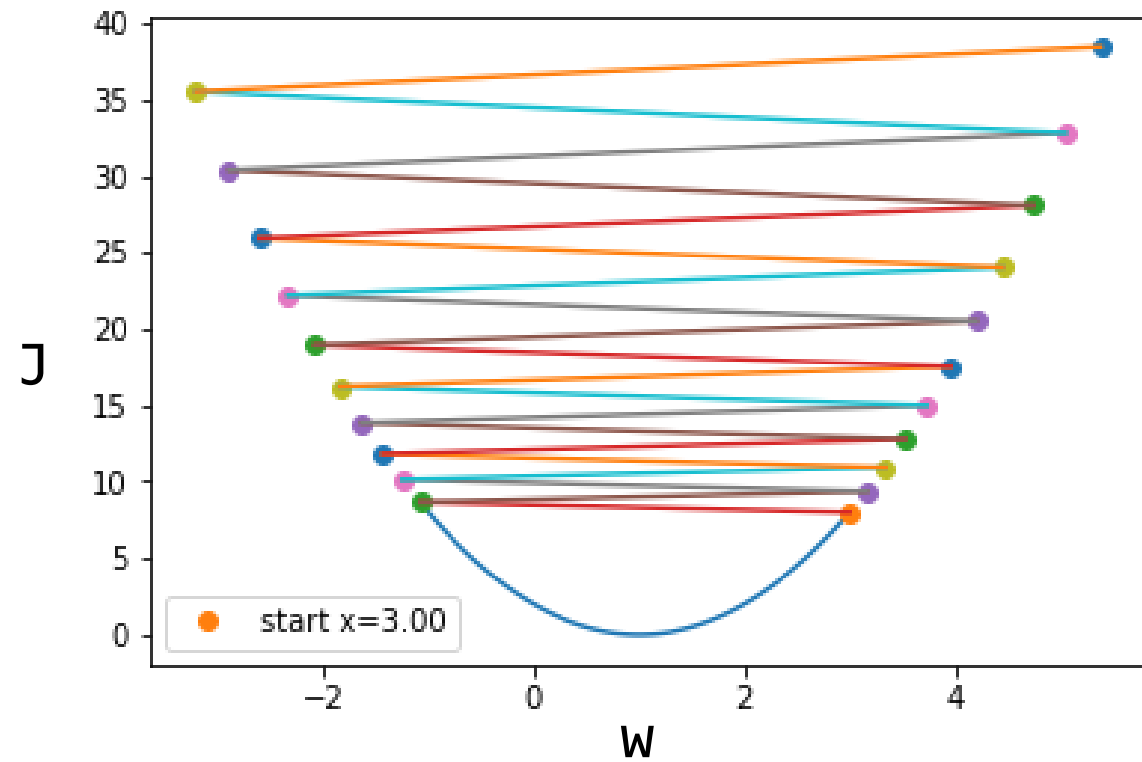
수렴까지 반복

{

$$w = w - \alpha * (\hat{y} - y) * x$$

}

α 가 0.51인 경우



Gradient descent algorithm

$$\frac{\partial J}{\partial \hat{y}} = \frac{1}{2} (\hat{y} - y)^2$$

$$= \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial w} = wx + b$$

$$= 1$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w} = (\hat{y} - y)$$

2.2 SGD (Stochastic Gradient Descent)

Chain Rule 사용

$$\hat{y} = wx + b$$

$$J(w,b) = \frac{1}{2} (\hat{y} - y)^2$$

$$\frac{\partial}{\partial b} J(w,b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial b}$$

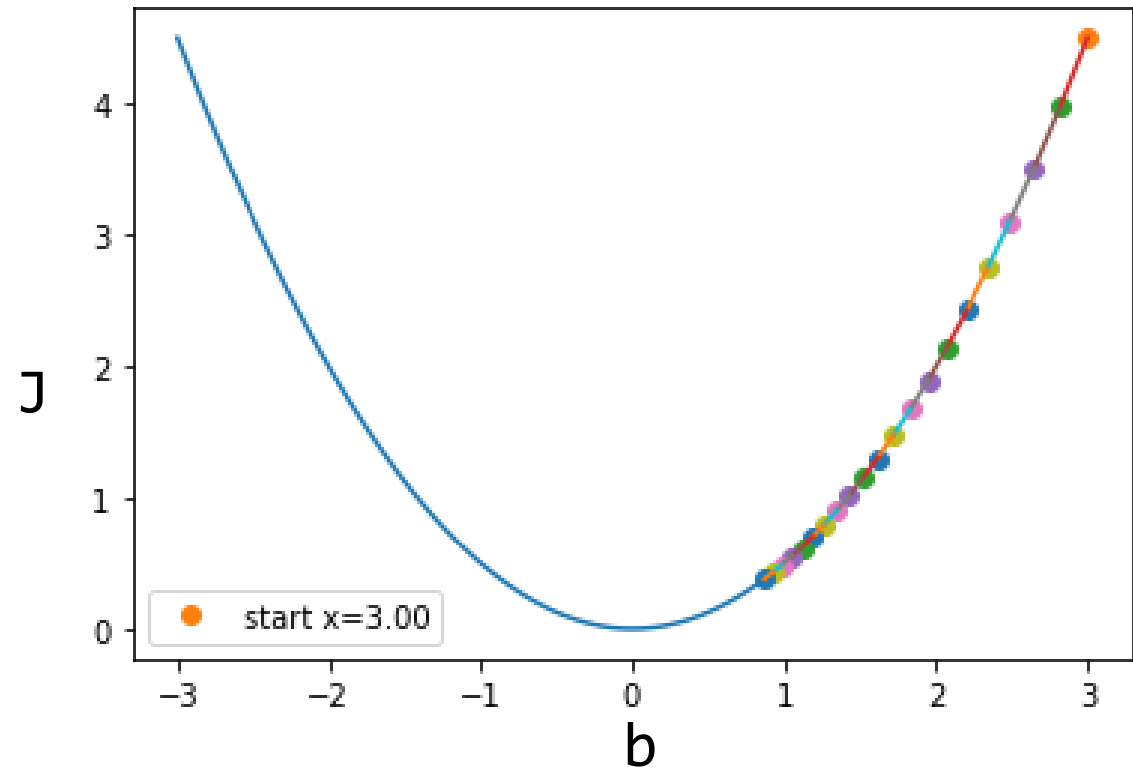
Gradient descent algorithm

2.2 SGD (Stochastic Gradient Descent)

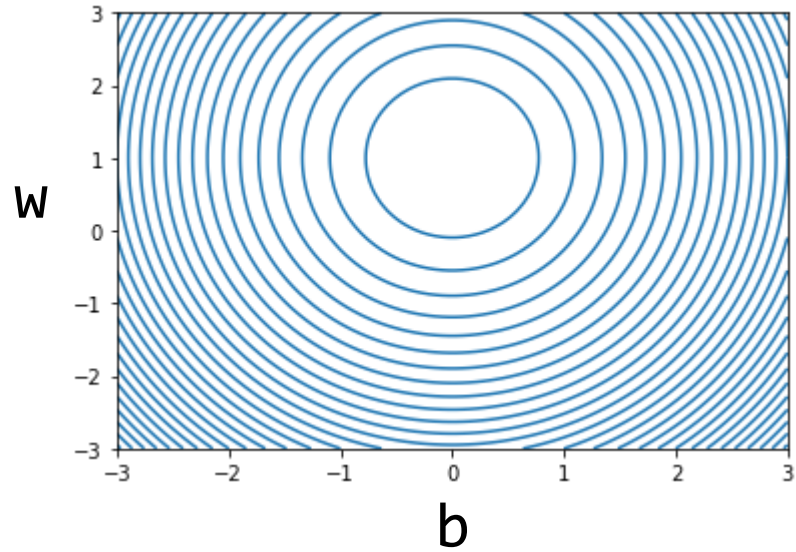
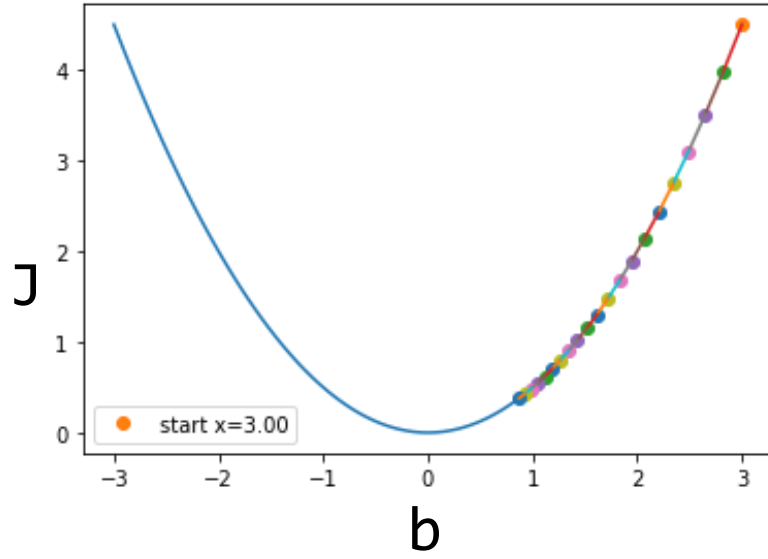
수렴까지 반복

$$\left\{ \begin{array}{l} b = b - \alpha * (\hat{y} - y) \end{array} \right.$$

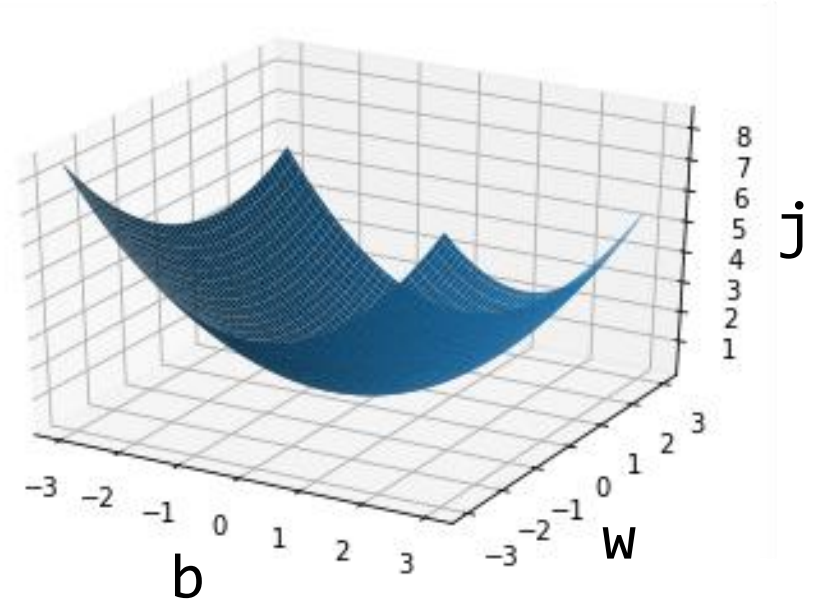
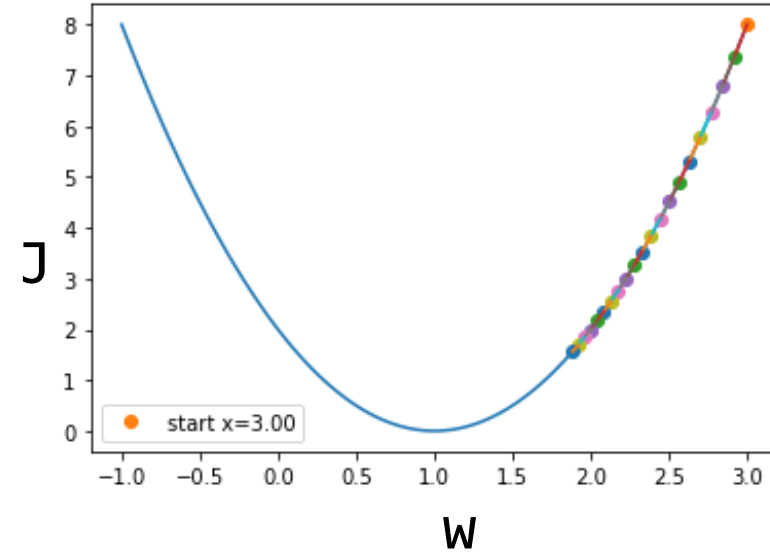
α 가 0.03인 경우



Gradient descent algorithm



2.2 SGD (Stochastic Gradient Descent)



2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

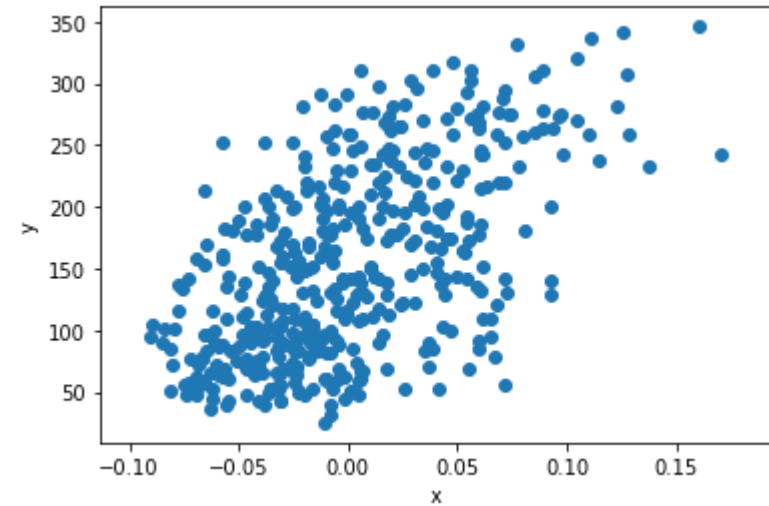
2.4 선형회귀 구현

2.5 시그모이드 함수

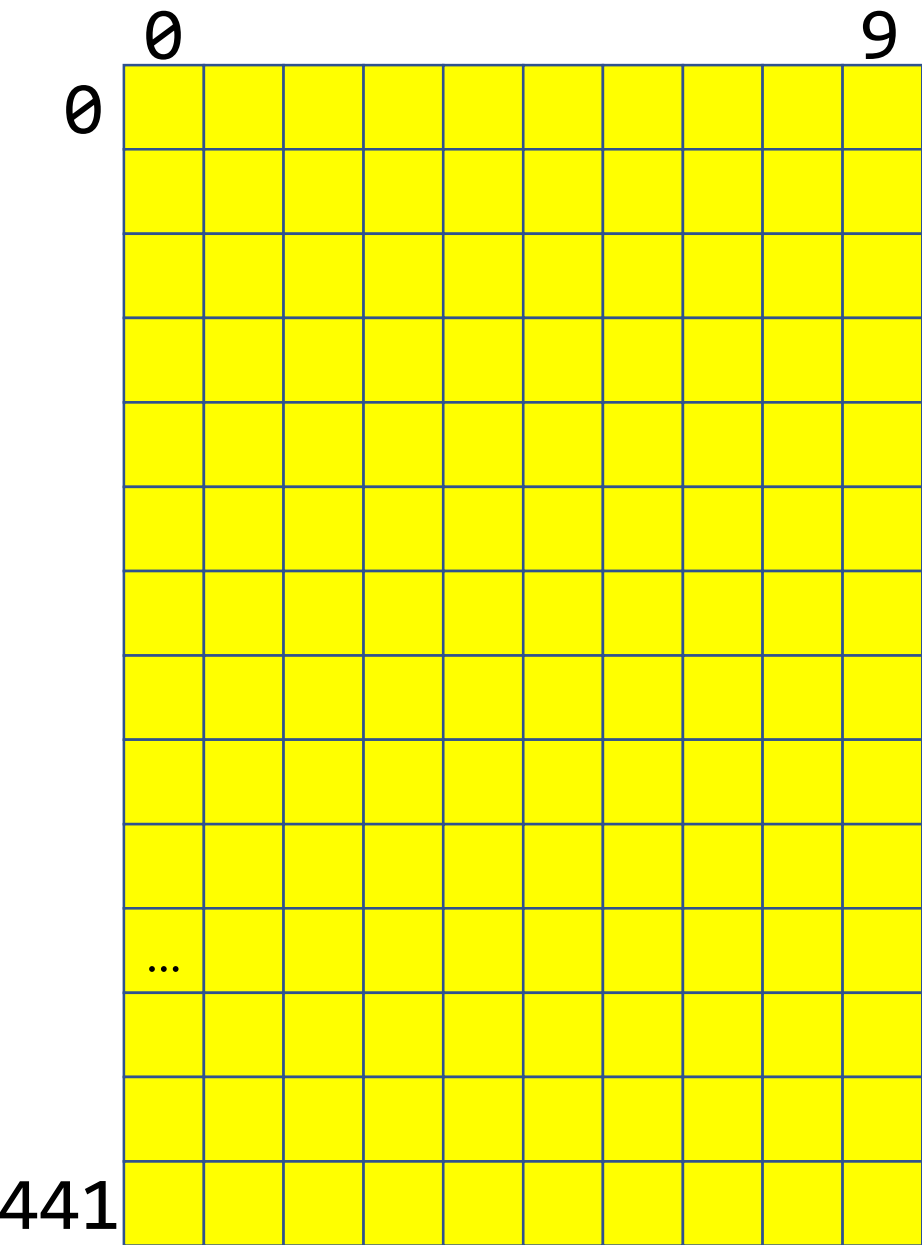
2.6 로지스틱 회귀 구현

data set 준비(sklearn 이용)

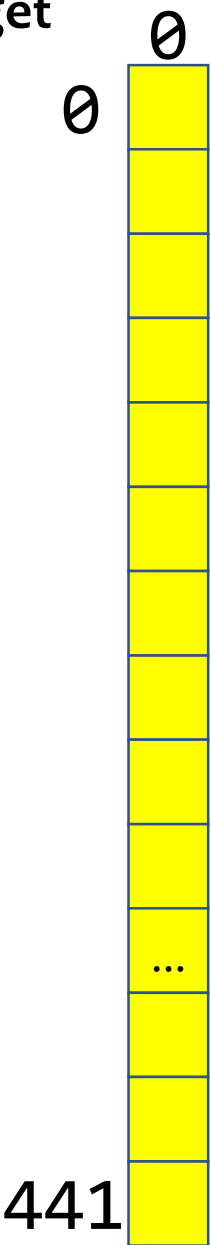
```
from sklearn.datasets import  
load_diabetes  
diabetes = load_diabetes()  
import matplotlib.pyplot as plt  
  
x = diabetes.data[:, 2]  
y = diabetes.target  
  
plt.scatter(x, y)  
plt.xlabel('x')  
plt.ylabel('y')  
plt.show()
```



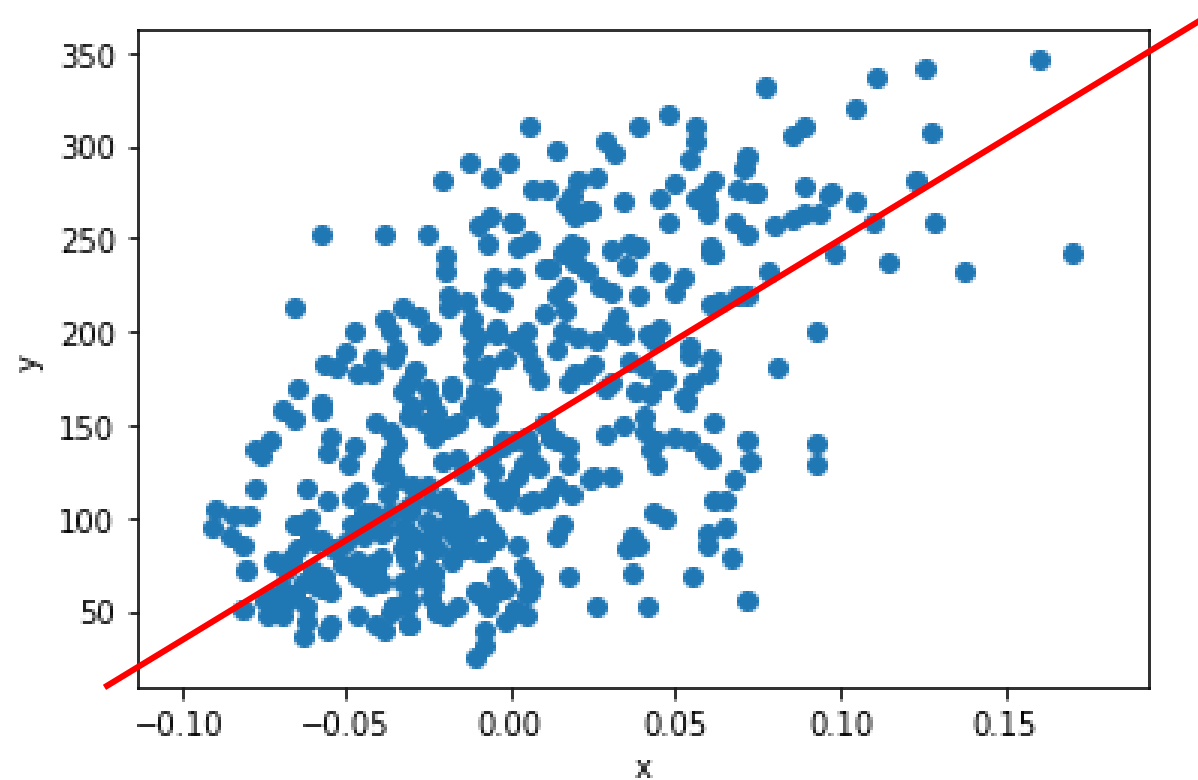
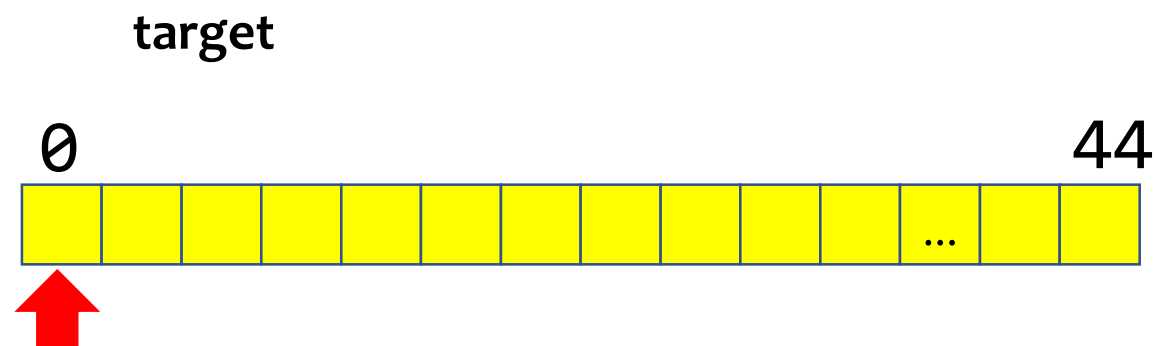
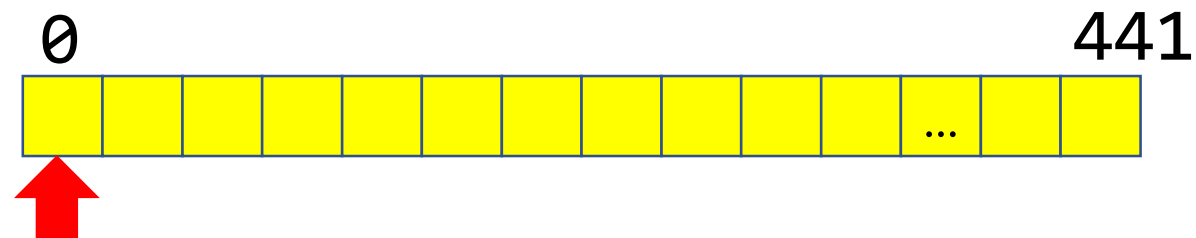
data



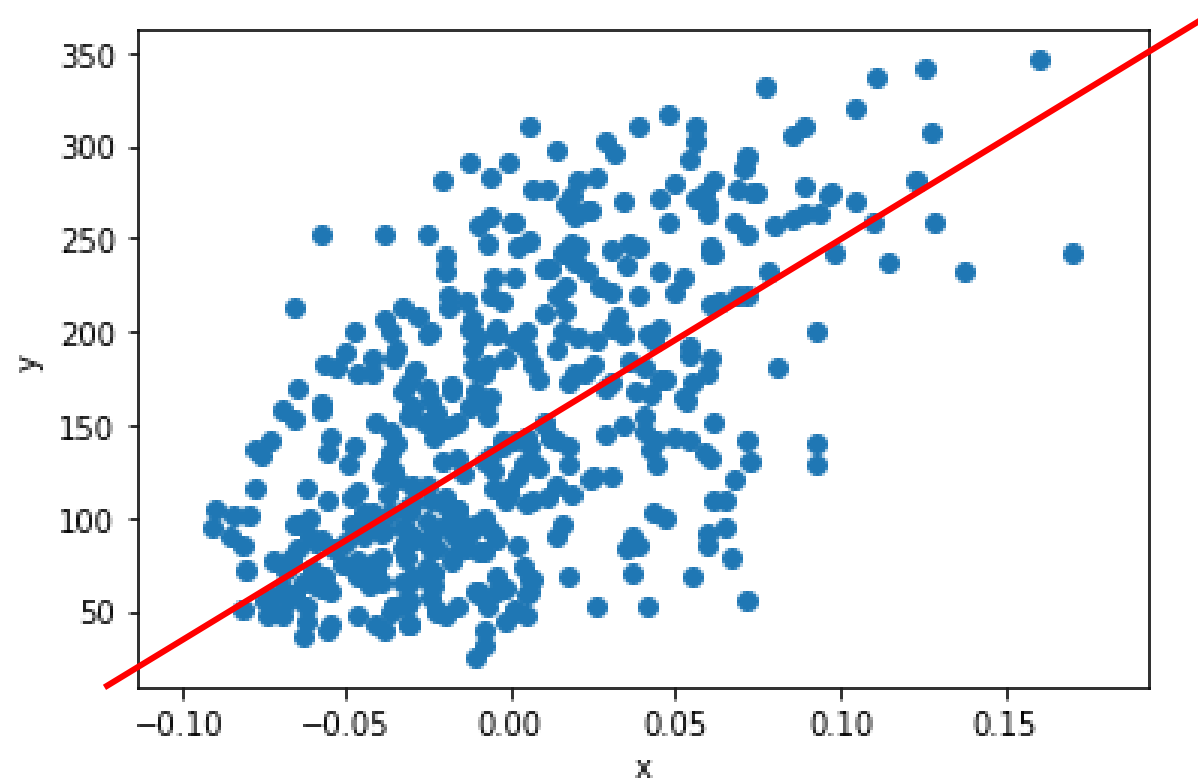
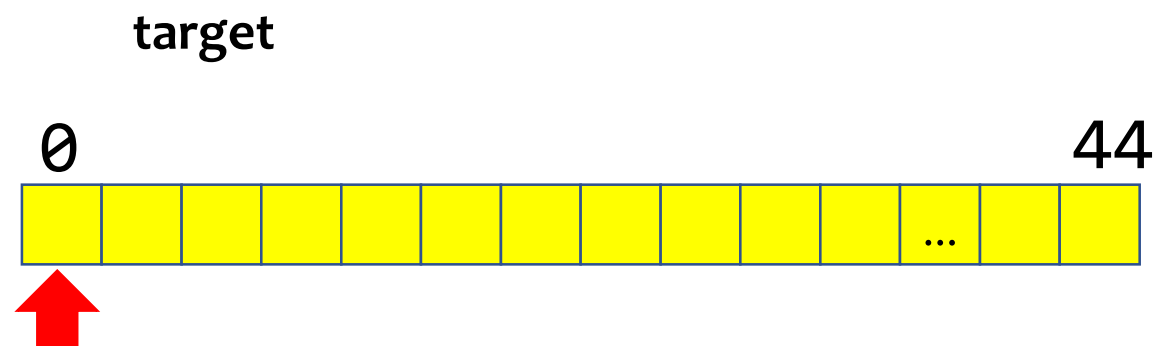
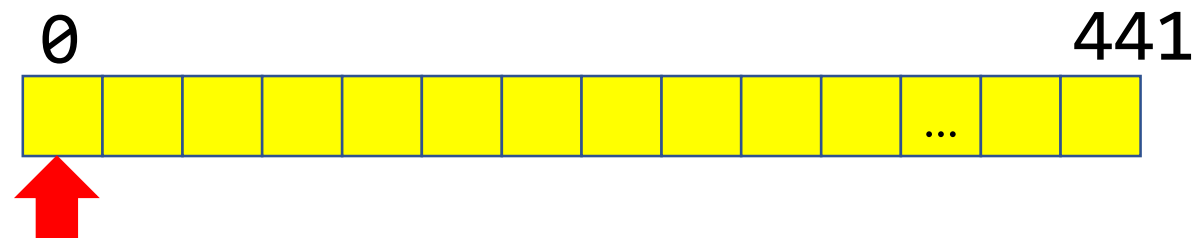
target



`x = diabetes.data[:, 2]`



`x = diabetes.data[:, 2]`



역전파 구현 I

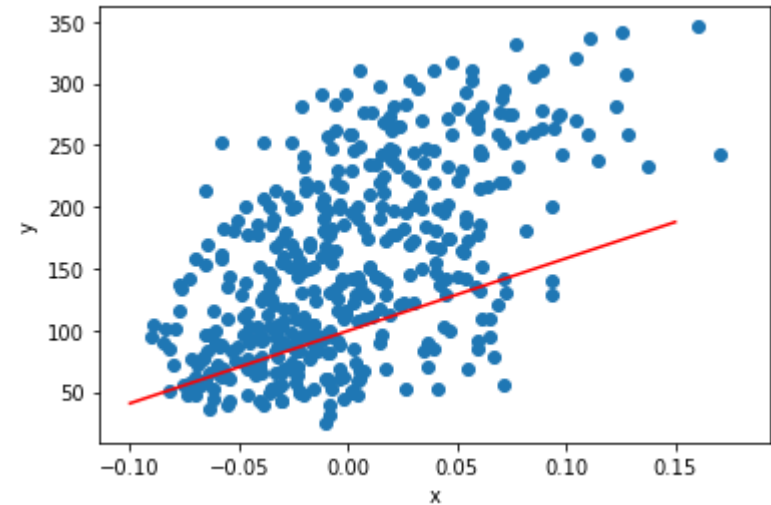
역전파 구현

1. batch=1
2. 점의 전체 개수만큼 1번 순회 (epoch=1)
 - : SGD=>원소 하나마다 w,b 갱신

```
w = 1.0
b = 1.0
rate = 0.01

for x_i, y_i in zip(x, y):
    y_hat = x_i * w + b
    err = y_hat - y_i
    w = w - rate * err * x_i
    b = b - rate * err * 1
```

2.3 역전파 구현



역전파 구현 I

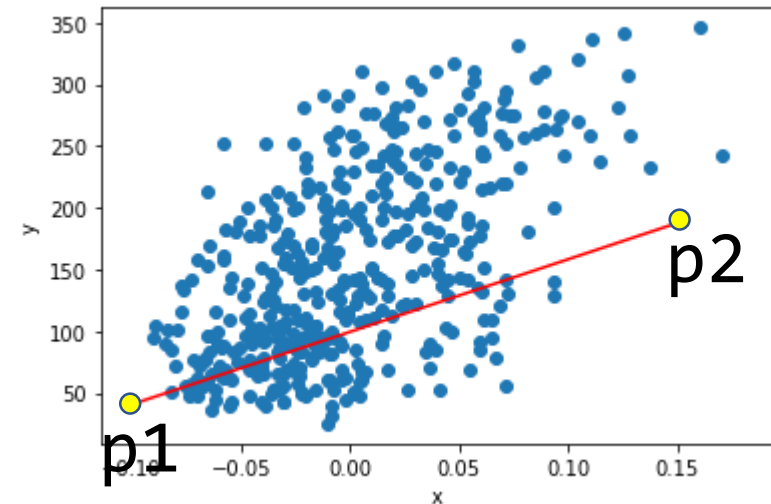
역전파 구현

1. batch=1
2. 점의 전체 개수만큼 1번 순회 (epoch=1)
 - : SGD=>원소 하나마다 w,b 갱신

```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.15, 0.15 * w + b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]], 'r-')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

2.3 역전파 구현

과소 적합 상태



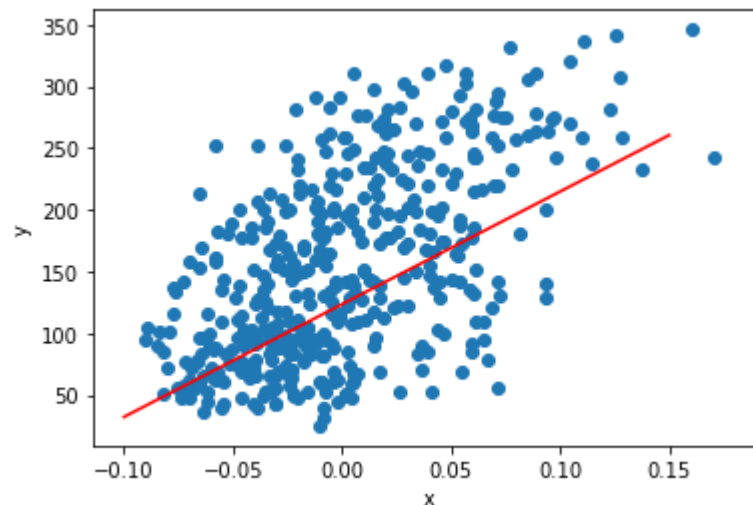
역전파 구현 II

2.3 역전파 구현

역전파 구현

1. batch=1
2. epoch=100번 만큼 외부 루프에서 순회
3. 점의 전체 개수만큼 1번 순회
: SGD=>원소 하나마다 w,b 갱신

```
for i in range(1, 100):  
    for x_i, y_i in zip(x, y):  
        y_hat = x_i * w + b  
        err = y_i - y_hat  
        w_rate = x_i  
        w = w + w_rate * err  
        b = b + 1 * err  
  
plt.scatter(x, y)  
pt1 = (-0.1, -0.1 * w + b)  
pt2 = (0.15, 0.15 * w + b)  
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]],  
         'r-')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.show()
```



2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

Neuron class 만들기

```
class Neuron:
    def __init__(self):
        # 초기화 작업을 수행합니다.
        ...

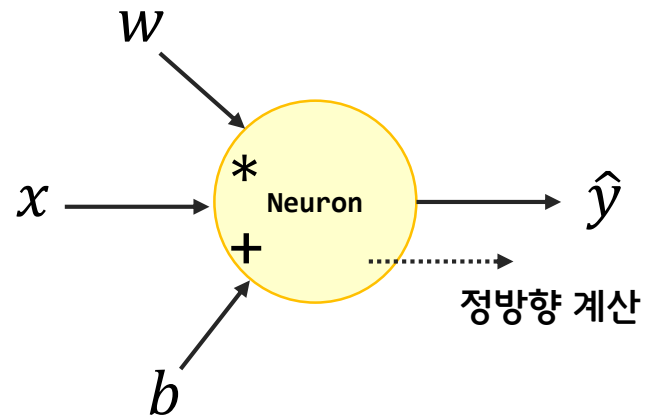
    # 필요한 함수를 추가합니다.
    ...
```

1. __init__() 함수 만들기

```
def __init__(self):
    self.w = 1.0
    self.b = 1.0
    self.rate = 0.01
```

2. 정방향 계산 만들기

```
def forpass(self, x):  
    y_hat = x * self.w + self.b    # 직선 방정식을 계산합니  
다  
    return y_hat
```



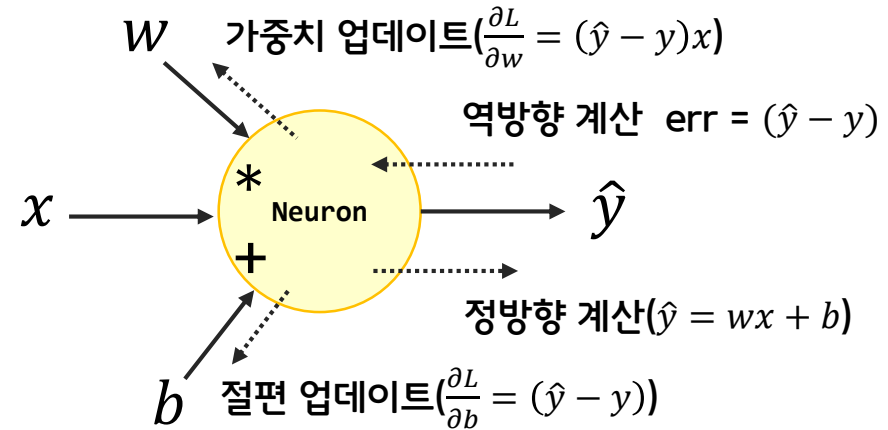
$$\hat{y} = wx + b$$

3. 역방향 계산 만들기

```
def backprop(self, x, err):  
    w_grad = x * err    # 가중치에 대한 그래디언트를 계산합니다  
    b_grad = 1 * err    # 절편에 대한 그래디언트를 계산합니다  
    return w_grad, b_grad
```

$$\frac{\partial L}{\partial w} = (\hat{y} - y)x$$

$$\frac{\partial L}{\partial b} = (\hat{y} - y)$$



4. 훈련을 위한 fit() 함수 구현

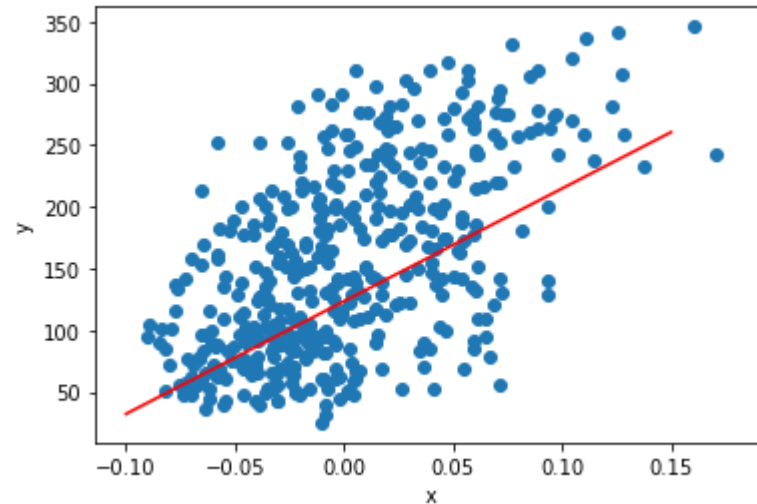
```
def fit(self, x, y, epochs=100):  
    for i in range(epochs):          # 에포크만큼 반복합니다  
        for x_i, y_i in zip(x, y):  # 모든 샘플에 대해 반복합니다  
            y_hat = self.forpass(x_i) # 정방향 계산  
            err = y_hat - y_i         # 오차 계산  
            w_grad, b_grad = self.backprop(x_i, err) # 역방향 계산  
            self.w -= self.rate*w_grad # 가중치 업데이트  
            self.b -= self.rate*b_grad # 절편 업데이트
```

5. 모델 훈련하기

```
neuron = Neuron()  
neuron.fit(x, y)
```


6. 학습이 완료된 모델의 가중치와 절편 확인

```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * neuron.w + neuron.b)
pt2 = (0.15, 0.15 * neuron.w + neuron.b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]], 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Logistic Regression

- 로지스틱 회귀 란?
 - Classification(분류)
 - Logistic vs Linear
- 동작 방식
 - 가설 표현
 - Sigmoid 함수
 - Decision Boundary(결정경계)
 - Cost Function
 - Optimizer (Gradient Descent)

Classification

Binary Classification(이진 분류) 란?

: 값은 0 또는 1

- Exam : Pass of **Fail**
- Spam : Not Spam of **Spam**
- Face : Real of **Fake**
- Tumor : Not Malignant or **Malignant**

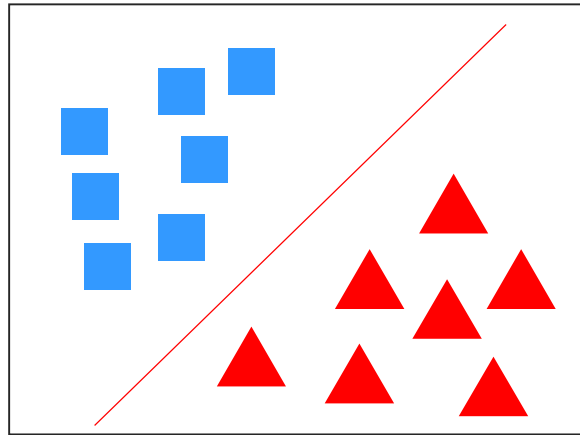
머신 러닝으로 시작하려면 변수를 인코딩해야 한다.

```
x_train = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
```

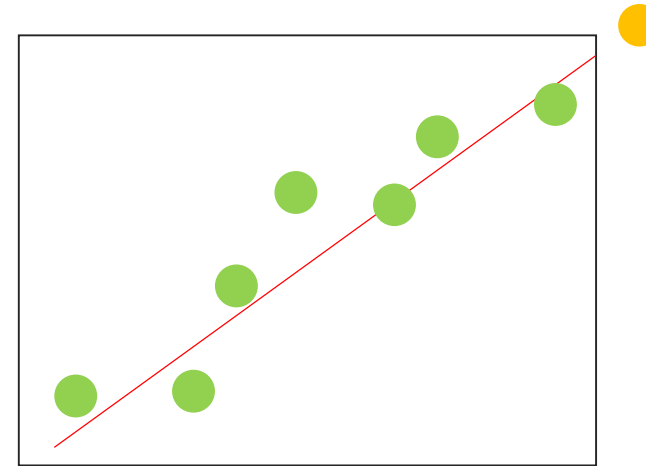
```
y_train = [[0], [0], [0], [1], [1], [1]]
```

Logistic vs Linear

로지스틱 회귀와 선형 회귀의 차이점은?



Discrete(분리) : 분류 목적
신발 사이즈 / 회사의 근로자수



Continuous (지속적인) : 값의 예측
시간/ 무게 / 높이

머신 러닝으로 시작하려면 변수를 인코딩해야 한다.

```
Logistic_Y = [[0], [0], [0], [1], [1], [1]]
```

```
Linear_Y = [[828.2], [764.2], [123.5], [342.5], [987.3],  
[234.1]]
```

keras로 구현한 이진 분류 문제

```
from keras.models import Sequential # 케라스의 Sequential()을 임
포트
from keras.layers import Dense      # 케라스의 Dense()를 임포트
from keras import optimizers        # 케라스의 옵티마이저를 임포트
import numpy as np # Numpy를 임포트
import matplotlib.pyplot as plt

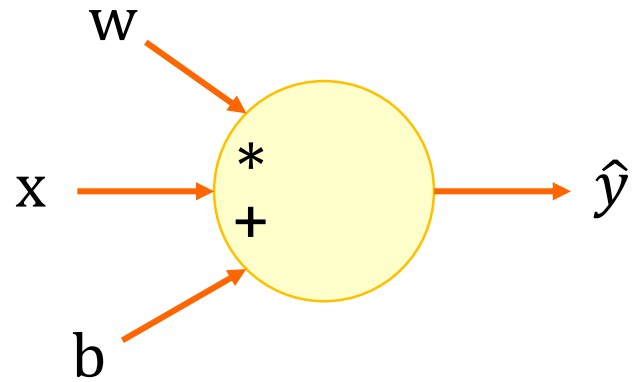
X=np.array([1,2,3,4,5,6,7,8])
Y=np.array([0,0,0,0,1,1,1,1])

model=Sequential()
model.add(Dense(1, input_dim=1, activation='sigmoid'))
RMSprop=optimizers.RMSprop(lr=0.01)
model.compile(optimizer=RMSprop ,loss='binary_crossentropy',metr
ics=['accuracy'])
model.fit(X,Y, batch_size=1, epochs=200, shuffle=False)

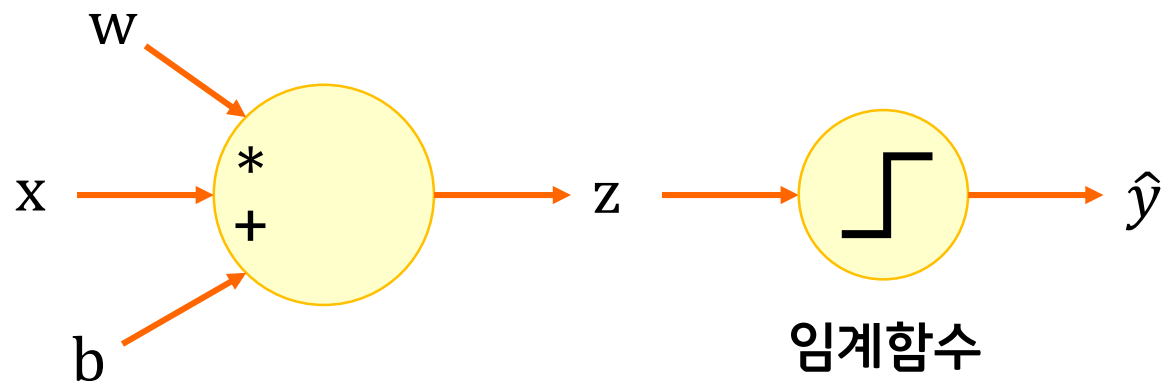
plt.plot(X, Y, 'rx')
print(X)
```

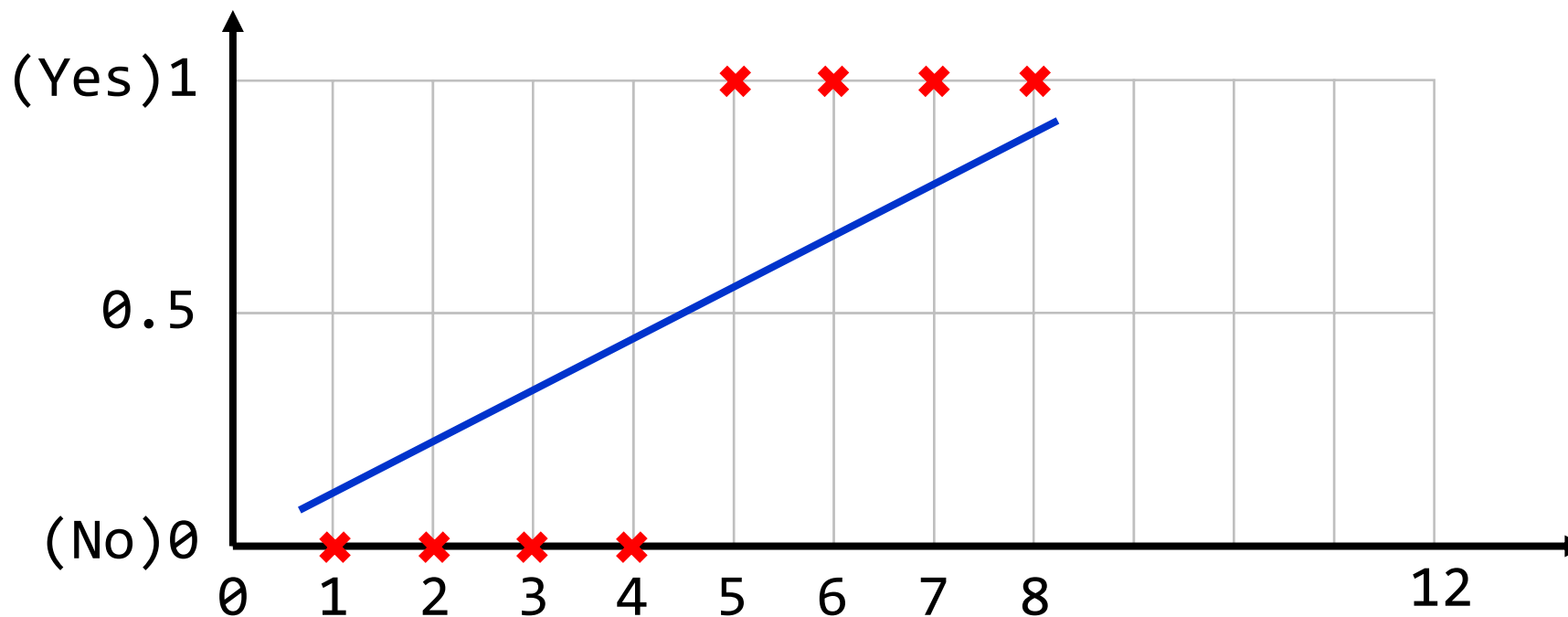
퍼셉트론

2.5 시그모이드 함수



퍼셉트론





Threshold classifier output at 0.5:

If $\hat{y} \geq 0.5$, predict "y=1"

If $\hat{y} < 0.5$, predict "y=0"

1 => 양성

2 => 양성

3 => 양성

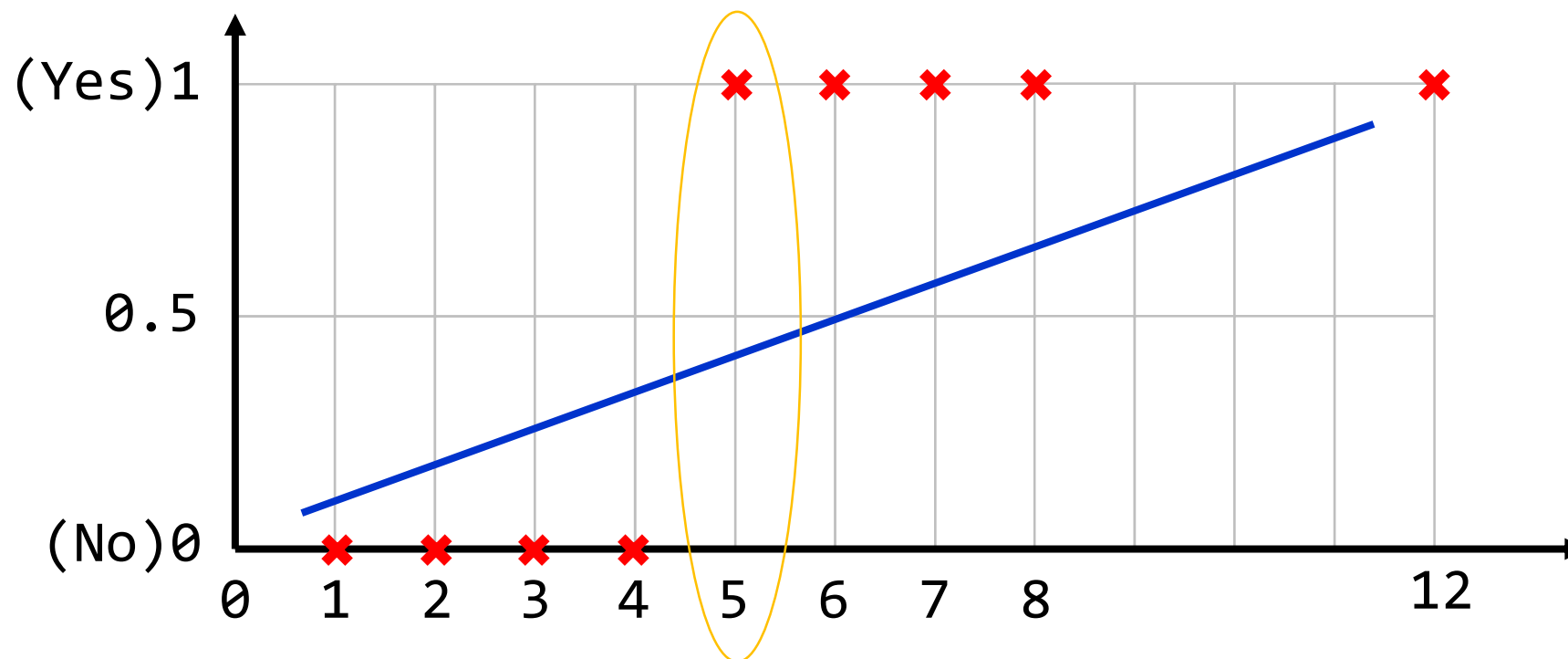
4 => 양성

5 => 악성

6 => 악성

7 => 악성

8 => 악성



Threshold classifier output at 0.5:

If $\hat{y} \geq 0.5$, predict "y=1"

If $\hat{y} < 0.5$, predict "y=0"

1 => 양성

2 => 양성

3 => 양성

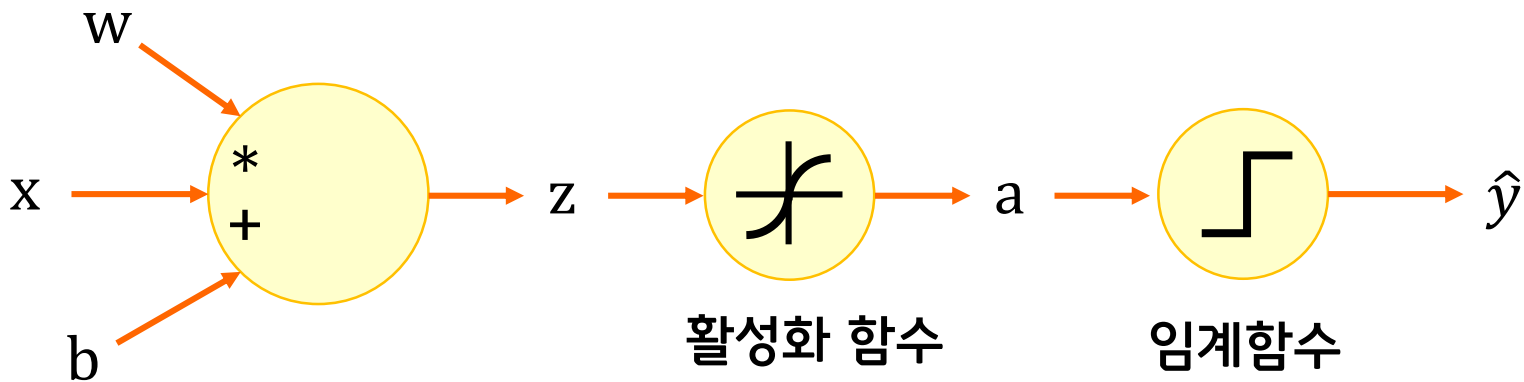
4 => 양성

5 => 양성

6 => 악성

7 => 악성

8 => 악성



Classification: $y=0$ or 1

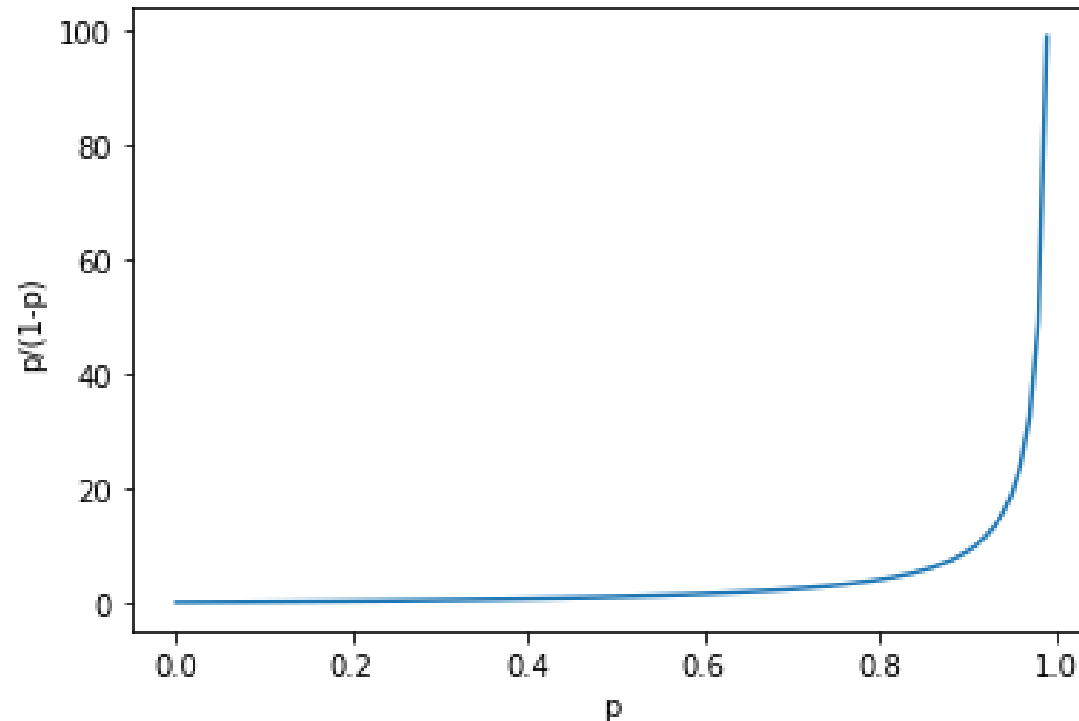
\hat{y} can be > 1 or < 0

Logistic Regression: $0 < \hat{y} < 1$

분류 문제 사용

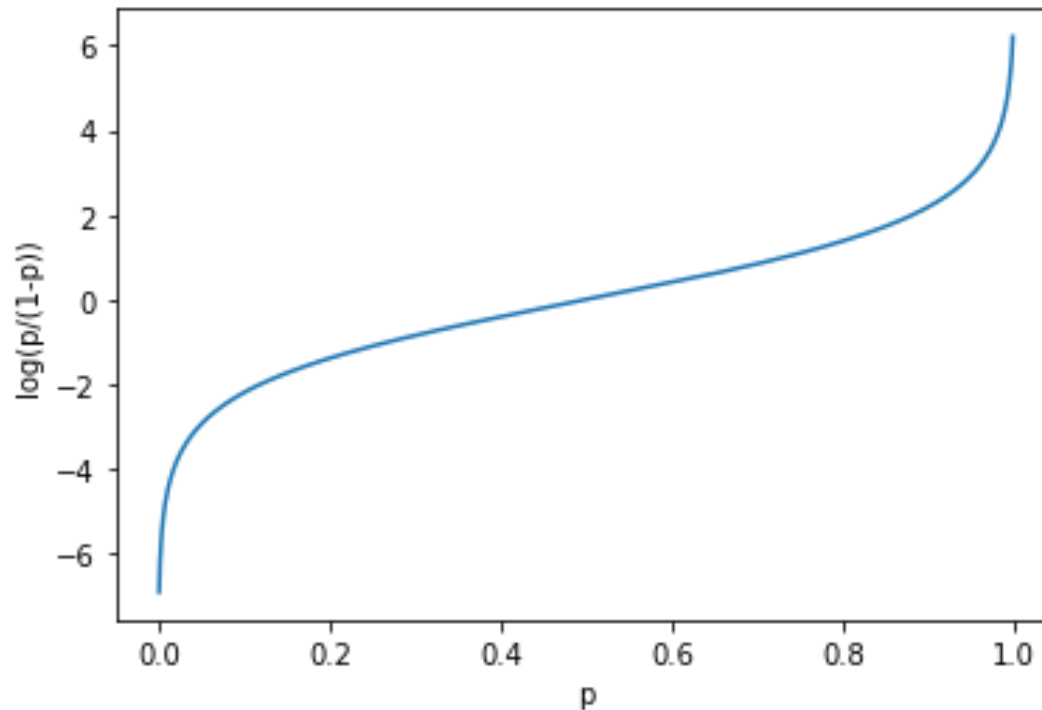
$$OR(odds\ ratio) = \frac{p}{1-p} \quad (p=\text{성공확률})$$

오즈 비를 그래프로 그리면 다음과 같다.
p가 0부터 1까지 증가할 때 오즈 비의 값은 처음에는
천천히 증가하지만 p가 1에 가까워지면 급격히 증가한다.



$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) \quad (p=\text{성공확률})$$

로짓 함수는 p 가 0.5일 때 0이 되고 p 가 0과 1일 때 각각 무한대로 음수와 양수가 되는 특징을 가진다.



시그모이드 함수(Sigmoid function)

2.5 시그모이드 함수

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = z$$

로지틱 함수의 유도 : p에 대해 정리

$$\log\left(\frac{p}{1-p}\right) = z$$

$$e^{\log\left(\frac{p}{1-p}\right)} = e^z$$

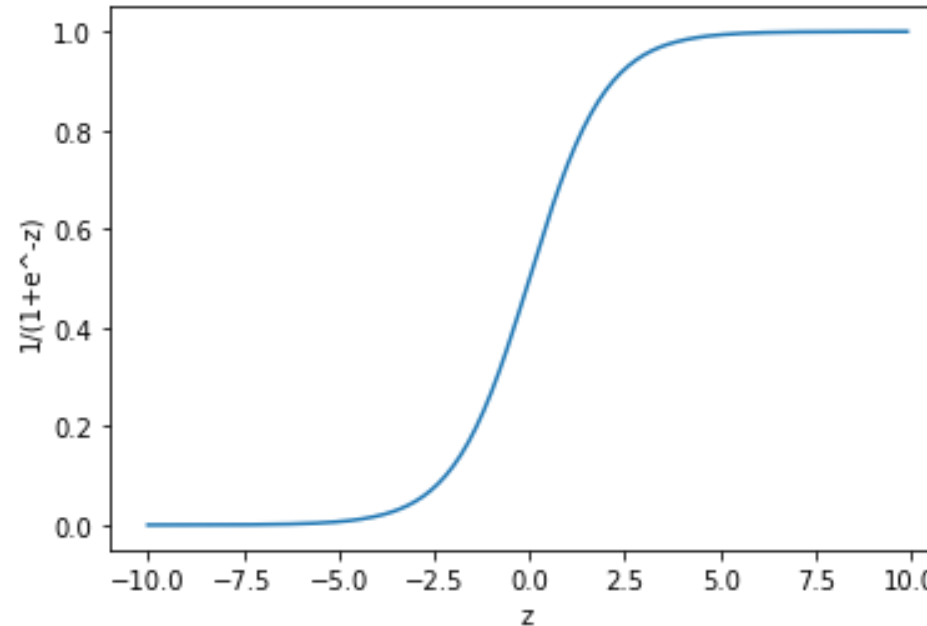
$$\frac{p}{1-p} = e^z$$

$$p = (1 - p) * e^z$$

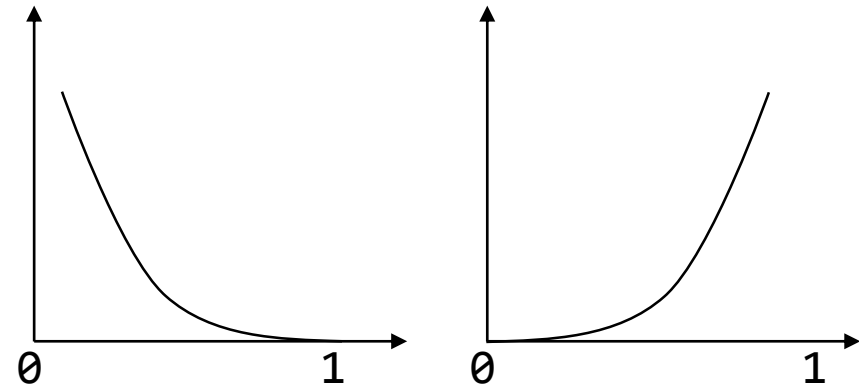
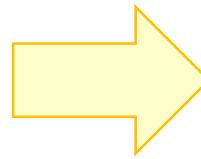
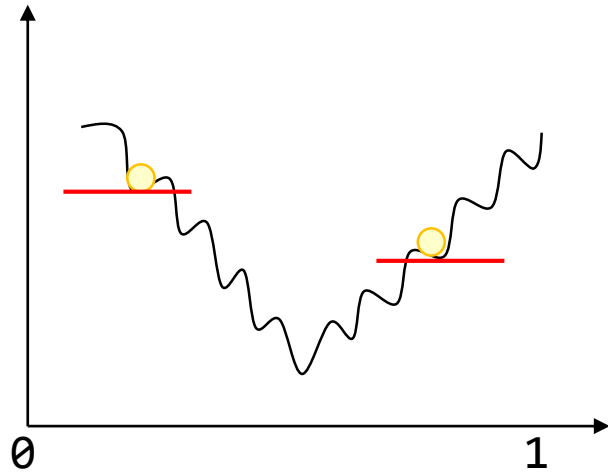
$$p = e^z - p * e^z$$

$$p + p * e^z = e^z$$

$$p = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$



A convex logistic regression cost function



$$\text{sigmoid} - \text{slope} = \text{cost}$$

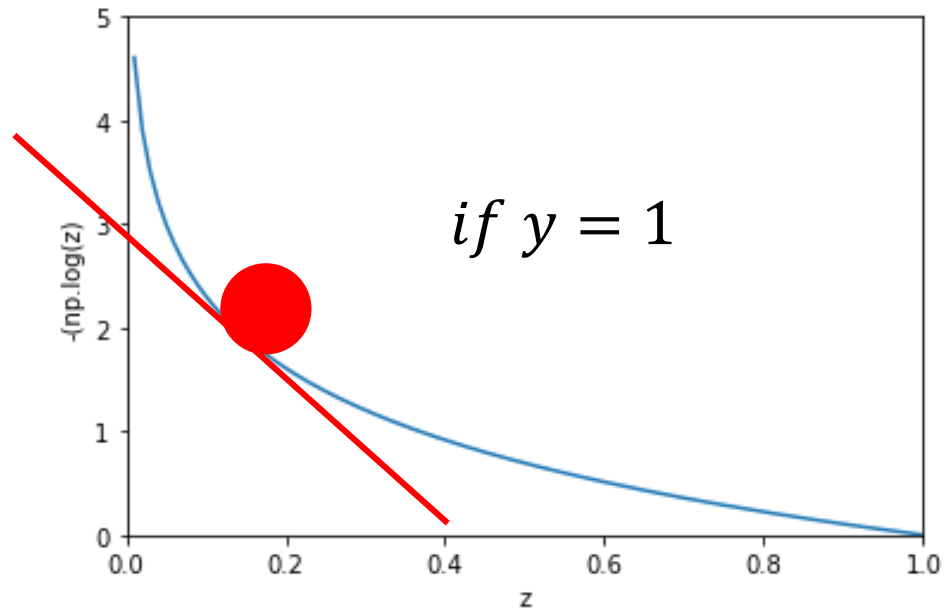
$$J(w) = \frac{1}{2} (\text{sigmoid}(\hat{y}) - y)^2$$

$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = 0 \end{cases}$$

$$\text{cost}(\hat{y}, y) = -(y * \log(a) + (1 - y) \log(1 - a))$$

Logistic regression cost function

$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = 0 \end{cases}$$

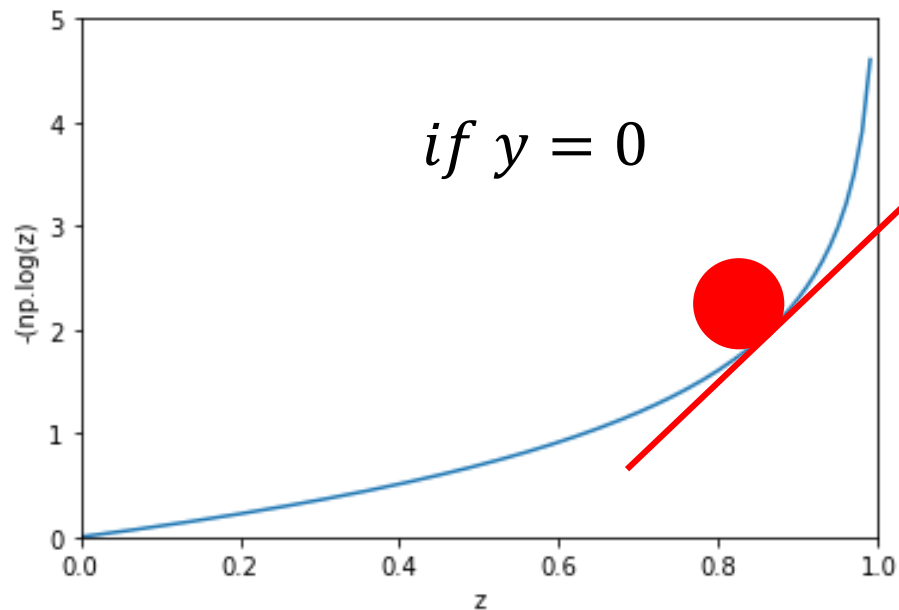


Cost = 0 if $y = 1$, $\hat{y} = 1$

But as $\hat{y} \rightarrow 0$
Cost $\rightarrow \infty$

Logistic regression cost function

$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = 0 \end{cases}$$

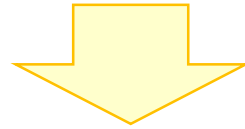


Cost = 0 if $y = 0$, $\hat{y} = 0$

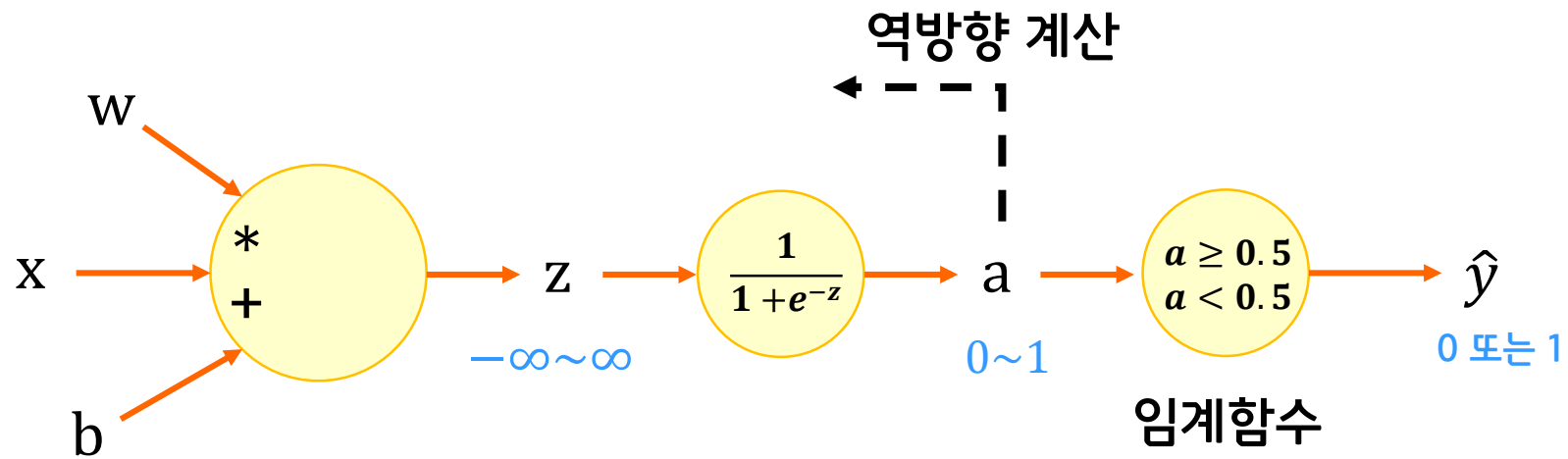
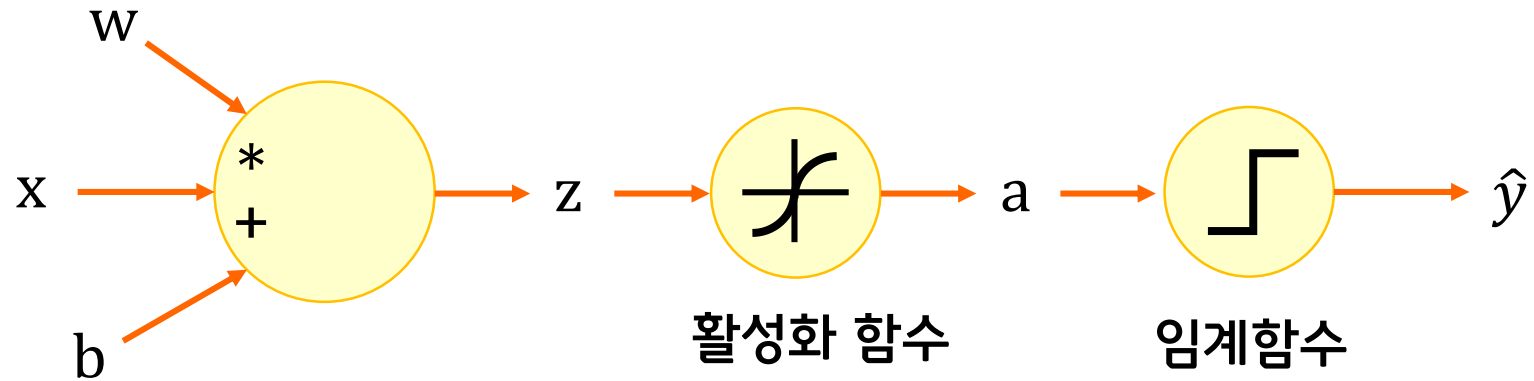
But as $\hat{y} \rightarrow 1$
Cost $\rightarrow \infty$

Logistic regression cost function

$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = 0 \end{cases}$$

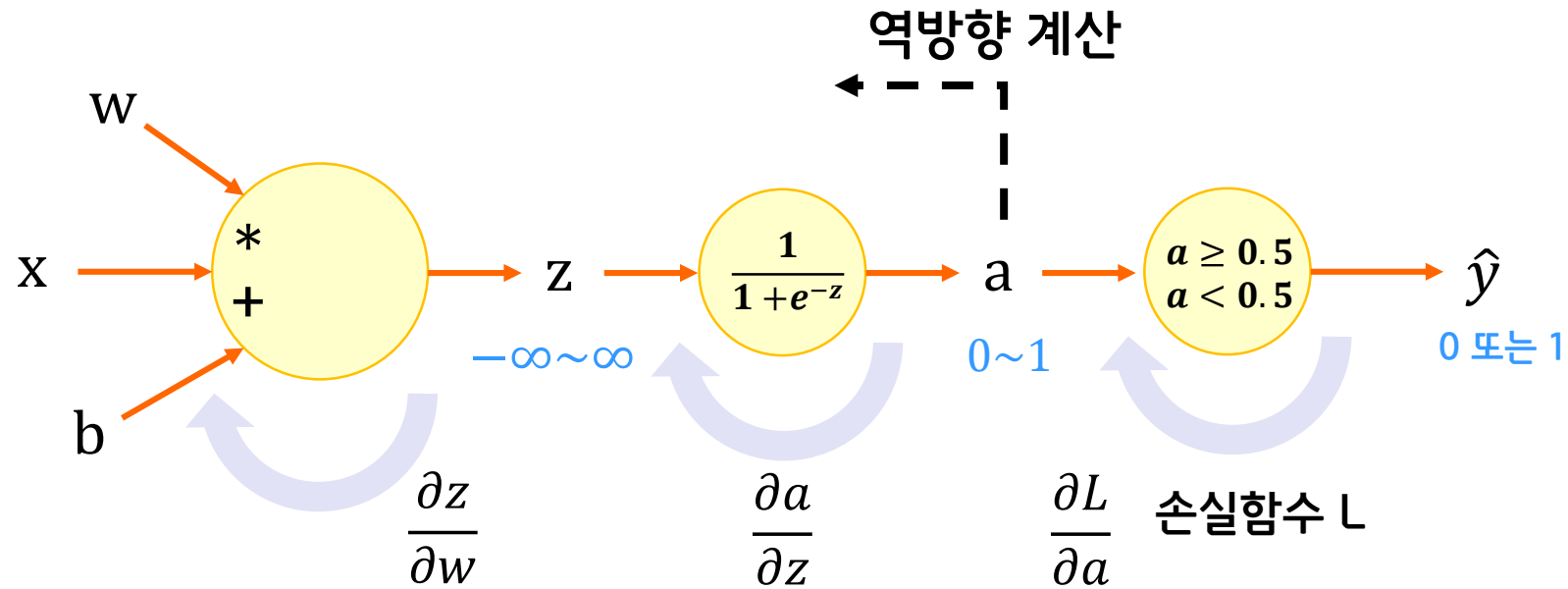


$$L = -(y * \log(a) + (1 - y) \log(1 - a))$$



특성이 하나인 경우 => x 1개

$$z = wx + b$$



chain rule을 이용하여 각 단계의 미분 결과를 곱한다.

w에 대하여 미분

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w} = (a - y)x$$

$$\frac{\partial L}{\partial a} = -(y \frac{1}{a} - (1 - y) \frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial w} = x$$

b에 대하여 미분

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial b} = (a - y)1$$

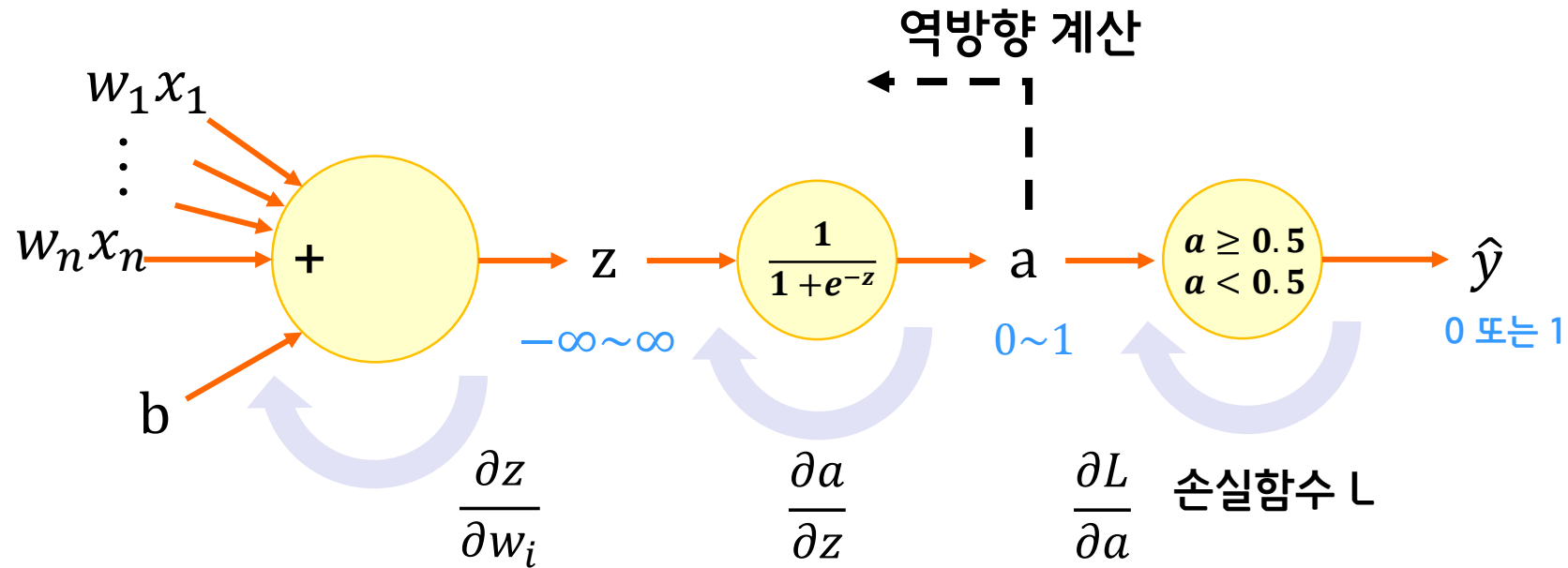
$$\frac{\partial L}{\partial a} = -(y \frac{1}{a} - (1 - y) \frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial b} = 1$$

특성이 여러개인 경우 => x n개

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$



chain rule을 이용하여 각 단계의 미분 결과를 곱한다.

w에 대하여 미분

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w_i} = -(y - a)x_i$$

$$\frac{\partial L}{\partial a} = -(y \frac{1}{a} - (1 - y) \frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial w_i} = x_i$$

b에 대하여 미분

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial b} = -(y - a)1$$

$$\frac{\partial L}{\partial a} = -(y \frac{1}{a} - (1 - y) \frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial b} = 1$$

$$\hat{y} = xw + b$$

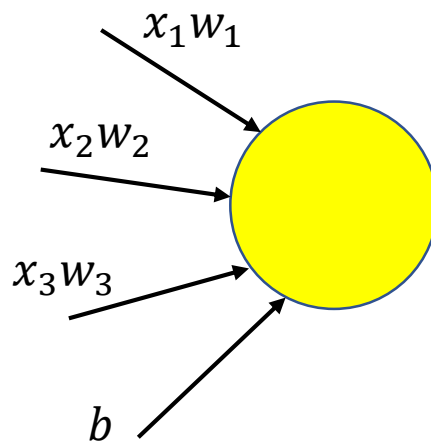
$$w \leftarrow (\hat{y} - y)x \quad b \leftarrow (\hat{y} - y)1$$

$$\hat{y} = x_1w_1 + x_2w_2 + x_3w_3 + \cdots + b$$

$$w_1 \leftarrow (\hat{y} - y)x_1$$

$$w_2 \leftarrow (\hat{y} - y)x_2$$

$$w_3 \leftarrow (\hat{y} - y)x_3$$



로지스틱 회귀 손실 함수

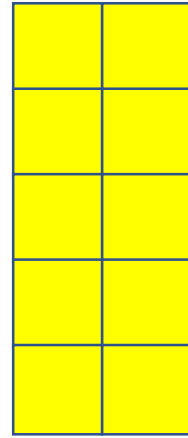
$$\hat{y}_1 = x_1 w_1^1 + x_2 w_1^2 + b_1$$

$$\hat{y}_2 = x_1 w_2^1 + x_2 w_2^2 + b_2$$

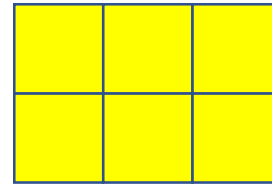
$$\hat{y}_3 = x_1 w_3^1 + x_2 w_3^2 + b_3$$

2.5 시그모이드 함수

$(m, 2)$



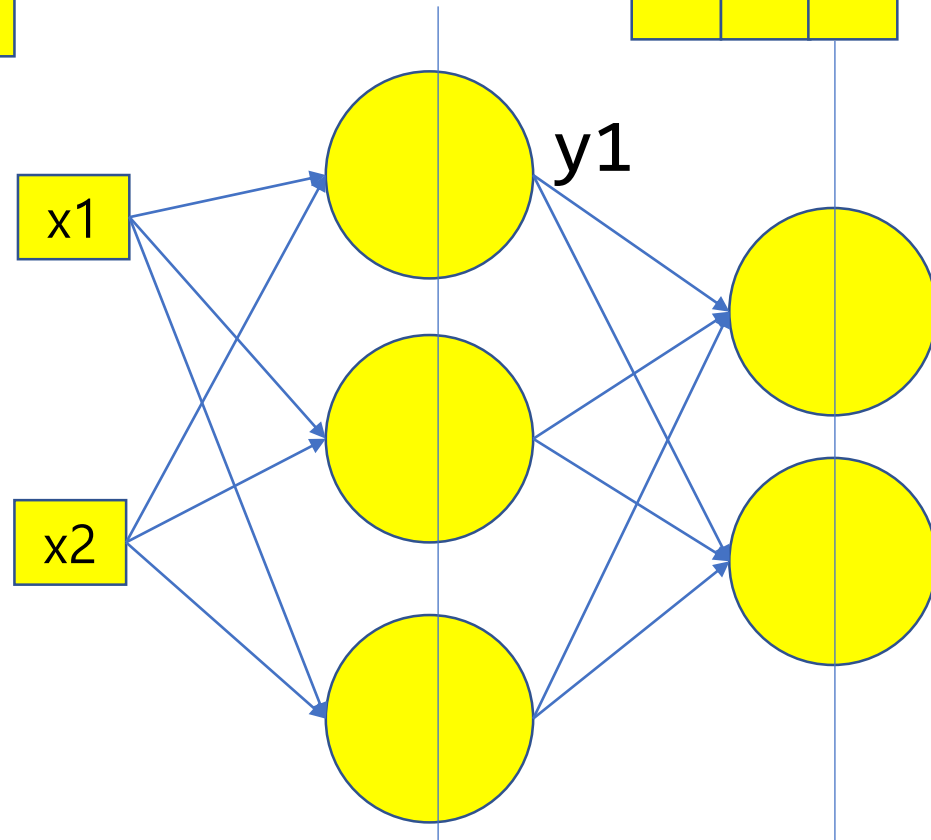
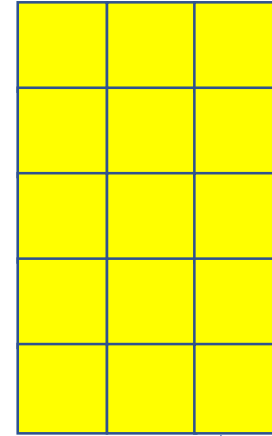
$(2, 3)$



*

=

$(m, 3)$



로지스틱 회귀 손실 함수

2.5 시그모이드 함수

$$w = w - (\text{예측값} - \text{실제값}) * x$$

$$w_0 = (\text{예측값} - \text{실제값}) * x_0$$

$$b = b - (\text{예측값} - \text{실제값})$$

$$w_1 = (\text{예측값} - \text{실제값}) * x_1$$

제곱 오차의 미분과 로지스틱 손실 함수의 미분은 \hat{y} 이 a로 바뀌었을 뿐 동일 하다.

따라서 선형함수의 결과 값을 activation 함수인 시그모이드를 적용한 값이 a 이다.

	제곱 오차의 미분	로지스틱 손실 함수의 미분
가중치에 대한 미분	$\frac{\partial SE}{\partial w} = (\hat{y} - y)x$	$\frac{\partial L}{\partial w} = (a - y)x_i$
절편에 대한 미분	$\frac{\partial SE}{\partial b} = (\hat{y} - y)1$	$\frac{\partial L}{\partial b} = (a - y)1$

분류용 데이터셋을 준비(위스콘신 유방암 데이터 세트)

	의학	이진 분류
좋은	양성 종양	0
나쁨	악성 종양	1

사이킷런의 datasets 에서 불러온다.

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print(cancer.data.shape, cancer.target.shape)
```

data와 target의 크기를 알아본다.

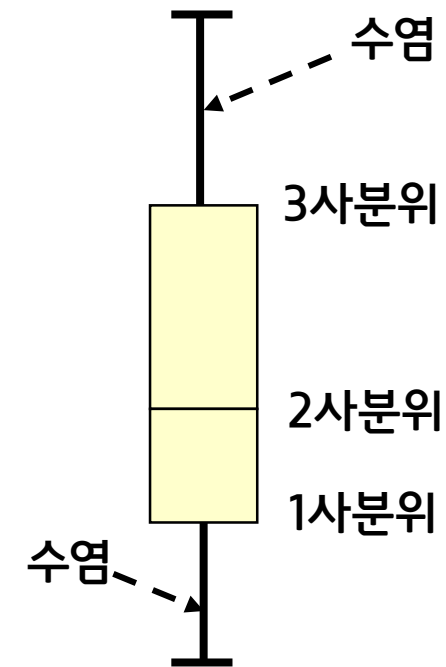
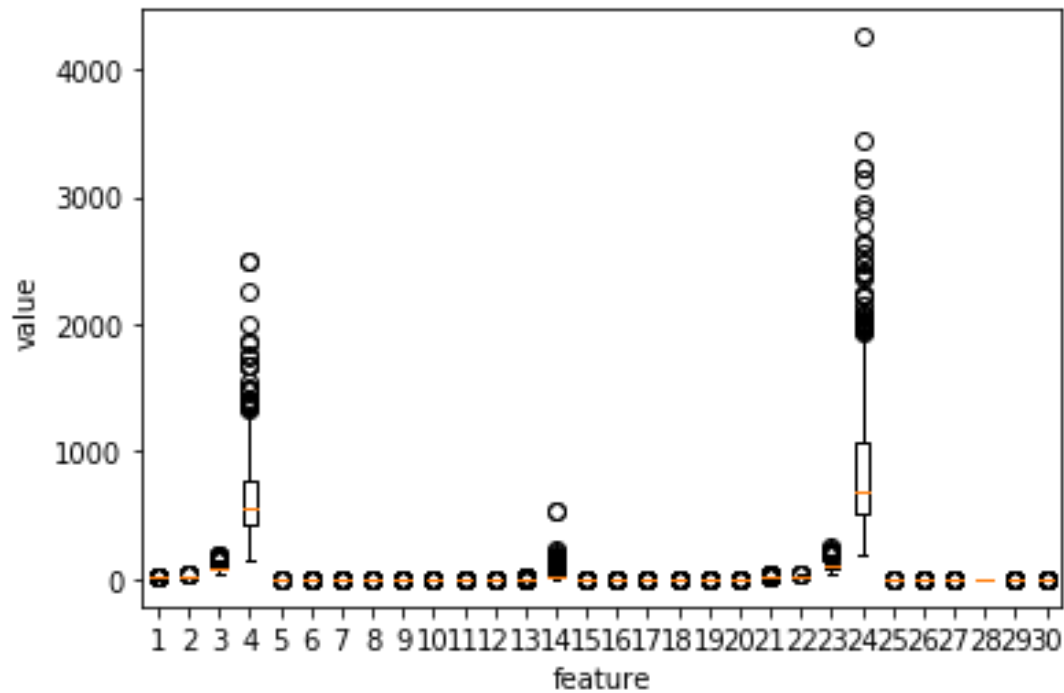
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print(cancer.data.shape, cancer.target.shape)
```

cancer에는 569개의 샘플과 30개의 특성이 있다.

```
cancer.data[:3]
array([[1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
        3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
        8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
        3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
        1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, 1.326e+03, 8.474e-02, 7.864e-02,
        8.690e-02, 7.017e-02, 1.812e-01, 5.667e-02, 5.435e-01, 7.339e-01,
        3.398e+00, 7.408e+01, 5.225e-03, 1.308e-02, 1.860e-02, 1.340e-02,
        1.389e-02, 3.532e-03, 2.499e+01, 2.341e+01, 1.588e+02, 1.956e+03,
        1.238e-01, 1.866e-01, 2.416e-01, 1.860e-01, 2.750e-01, 8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, 1.203e+03, 1.096e-01, 1.599e-01,
        1.974e-01, 1.279e-01, 2.069e-01, 5.999e-02, 7.456e-01, 7.869e-01,
        4.585e+00, 9.403e+01, 6.150e-03, 4.006e-02, 3.832e-02, 2.058e-02,
        2.250e-02, 4.571e-03, 2.357e+01, 2.553e+01, 1.525e+02, 1.709e+03,
        1.444e-01, 4.245e-01, 4.504e-01, 2.430e-01, 3.613e-01, 8.758e-02]])
```

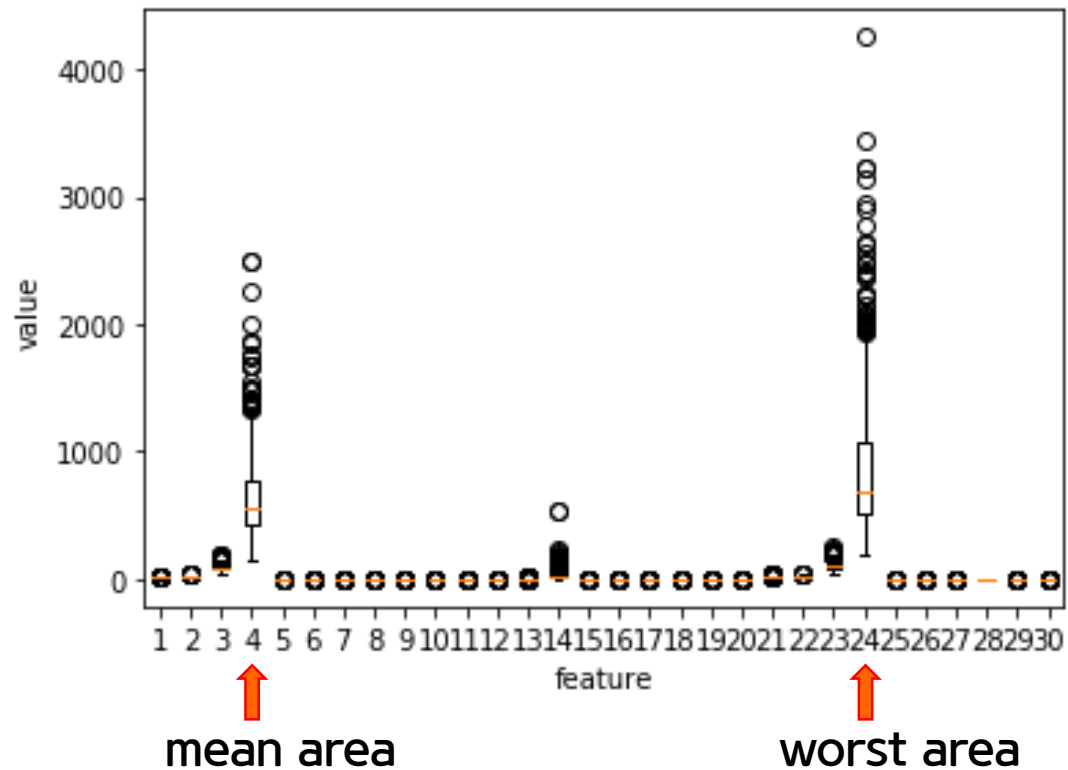
박스 플롯을 이용하면 각 특성의 사분위 값을 볼 수 있다.

```
plt.boxplot(cancer.data)
plt.xlabel('feature')
plt.ylabel('value')
plt.show()
```



눈에 띄는 특성 보기 : 모두 넓이에 관한 특성이다.

```
cancer.feature_names[[3,23]]  
  
array(['mean area', 'worst area'], dtype='<U23')
```



타겟 데이터 확인

```
np.unique(cancer.target, return_counts=True)  
  
(array([0, 1]), array([212, 357], dtype=int64))
```

넘파이의 unique 함수는 중복 제거후 고유한 값을 찾아 반환한다.

cancer.target에는 0=>악성, 1=>악성 의 값이 있다.

return_counts에는 고유한 값인 0의 개수와 1의 개수가 리턴 되므로

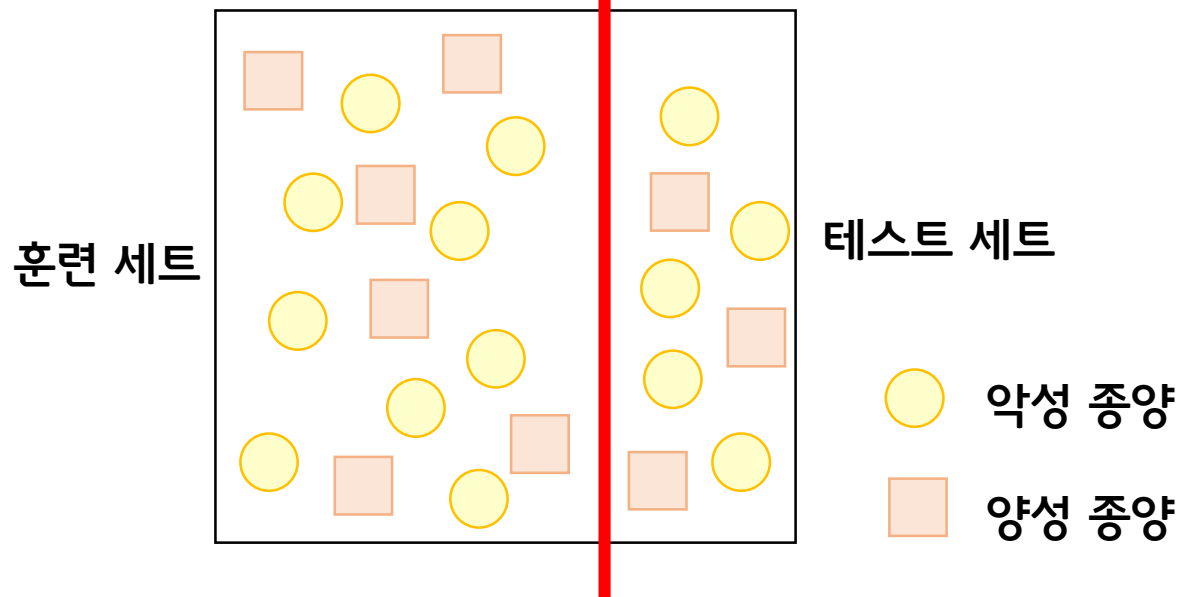
양성종양 212개와 357개의 악성종양 값이 있다.

모델의 성능 평가를 위한 훈련 세트와 테스트 세트

: 모델을 학습 시킨 훈련 데이터 세트로 성능을 평가 하면 상당히 좋은 성능이 나온다.
하지만 이렇게 하면 '과도하게 낙관적으로 일반화 성능을 추정한다' 라고 한다.
따라서 전체 데이터를 나누어 훈련 데이터 세트로 학습시킨 후 테스트 데이터로 성능을
평가 해야 한다.

주의 사항

- 보통 훈련 데이터가 더 많아야 하므로 8:2 정도로 나누어 훈련과 테스트를 진행한다.
- 데이터가 어느 한쪽에 몰리지 않도록 비율이 일정하게 골고루 섞어야 한다.



타겟 데이터 확인

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    stratify=y,
                                                    test_size=0.2, random_state=42)
```

훈련 데이터와 테스트 데이터의 비율을 8:2로 설정한다.

난수 초깃값 42로 지정하여 무작위로 데이터 세트를 섞은 다음 나눈다.

훈련 데이터를 나눌때 class의 비율을 동일하게 만든다.

8:2 나눈 비율 확인

```
print(x_train.shape, x_test.shape)
(455, 30) (114, 30)
```

data 비율 유지 확인

```
np.unique(y_train, return_counts=True)
(array([0, 1]), array([170, 285], dtype=int64))
```


이전 코드에서 바뀐 코드 설명

```
def __init__(self):  
    self.w = None  
    self.b = None
```

특성이 하나가 아니라 여러개가 될수 있으므로 __init__ 함수에서 초기화 하지 않고
실행중 x에 입력되는 특성의 개수에 따라 초기화 한다.

```
np.sum(x * self.w)
```

forpass함수의 해당 코드는 x가 더이상 하나의 값이 아니라 30개의 특성을 가지고 있는 넘파이 배열이므로
 $\text{np.sum}(x * \text{self.w}) \Rightarrow x[0]*w[0] + x[1]*w[1] + \dots + x[29]*w[29]$ 로 해석 된다.

```
self.w = np.ones(x.shape[1])
```

fit 함수의 w 초기화는 x.shape[1]의 값이 30 이므로 넘파이 배열의 개수를 30개로 할당 하고
ones 함수에 의해 각 원소는 1로 초기화 된다.

```
def activation(self, z):  
    a = 1 / (1 + np.exp(-z)) # 시그모이드 계산  
    return a
```

activation 함수를 구현하고 시그모이드 함수의 결과 값을 리턴한다.

이전 코드에서 바뀐 코드 설명

```
def predict(self, x):  
    z = [self.forpass(x_i) for x_i in x]      # 정방향 계산  
    a = self.activation(np.array(z))          # 활성화 함수  
    return a >= 0.5
```

적용
예측 함수를 구현한다.

`z = [self.forpass(x_i) for x_i in x]` 코드는 파이썬에서 지원하는 코드로 대괄호([]) 안에 `for in`과 함수의 호출부를 넣을수 있다. 이때 동작은 `x`의 각 원소를 `x_i`에 뽑아서 `x_i`를 `forpass`의 인자로 전달하여 `forpass`의 리턴값을 list로 append 하여 `x`의 가공된 값을 `z`에 대입한다.

`np.array(z)` 함수는 list를 넘파이 배열로 바꿔준다.

`a`에는 최종적으로 시그모이드 함수의 결과가 대입되므로

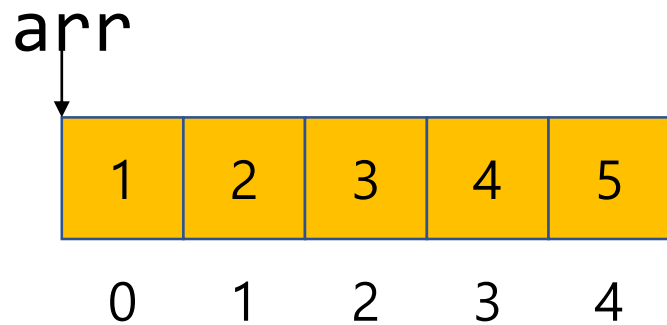
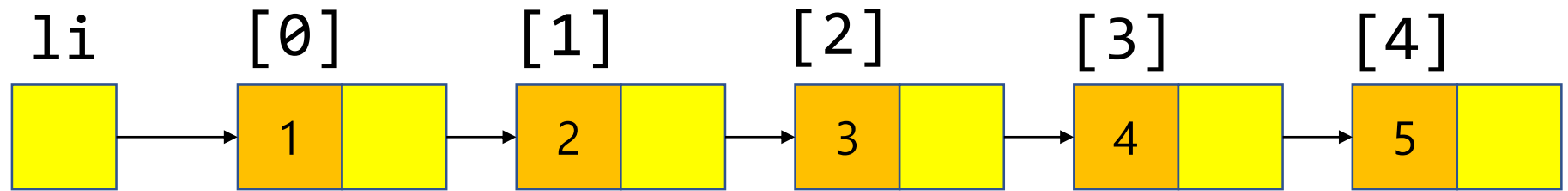
결과값이 0.5 보다 크다면 악성종양으로 판단하여 1을 리턴한다.

```
np.mean(neuron.predict(x_test) == y_test)  
0.8245614035087719
```

`np.mean` 함수는 평균을 계산하는 함수로 예측값과 실제값의 비교 값을 평균을 계산하여 정확도를 측정한다.

```
li = [1, 2, 3, 4, 5]  
print(type(li))
```

```
arr = np.array(li) # ndarray 로 변환  
print(type(arr))
```



```
arr = np.arange(1,7)
```

```
arr = np.reshape( arr, (2,3) )
```

0	1	2	3	4	5
1	2	3	4	5	6

arr

	0	1	2
0	1	2	3
1	4	5	6

```
arr = np.arange(1,7)
```

```
arr = np.reshape( arr, (3,2) )
```

0	1	2	3	4	5
1	2	3	4	5	6

	0	1
0	1	2
1	3	4
2	5	6

```
arr = np.arange(1,7)
```

```
np.shape # ( 6, )
```

	0	1	2	3	4	5
arr	1	2	3	4	5	6

```
arr = np.reshape(arr, (-1,1) )
```

	0
0	1
1	2
2	3
3	4
4	5
5	6

```
arr.shape # ( 6, 1 )
```

```
arr = np.arange(1,7)
arr = np.reshape( arr, (-1,3) )
arr
```

arr

	0	1	2
0	1	2	3
1	4	5	6

```
arr = np.reshape( arr, (-1,2) )
arr
```

arr

	0	1
0	1	2
1	3	4
2	5	6

arr.T

arr

	0	1
0	1	4
1	2	5
2	3	6

```
a = np.array([1,2,3])
```

```
b = 10
```

```
c = a+b
```

```
c
```

	0	1	2
a	1	2	3

b	10
---	----

	0	1	2
	1	2	3

+

	10	10	10
--	----	----	----

c	11	12	13
---	----	----	----


```
a = np.array([[1,2],[3,4]])
```

```
b = np.array([10,20])
```

```
c = a+b
```

```
c
```

a

	0	1
0	1	2
1	3	4

1	2
3	4

+

b

10	20
----	----

10	20
10	20

11	22
13	24

```
arr2 = np.array( [[1,2,3,4],  
                  [5,6,7,8],  
                  [9,10,11,12]])
```

arr2

	0	1	2	3
arr2				
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
arr2.shape  
(3,4)
```

```
arr2.ndim  
2
```

```
arr2 = np.array( [[1,2,3,4],  
                  [5,6,7,8],  
                  [9,10,11,12]])
```

arr2

arr2

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
arr2.shape  
(3,4)
```

```
arr2.ndim  
2
```

`arr2[:, 2]`

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

`(3,)`

3	7	11
---	---	----

`arr2[:, 1:3]`

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

`(3, 2)`

2	3
6	7
10	11

arr2[:, [1,3]]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

(3,2)

2	4
6	8
10	12

arr2[[0,2] , :]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

(2,4)

1	2	3	4
9	10	11	12

arr2[[0,2] , 1]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

(2,)

2	10
---	----

arr2[[0,2] , 1:3]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

(2,2)

2	3
10	11

arr2[[0,2] , [1,3]]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

(2,)

2	12
---	----

```
index = np.array([0, 2])  
arr2[ index , :]
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

(2,4)

1	2	3	4
9	10	11	12

```
names = np.array([ 'Kim', 'Lee', 'Park' , 'Choi'])  
names_mask = ( names == 'Lee' )
```

	0	1	2	3
names	'Kim'	'Lee'	'Park'	'Choi'
	==	==	==	==
	'Lee'	'Lee'	'Lee'	'Lee'
names_mask	False	True	False	False

```
data = np.arange(16).reshape((4,4))
names = np.array([ 'Kim', 'Lee', 'Park' , 'Choi'])
names_mask = ( names == 'Lee' )
data[:, names_mask ]
```

	0	1	2	3	(4,1)
0	0	1	2	3	1
1	4	5	6	7	5
2	8	9	10	11	9
3	12	13	14	15	13

```
data[ data[:, 0] > 4, : ]
```

False	False	True	True
-------	-------	------	------

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

(2,4)

8	9	10	11
12	13	14	15

`np.sum(data, axis=1)`
열간 합

axis 0 1
(4,4)

	0	1	2	3	
0	0	1	2	3	6
1	4	5	6	7	22
2	8	9	10	11	38
3	12	13	14	15	54

(4,)

6	22	38	54
---	----	----	----

`np.sum(data, axis=0)`
행간 합

axis 0 1
(4,4)

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
	24	28	32	36

(4,)

24	28	32	36
----	----	----	----


```
a = np.array([[1,2],  
              [3,4]])
```

```
b = np.array([[2,2],  
              [2,2]])
```

`np.dot(a,b)`

$$1*2 + 2*2 = 6$$

1	2
3	4

 •

2	2
2	2

 =

6	6
14	14

$$3*2 + 4*2 = 14$$

1	2
3	4

 •

2	2
2	2

 =

6	6
14	14

$$1*2 + 2*2 = 6$$

1	2
3	4

 •

2	2
2	2

 =

6	6
14	14

$$3*2 + 4*2 = 14$$

1	2
3	4

 •

2	2
2	2

 =

6	6
14	14

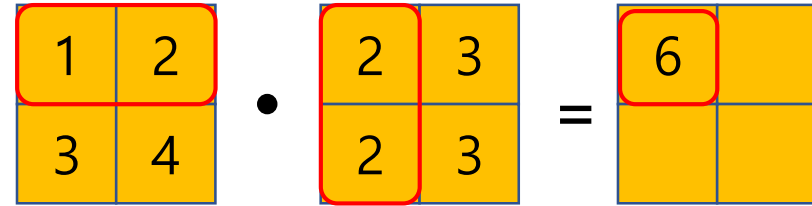
a.shape => (2,2)
a = np.array([[1,2],
 [3,4]])

b.shape => (2,2)
b = np.array([[2,3],
 [2,3]])

(2,**2**)(**2**,2)

np.dot(a,b)

$$1*2 + 2*2 = 6$$



1	2
3	4

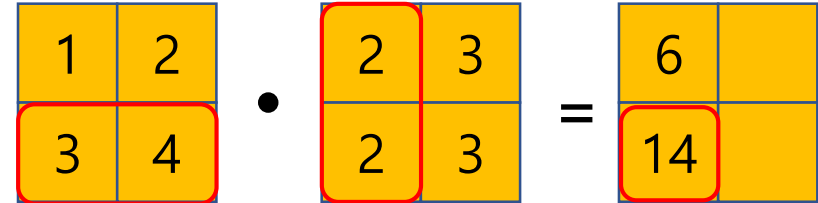
 ·

2	3
2	3

 =

6	

$$3*2 + 4*2 = 14$$



1	2
3	4

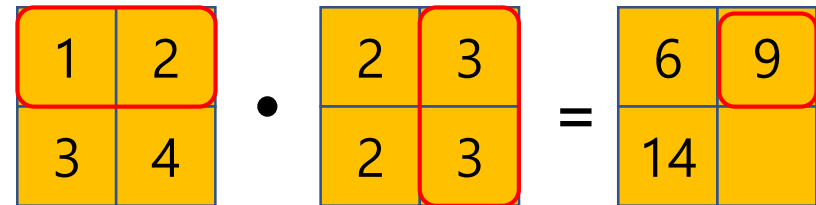
 ·

2	3
2	3

 =

6	
14	

$$1*3 + 2*3 = 9$$



1	2
3	4

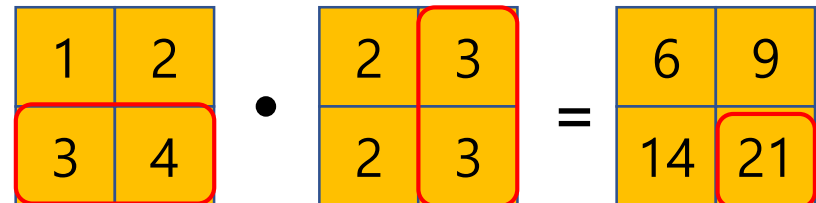
 ·

2	3
2	3

 =

6	9
14	

$$3*3 + 4*3 = 21$$



1	2
3	4

 ·

2	3
2	3

 =

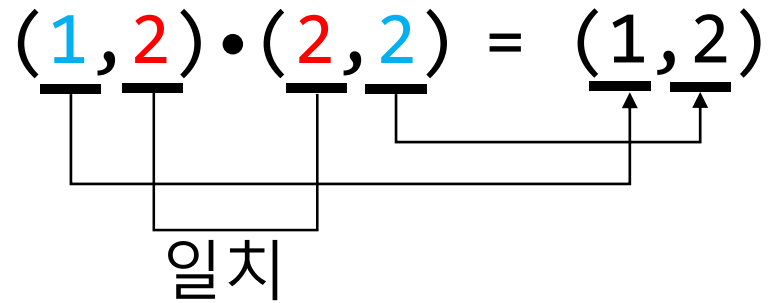
6	9
14	21

a.shape => (1,2)
a = np.array([[1,2]])

b.shape => (2,2)
b = np.array([[2,3],
[2,3]])

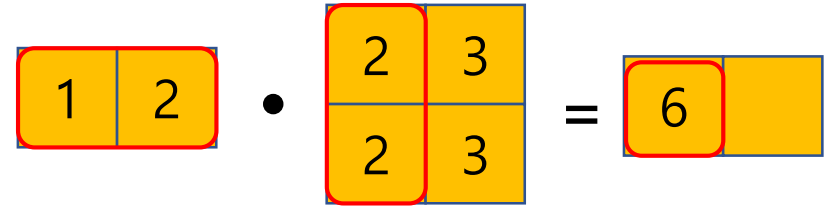
$$(\underline{1}, \underline{2}) \cdot (\underline{2}, \underline{2}) = (\underline{1}, \underline{2})$$

일치

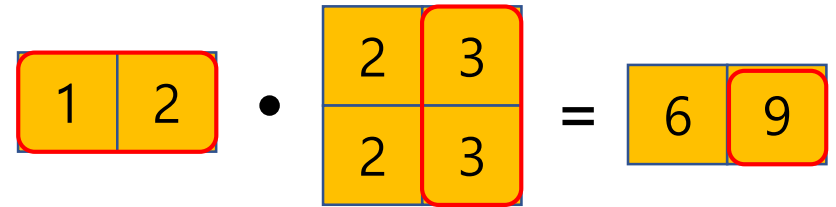


np.dot(a,b)

$$1*2 + 2*2 = 6$$



$$1*3 + 2*3 = 9$$



`np.dot(b, a)`

```
a = np.array([[1,2],  
              [3,4]])
```

```
b = np.array([[2,3],  
              [2,3]])
```

$$2*1 + 3*3 = 1$$

2	3
2	3

 •

1	2
3	4

 =

11	16
11	16

$(\underline{2}, \underline{2}) \cdot (\underline{2}, \underline{2}) = (\underline{2}, \underline{2})$

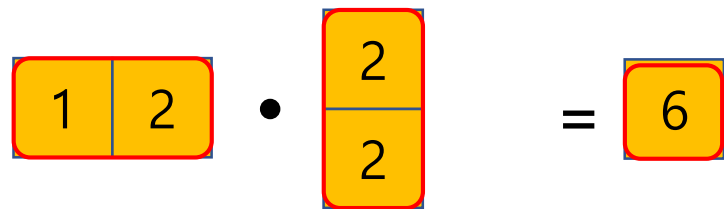
일치

`np.dot(a, b)`

`a = np.array([[1, 2]])`

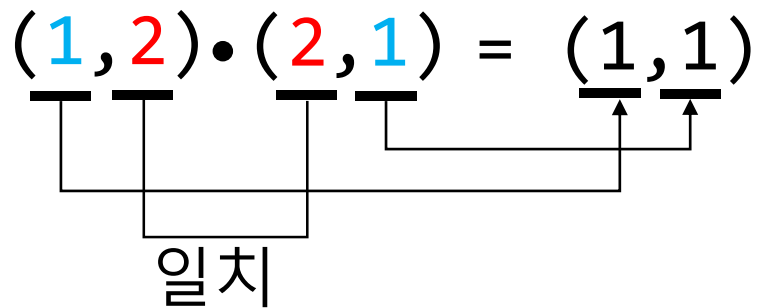
`b = np.array([[2],
[2]])`

$$1*2 + 2*2 = 6$$



A diagram illustrating the dot product of two vectors. On the left, a horizontal yellow box with a red border is divided into two cells containing the numbers 1 and 2. To its right is a black dot representing multiplication. Next is a vertical yellow box with a red border, divided into two cells, each containing the number 2. To the right of this is an equals sign, followed by a single yellow box with a red border containing the number 6.

$(\underline{1}, \underline{2}) \cdot (\underline{2}, \underline{1}) = (\underline{1}, \underline{1})$



A diagram illustrating the dot product of two vectors. The first vector is (1, 2) and the second is (2, 1). The result is (1, 1). Lines connect the first element of the first vector (1) to the first element of the result (1), and the second element of the first vector (2) to the second element of the result (1). A bracket labeled "일치" (match) is placed under the first two elements of the first vector, indicating that the first two elements of the first vector match the first two elements of the second vector.

`np.dot(b, a)`

`a = np.array([[1, 2]])`

`b = np.array([[2],
[2]])`

$$\begin{array}{c} \textcolor{blue}{(2, 1)} \cdot \textcolor{red}{(1, 2)} = \textcolor{black}{(2, 2)} \\ \begin{array}{ccccccc} \text{---} & \text{---} & \text{---} & \text{---} & & \text{---} & \text{---} \\ | & | & | & | & & | & | \\ \text{---} & \text{---} & \text{---} & \text{---} & & \text{---} & \text{---} \\ & & & & & \uparrow & \uparrow \\ & & & & & \text{---} & \text{---} \\ & & & & & | & | \\ & & & & & \text{---} & \text{---} \end{array} \\ \text{일치} \end{array}$$

$$\begin{array}{|c|} \hline 2 \\ \hline 2 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 2 & \\ \hline & \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 2 \\ \hline 2 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 2 & \\ \hline 2 & \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 2 \\ \hline 2 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 2 & 4 \\ \hline 2 & \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 2 \\ \hline 2 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 2 & 4 \\ \hline 2 & 4 \\ \hline \end{array}$$

$$x_1w_1 + x_2w_2 + b$$

```
x = np.array([1,2])
```

```
w = np.array([2,3])
```

```
b = 1
```

```
h = np.sum(x*w) + b
```

$$x_1w_1 + x_2w_2 + b$$

```
x = np.array([1,2])
```

```
w = np.array([2,3])
```

```
b = 1
```

```
h = np.dot(x,w) + b
```


$$h^0 = x_1^0 w_1 + x_2^0 w_2 + b$$

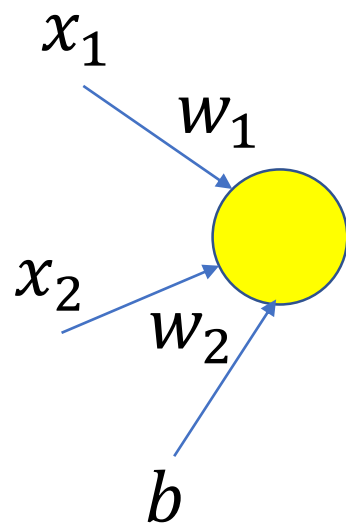
$$h^1 = x_1^1 w_1 + x_2^1 w_2 + b$$

```
x = np.array([1,2],
              [4,5])
```

```
w = np.array([[2,3]])
```

```
b = 1
```

```
h = np.dot(x,w.T) + b
```



1	2
4	5

•

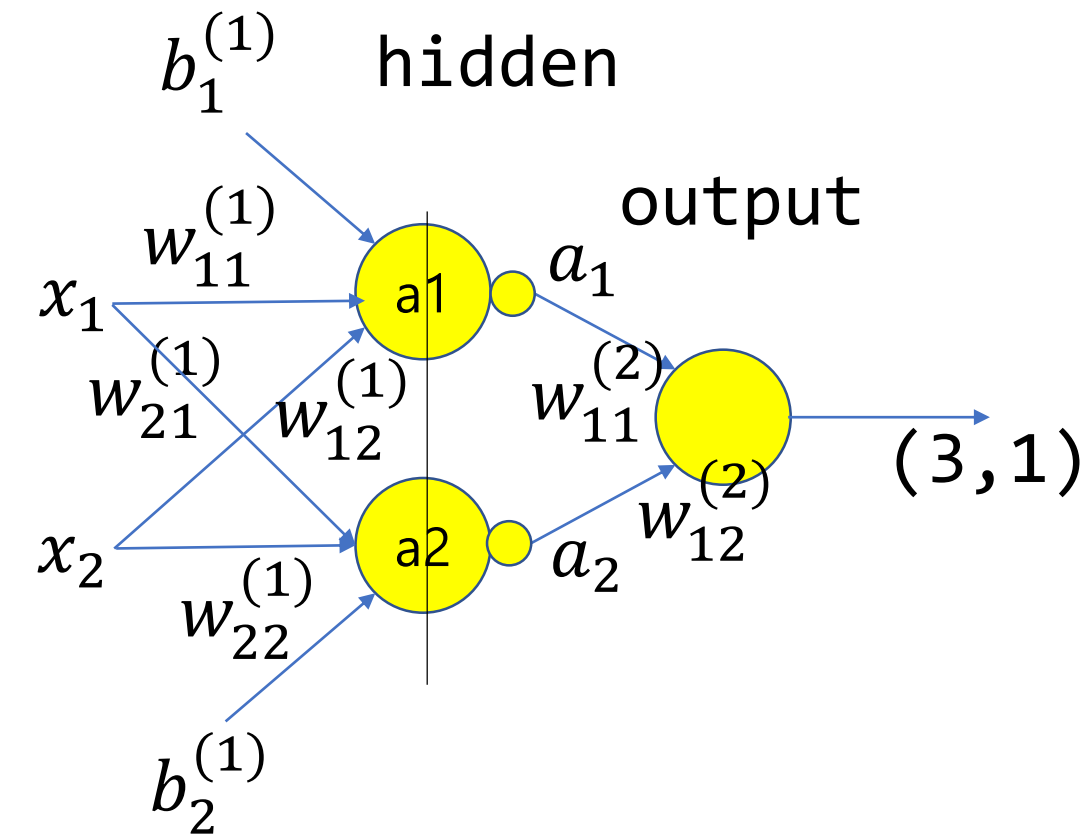
2
3

$$(\underline{2}, \underline{2}) \cdot (\underline{2}, \underline{1}) = (\underline{2}, \underline{1})$$

일치

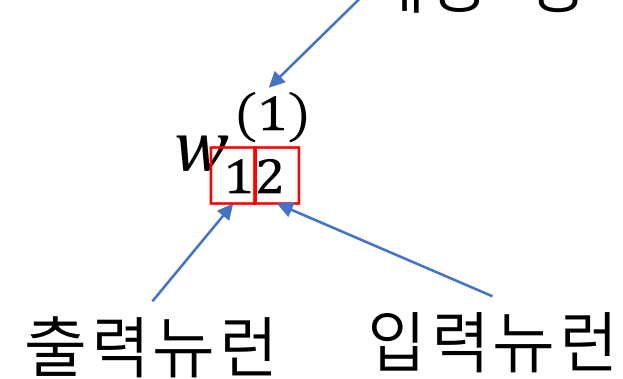
$$z_1 = x_1^{(1)} w_{11}^{(1)} + x_2^{(1)} w_{12}^{(1)} + b_1^{(1)}$$

$$z_2 = x_1^{(1)} w_{21}^{(1)} + x_2^{(1)} w_{22}^{(1)} + b_2^{(1)}$$

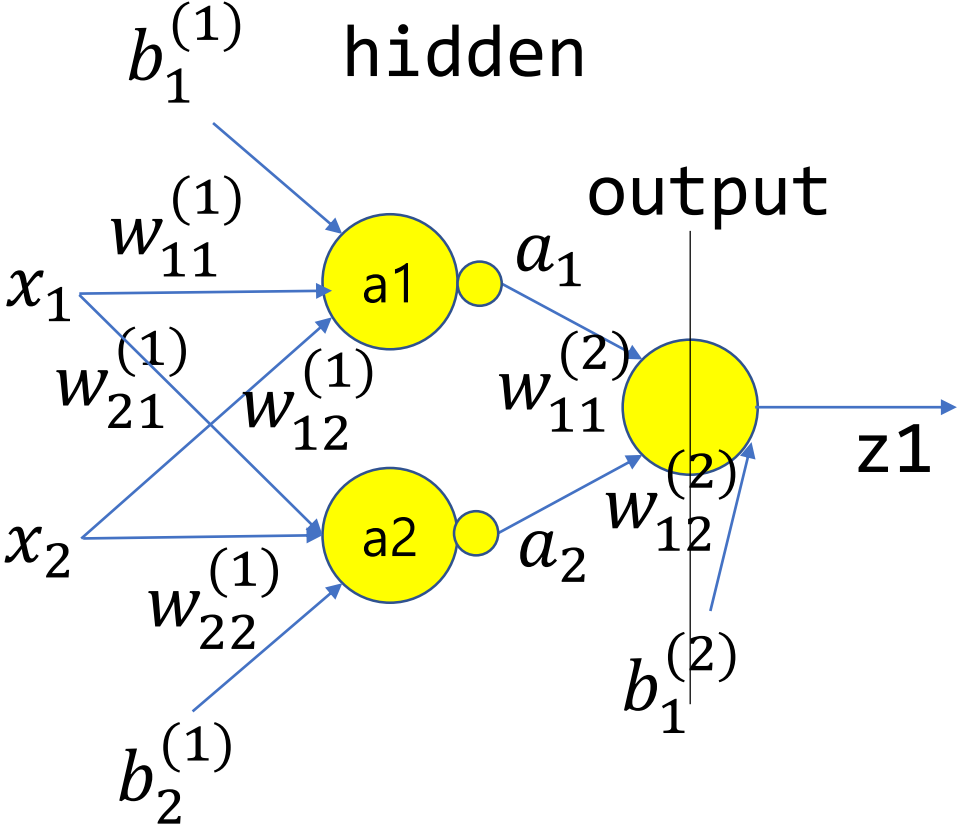
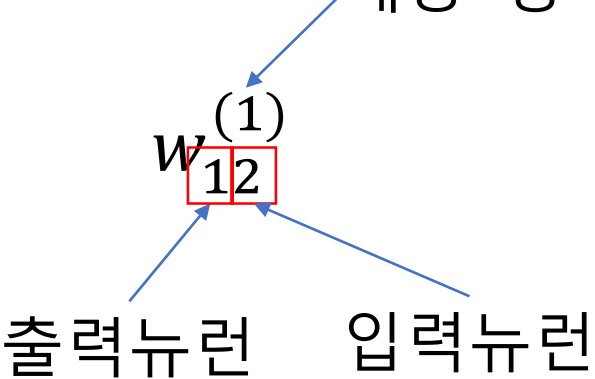


$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \end{bmatrix} \cdot \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} & b_2^{(1)} \end{bmatrix} =$$

$$X \cdot W + B = Z$$



$$z_1 = a_1^{(2)} w_{11}^{(2)} + a_2^{(2)} w_{12}^{(2)} + b_1^{(2)}$$



$$\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_1^{(3)} \\ a_2^{(3)} \end{bmatrix} \cdot \begin{bmatrix} w_{11}^{(2)} \\ w_{12}^{(2)} \end{bmatrix} + b_1^{(2)} = \begin{bmatrix} z_1^{(1)} \\ z_1^{(2)} \\ z_1^{(3)} \end{bmatrix}$$

$$A \cdot W + B = Z$$

고차원 배열 사용 법

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

[1]

[0]

		5
1	2	
3	4	6

a

1	2
3	4

a.T

1	3
2	4

x.T

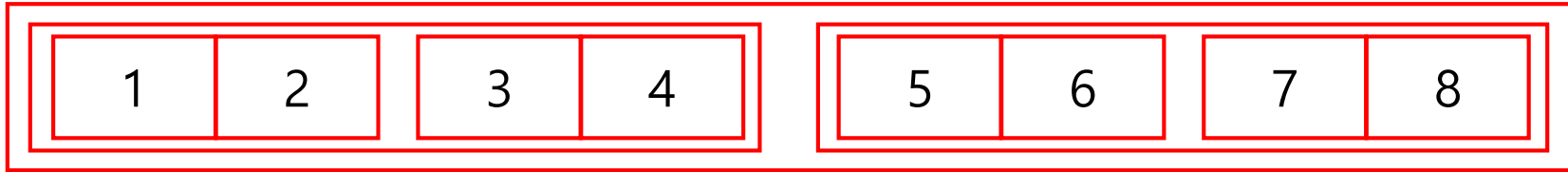
[1]

[0]

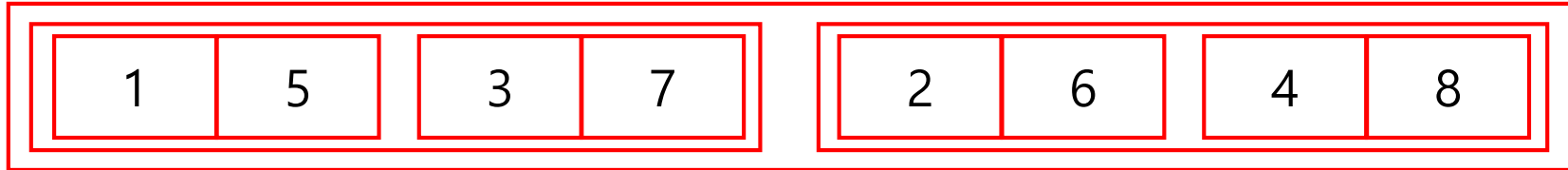
		5
1	5	
3	7	6

고차원 배열 사용 법

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

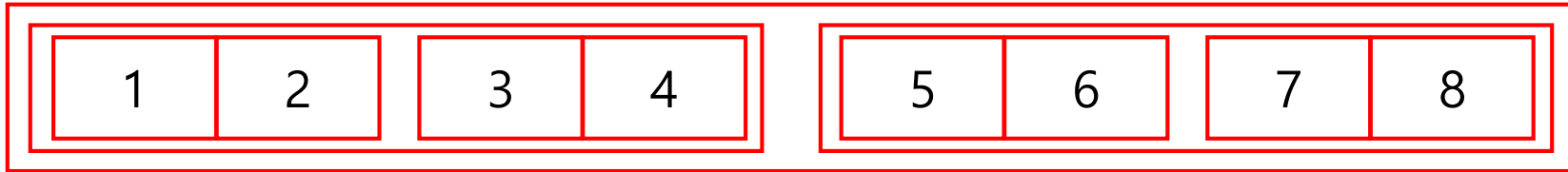


```
x.T      np.swapaxes(x, 0, 2)
```

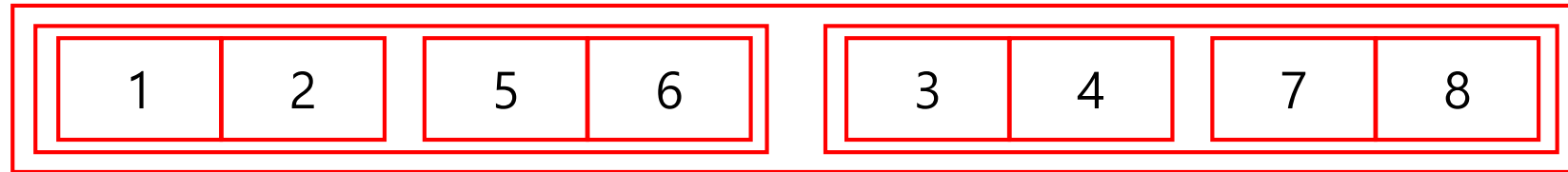


고차원 배열 사용 법

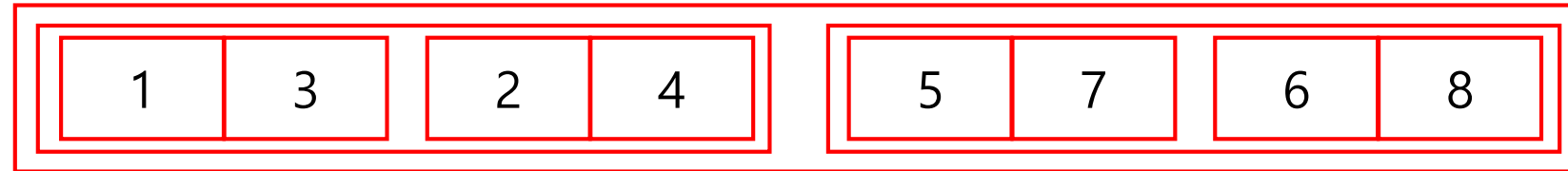
```
x = np.array([ [[1,2],[3,4]] , [[5,6],[7,8]] ])
```



```
np.swapaxes(x, 0, 1)
```

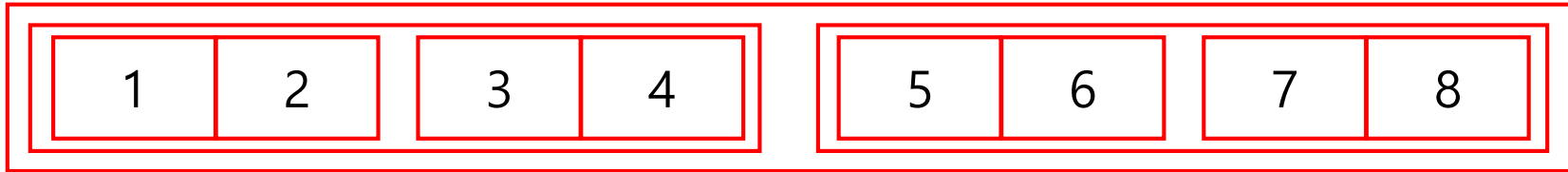


```
np.swapaxes(x, 1, 2)
```

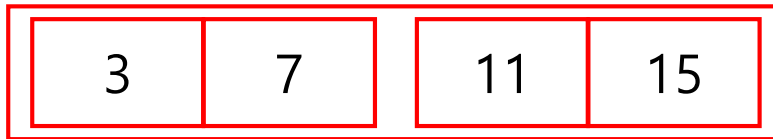


고차원 배열 사용 법

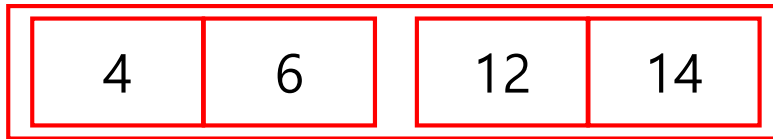
```
x = np.array([ [1,2],[3,4]] , [[5,6],[7,8]] ])
```



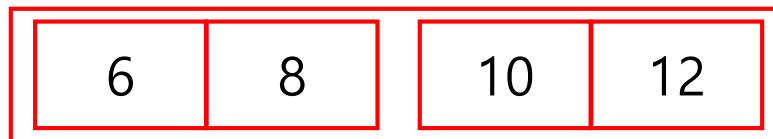
```
np.sum(x, axis=2)
```



```
np.sum(x, axis=1)
```

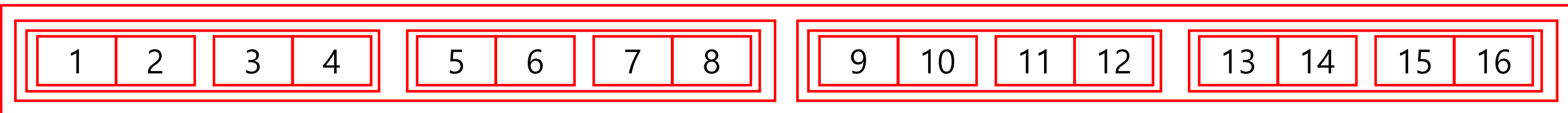


```
np.sum(x, axis=0)
```

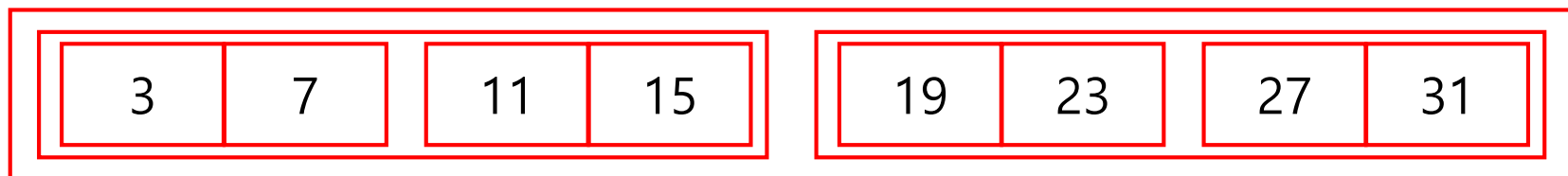


고차원 배열 사용 법

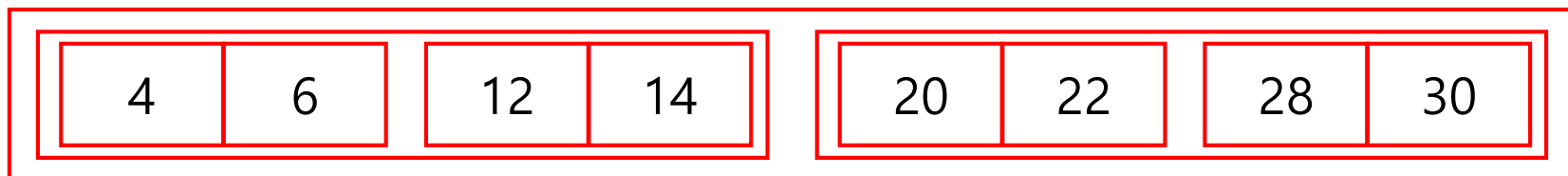
```
x = np.array([[[[1,2],[3,4]],[[5,6],[7,8]]],[[[9,10],[11,12]],[[13,14],[15,16]]])
```



```
np.sum(x, axis=3)
```

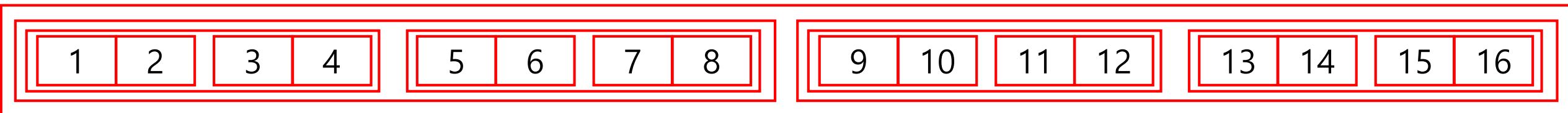


```
np.sum(x, axis=2)
```

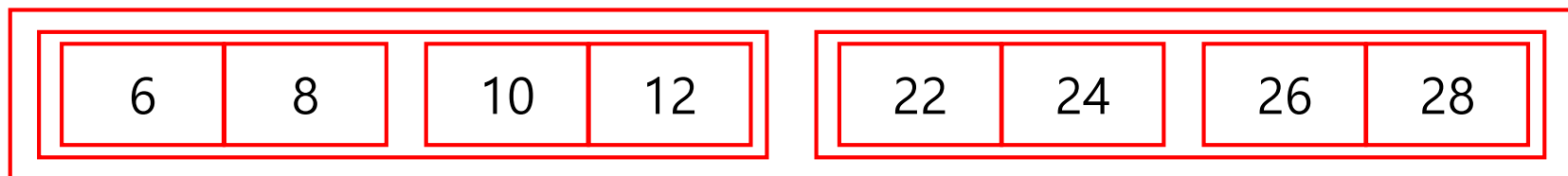


고차원 배열 사용 법

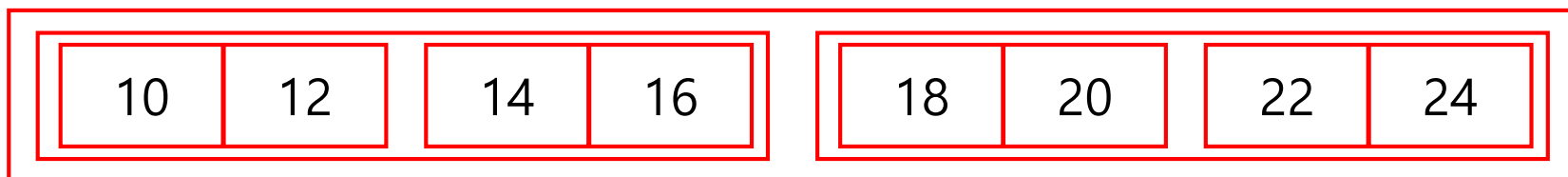
```
x = np.array([[[[1,2],[3,4]],[[5,6],[7,8]]],[[[9,10],[11,12]],[[13,14],[15,16]]])
```



```
np.sum(x, axis=1)
```

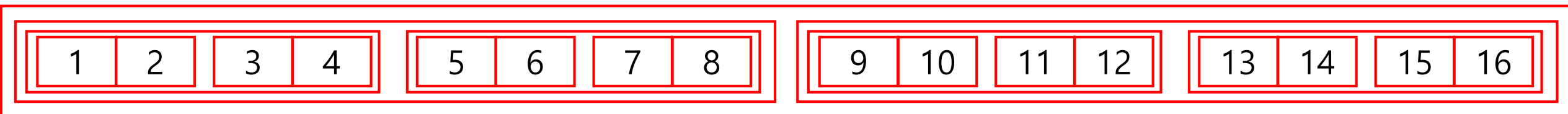


```
np.sum(x, axis=0)
```

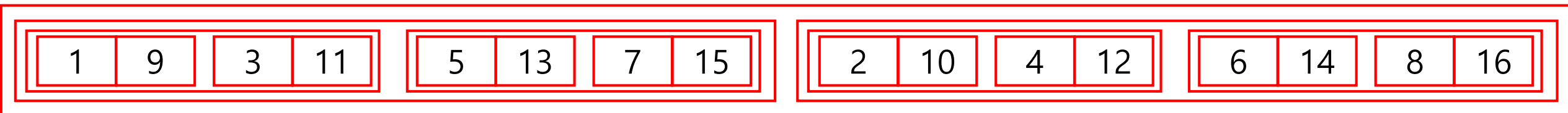


고차원 배열 사용 법

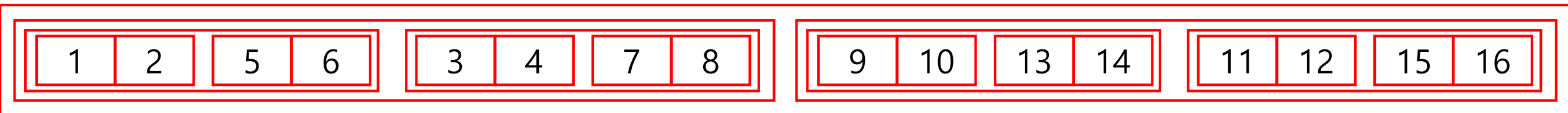
```
x = np.array([[[[1,2],[3,4]],[[5,6],[7,8]]],[[[9,10],[11,12]],[[13,14],[15,16]]])
```



```
np.swapaxes(x, 0, 3)
```



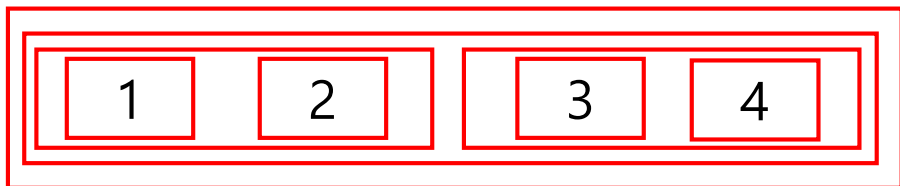
```
np.swapaxes(x, 1, 2)
```



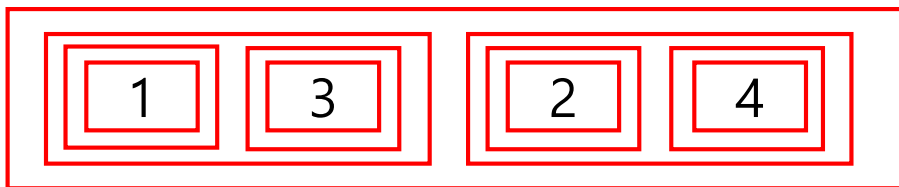
고차원 배열 사용 법

`x.shape => (1,2,2,1)`

`x = np.array([[[[1],[2]], [[3],[4]]]])`



`np.swapaxes(x, 0, 2) => (2,2,1,1)`

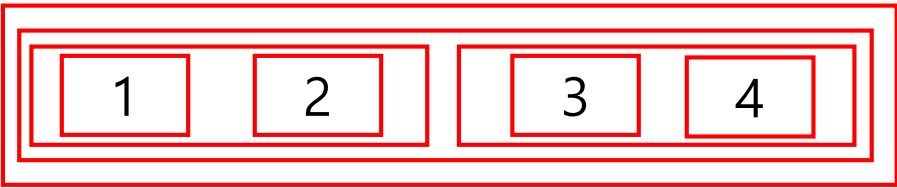


고차원 배열 사용 법

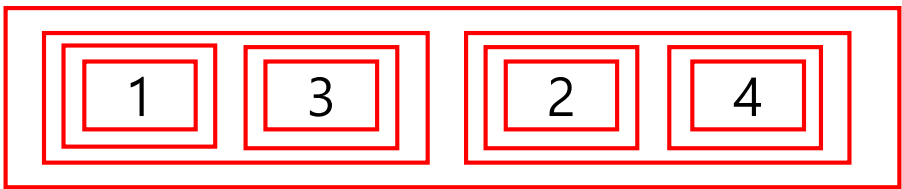
`x.shape => (1,2,2,1)`

`(2,2,3,3)`

`x = np.array([[[[1],[2]], [[3],[4]]]])`



`np.swapaxes(x, 0, 2) => (2,2,1,1)`



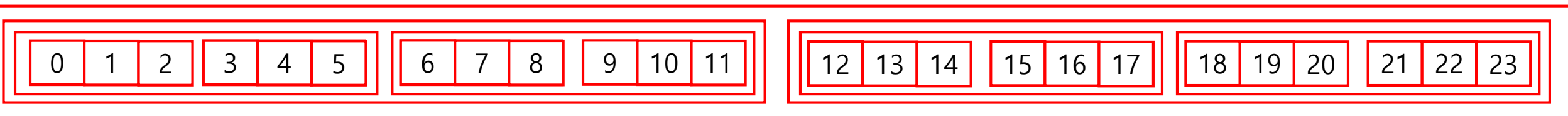
고차원 배열 사용 법

```
x = np.arange(24).reshape( (2,2,2,3) )
```

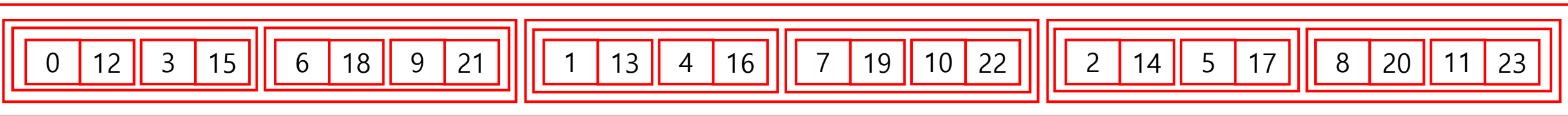
```
np.swapaxes(x, 0, 3) => (3,2,2,2)
```

고차원 배열 사용 법

```
x = np.arange(24).reshape( (2,2,2,3) )
```



```
np.swapaxes(x, 0, 3) => (3,2,2,2)
```



X

[illegible]

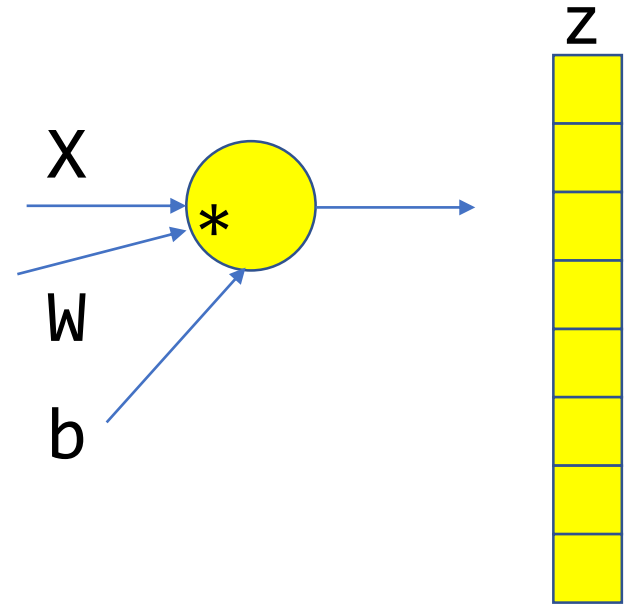
m : 미니배치 사이즈

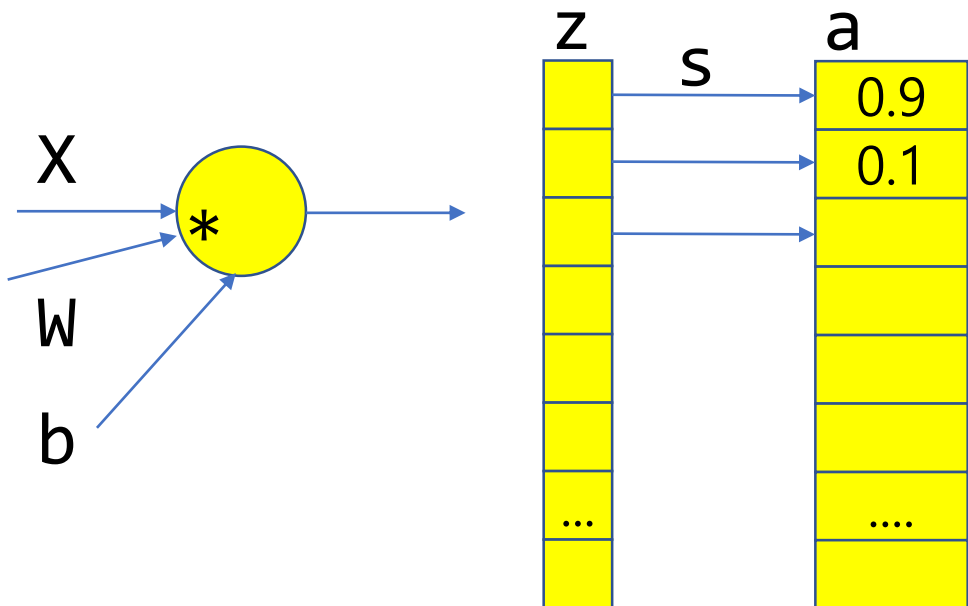
(미니배치 사이즈, x 의 특성수) (x 의 특성수 , 출력 뉴런수)

m : 364

$$(364, 30)(30, 1) = (8, 1)$$

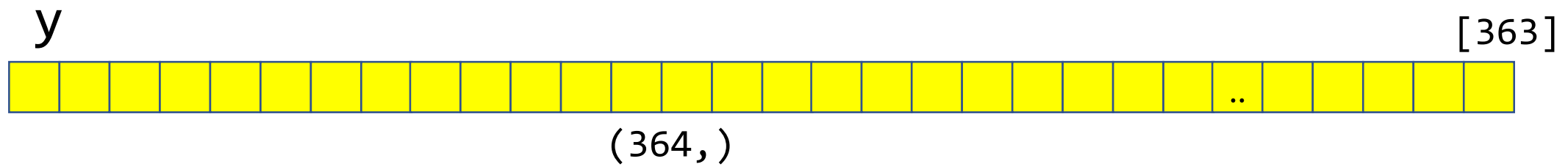
```
z = np.dot(x, self.w) + self.b
```





$$\text{sigmoid} = \frac{1}{1 + e^{-z}}$$

$$a = 1 / (1 + \text{np.exp}(-z))$$



```
def fit(self, x, y, epochs=100, x_val=None, y_val=None):  
    y = y.reshape(-1, 1) # 타깃을 열 벡터로 바꿉니다.  
    y_val = y_val.reshape(-1, 1)  
    m = len(x) # 샘플 개수를 저장합니다.  
    self.w = np.ones((x.shape[1], 1)) # 가중치를 초기화합니다.  
    self.b = 0 # 절편을 초기화합니다.
```

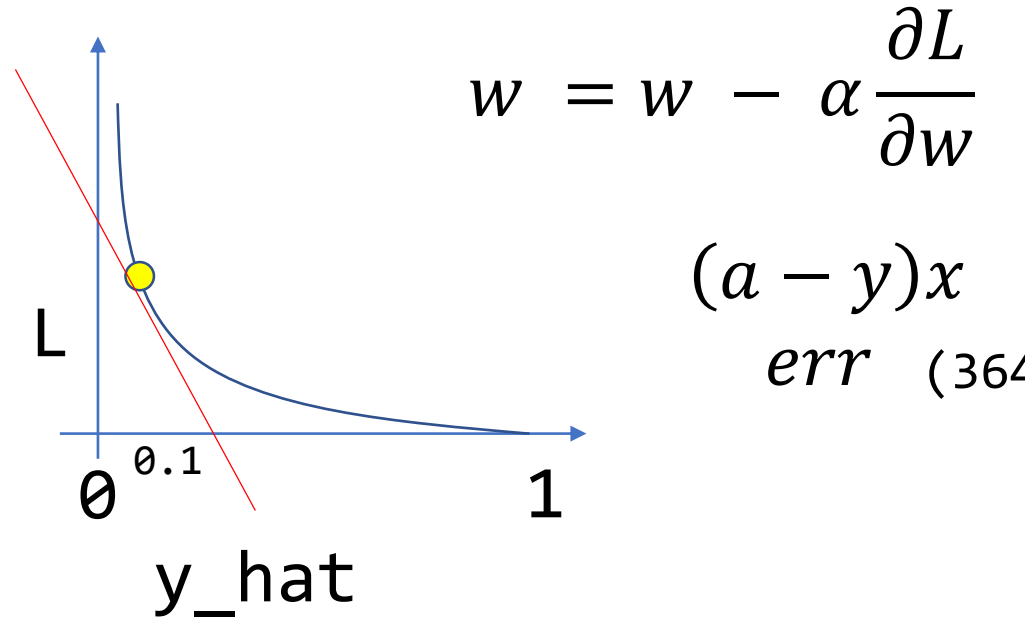
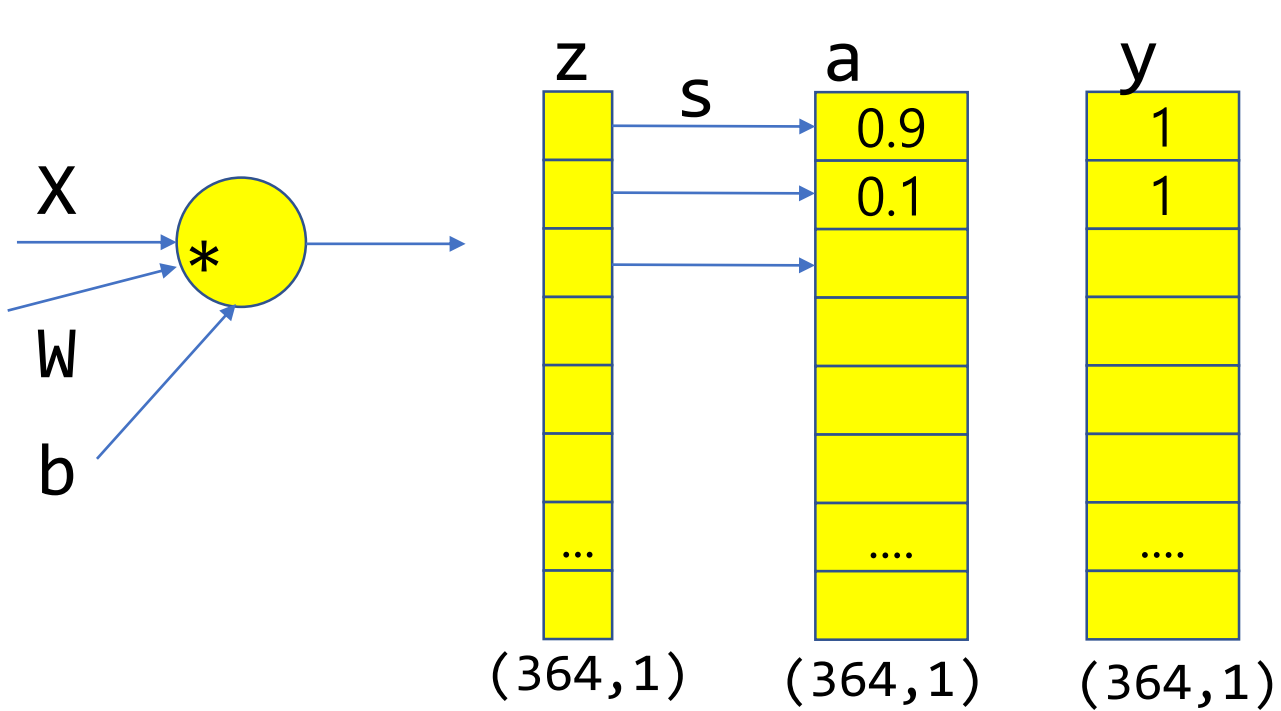
`x.shape => (364, 30)`

w
(30,1)



y
(364,1)





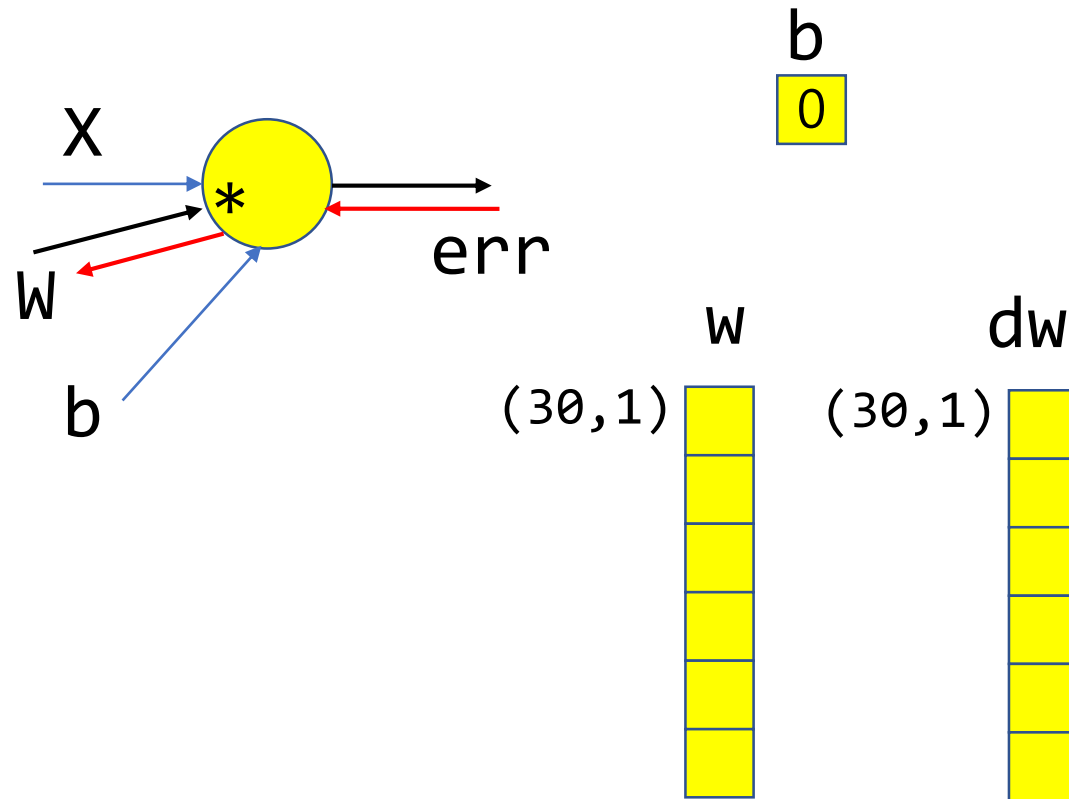
$$w = w - \alpha \frac{\partial L}{\partial w}$$

$$(a - y)x$$

err (364)

```
def backprop(self, x, err):
    m = len(x)
    w_grad = np.dot(x.T, err) / m
    b_grad = np.sum(err) / m
    return w_grad, b_grad
```

행렬의 곱의 미분에서는
뒤에서 전달된 미분값을
사라진 자신의 자리에 쓴다.
남아 있는 값은 전치 한다.



$$Z = X \cdot W + b$$

(30,364) (364,1)

$X.T$ err

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w}$$

$$w = w - \alpha \frac{\partial}{\partial w}$$

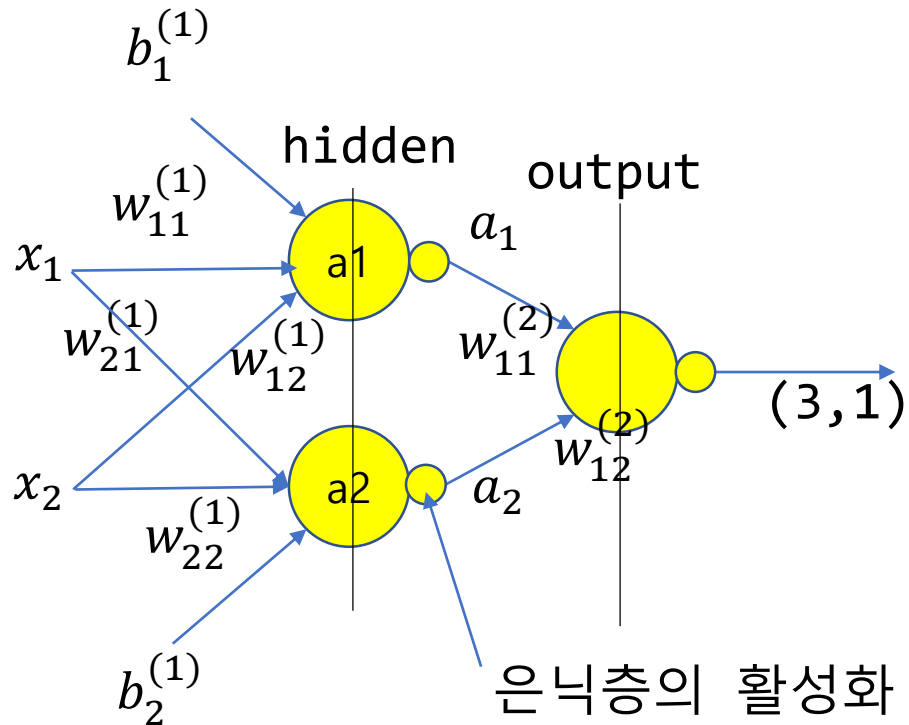
$$(a - y)$$

err (36

```
def forpass(self, x):
    z1 = np.dot(x, self.w1) + self.b1      # 첫 번째 층의 선형 식을 계산합니다
    self.a1 = self.activation(z1)          # 활성화 함수를 적용합니다
    z2 = np.dot(self.a1, self.w2) + self.b2 # 두 번째 층의 선형 식을 계산합니다.
    return z2
```

$$\begin{matrix} (364, 30) & (30, 2) & + & (2,) & = & (364, 2) \\ x & w1 & & b & & z1 \end{matrix}$$

$$\begin{matrix} (364, 2) \\ a1 \end{matrix}$$



$$\begin{matrix} (364, 2) & (2, 1) & + & b & = & (364, 1) \\ a1 & w2 & & & & z2 \end{matrix}$$

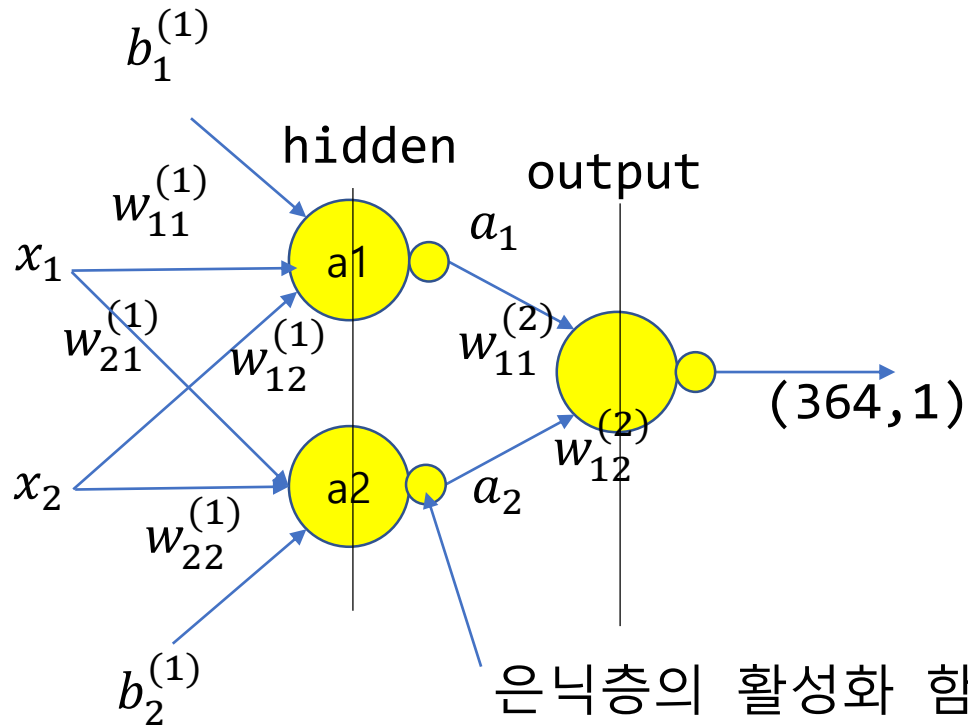
은닉층의 활성화 함수는 반드시 비선형 함수를 사용 해야한다.
ex) sigmoid

```
def backprop(self, x, err):
    m = len(x)
    w2_grad = np.dot(self.a1.T, err) / m
    b2_grad = np.sum(err) / m
```

$$Z2 = A1 \cdot W2 + b$$

$$err \quad \frac{\partial L}{\partial W_2} = A1^T \cdot err$$

$$\begin{matrix} (2, 364) & (364, 1) & = & (2, 1) \\ a1.T & err & & w2_grad \end{matrix}$$



$$\begin{matrix} (364, 2) & (2, 1) & + & b & = & (364, 1) \\ a1 & w2 & & & & z2 \end{matrix}$$

은닉층의 활성화 함수는 반드시 비선형 함수를 사용 해야한다.
ex) sigmoid

```
def backprop(self, x, err):
```

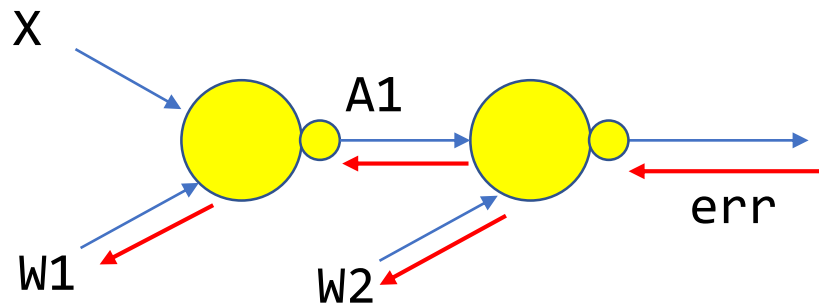
```
    err_to_hidden = np.dot(err, self.w2.T) * self.a1 * (1 - self.a1)
```

```
    w1_grad = np.dot(x.T, err_to_hidden) / m
```

```
    b1_grad = np.sum(err_to_hidden, axis=0) / m
```

sigmoid의 미분식
 $a*(1-a)$

$(364, 30)(30, 2) = (364, 2)$



$$Z2 = A1 \cdot W2 + b2$$

$$\frac{\partial L}{\partial A_1} = err \cdot W_2^T$$

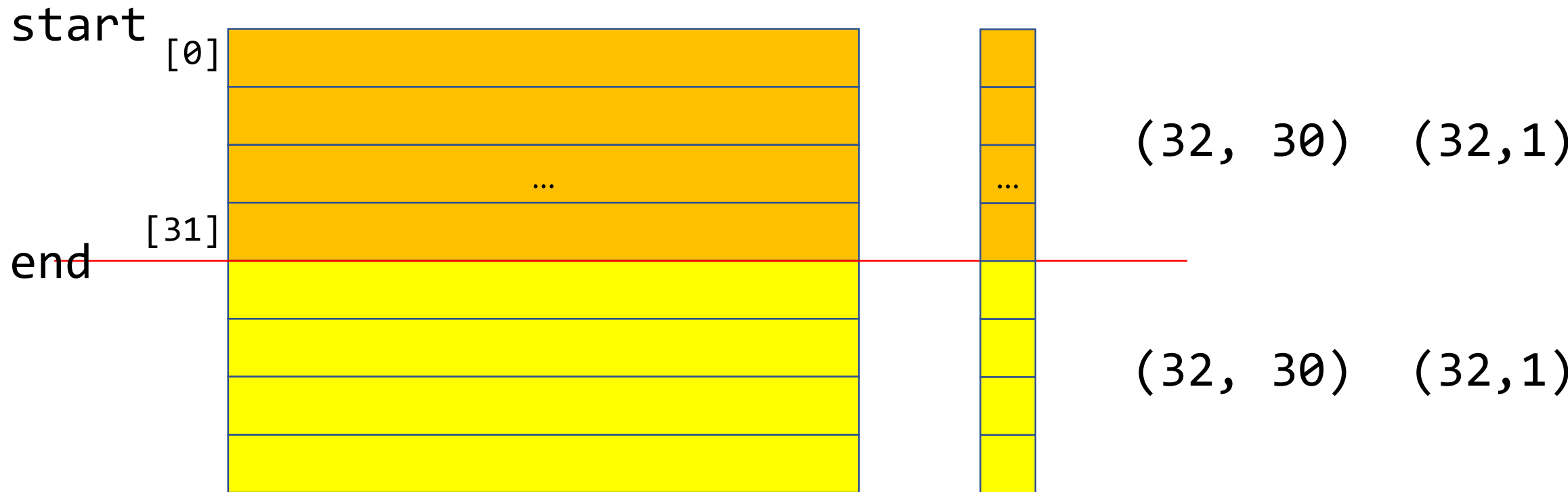
$$(364, 1)(1, 2) = (364, 2)$$

$$Z1 = X \cdot W1 + b1$$

$$\frac{\partial L}{\partial W_1} = X^T \cdot err_to_hidden$$

$$(30, 364)(364, 2) = (30, 2)$$

```
for i in range(bins):
    start = self.batch_size * i
    end = self.batch_size * (i + 1)
    yield x[start:end], y[start:end]
```



```

import numpy as np
temp = np.array([[2.20,1.39,0.85],
                 [0.00,-1.39,-2.20]])
exp_temp = np.exp(temp)
ret = exp_temp / np.sum(exp_temp, axis=1).reshape(-1, 1)
print( ret )

```

2.20, 1.39, 0.85

2.20,1.39,0.85

50% , 31%, 19%

$$\begin{array}{ccc}
 \frac{e^{2.20}}{e^{2.20} + e^{1.39} + e^{0.85}} & \frac{e^{1.39}}{e^{2.20} + e^{1.39} + e^{0.85}} & \frac{e^{0.85}}{e^{2.20} + e^{1.39} + e^{0.85}} \\
 58\% & 26\% & 15\%
 \end{array}$$


```

import numpy as np
temp = np.array([[2.20,1.39,0.85],
                 [0.00,-1.39,-2.20]])
exp_temp = np.exp(temp)
ret = exp_temp / np.sum(exp_temp, axis=1).reshape(-1, 1)
print( ret )

```

0.00, -1.39, -2.20

$$\begin{array}{ccc}
 \frac{e^{0.00}}{e^{0.00} + e^{-1.39} + e^{-2.20}} & \frac{e^{-1.39}}{e^{0.00} + e^{-1.39} + e^{-2.20}} & \frac{e^{-2.20}}{e^{0.00} + e^{-1.39} + e^{-2.20}} \\
 74\% & 18\% & 8\%
 \end{array}$$

다음 주소에서 vc_redist.x64.exe를 다운로드 받아 설치하면 해결됩니다.

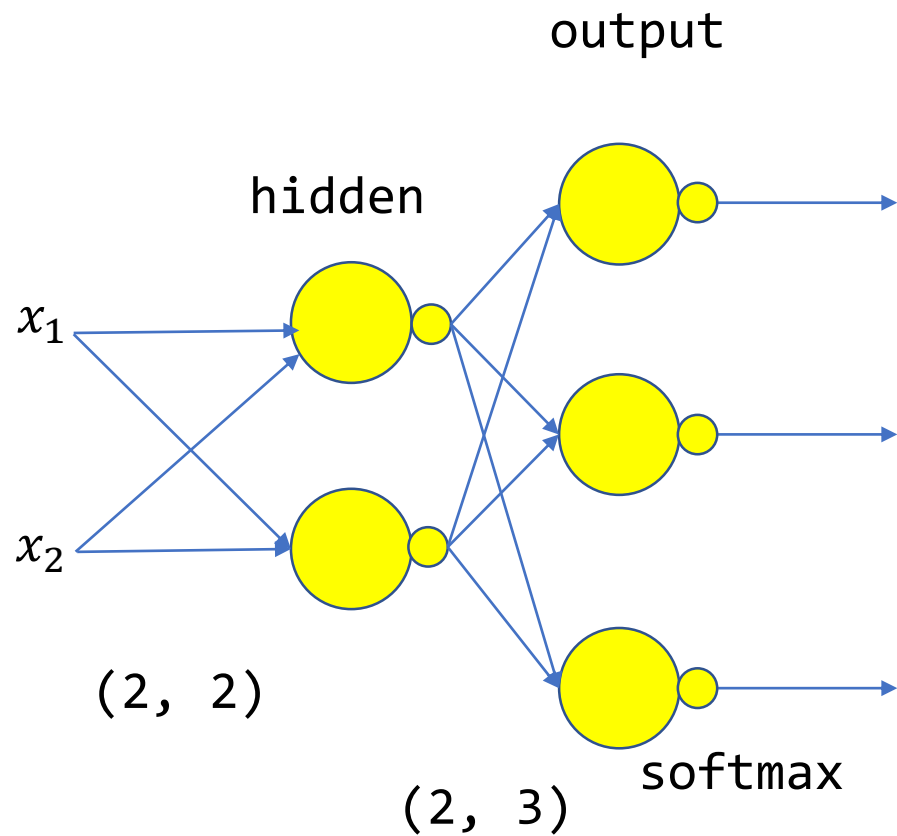
<https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c>

```
import numpy as np
temp = np.array([1,1,2,2,3,3,4,4])
np.bincount(temp)
```

```
[0, 2, 2, 2]
```

	0	1	2	3	4	5	6	7
temp	1	1	2	2	3	3	4	4

	0	1	2	3	4
	0	2	2	2	2



0.7

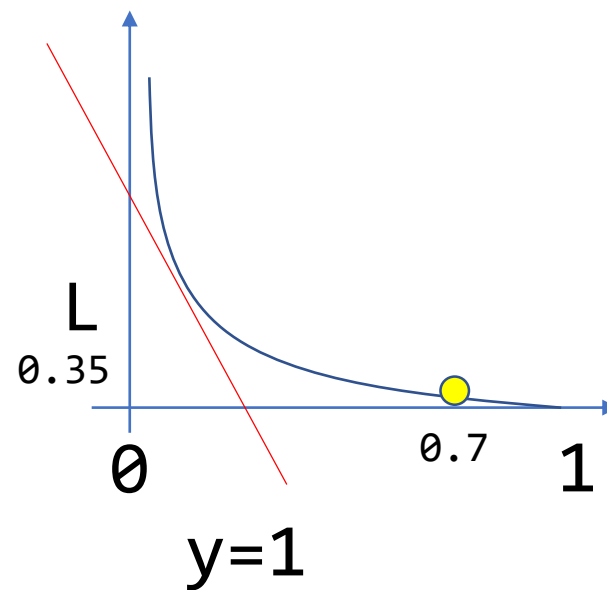
0.2

0.1

0.7 0.2 0.1

1 0 0

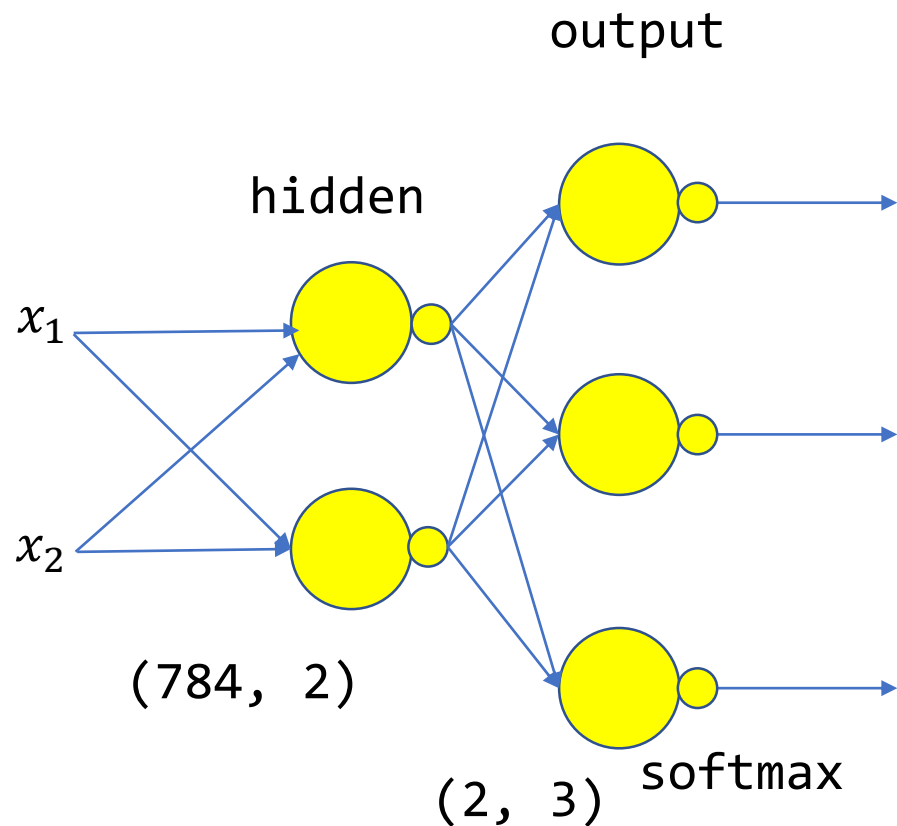
$y=[1,0,0]$



$$-(y * \text{np.log}(a) + (1 - y) * \text{np.log}(1 - a))$$

$$\text{np.sum}(-y * \text{np.log}(a))$$

0.35



0.7

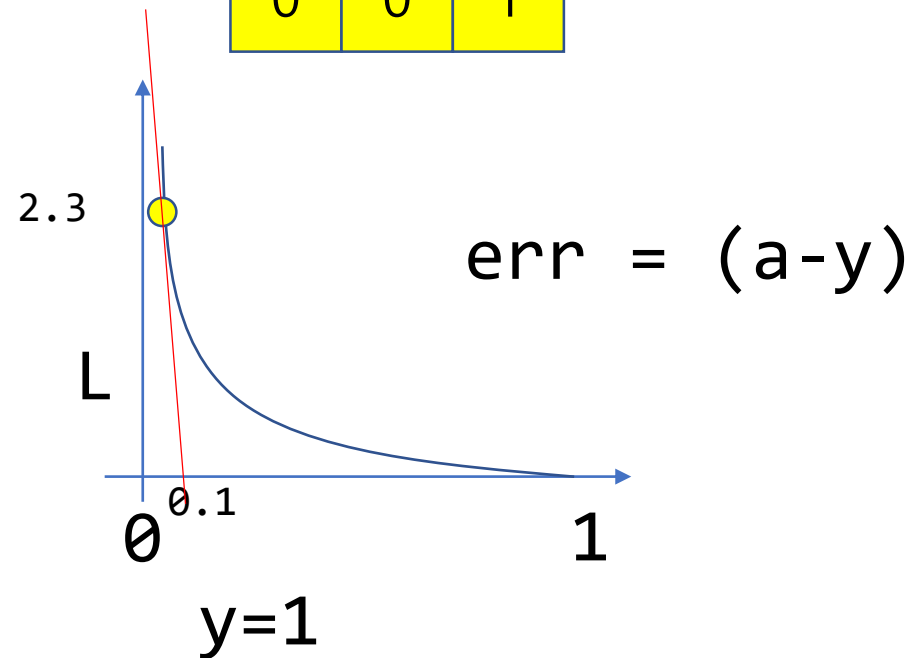
0.2

0.1

0.7 0.2 0.1

0 0 1

$y=[0,0,1]$



(5, 784)(784, 2) (256, 2)

w_1

(256, 2)(2, 3) (256, 3)

w_2

forpass

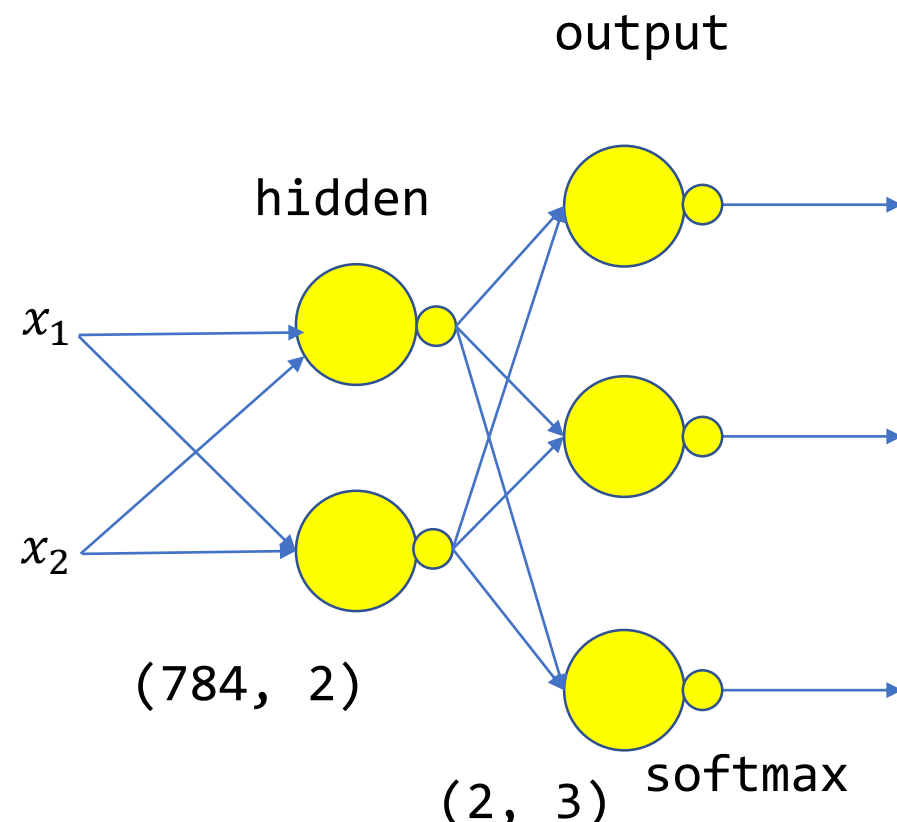
(2, 256)(256, 3) (2, 3)
 $a1.T$ $w2_grad$

backward

$-(y * \text{np.log}(a) + (1 - y) * \text{np.log}(1 - a))$

$\text{np.sum}(-y * \text{np.log}(a))$

2.3



$(256, 784)$ $(784, 2)$ $(256, 2)$
 x w_1 a_1
 $(256, 2)$ $(2, 3)$ $(256, 3)$
 w_2

forpass

0.7
 0.2
 0.1

0.7	0.2	0.1
0	0	1

$y=[0,0,1]$

`np.dot(a1, w2)`

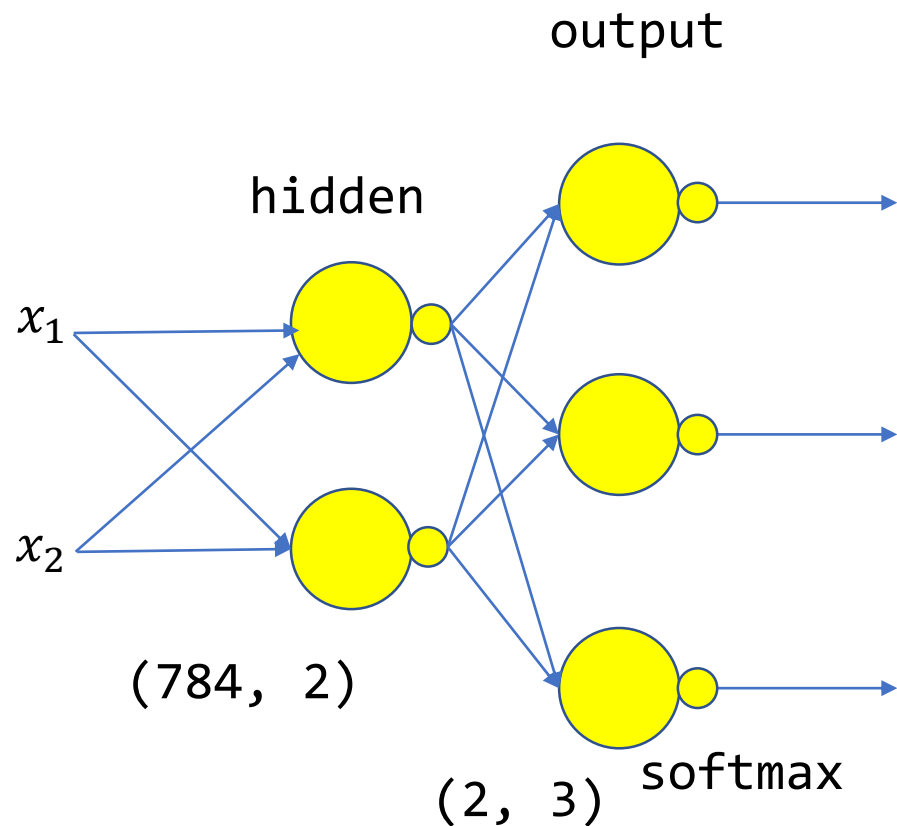
err
 $(256, 3)(3, 2)$ $(256, 2)$
 $w_2.T$ w_2_{grad}

`err_to_hidden = np.dot(err, self.w2.T) * self`

`w1_grad = np.dot(x.T, err_to_hidden) / m`

$(784, 256)(256, 2)$ $(784, 2)$
 w_1_{grad}

backward



0.7

0.2

0.1

0.7	0.2	0.1
-----	-----	-----

0	0	1
---	---	---

$y=[0,0,1]$

```
def predict(self, x):
    z = self.forpass(x)
    return np.argmax(z, axis=
```

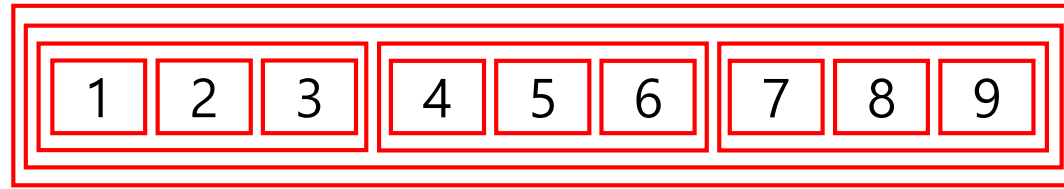
9

$(5, 784)$ $(784, 2)$ $(256, 2)$
 x w_1 a_1

$(56, 2)$ $(2, 3)$ $(256, 3)$
 w_2

forpass

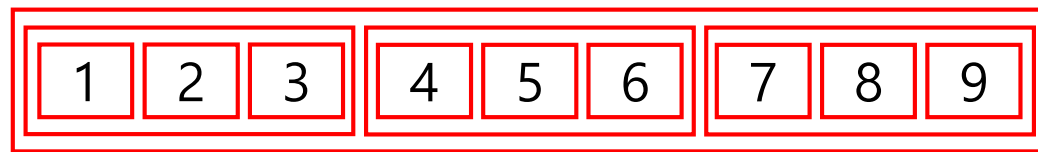
```
xL  
[  
  [[1],[2],[3]],  
  [[4],[5],[6]],  
  [[7],[8],[9]]  
]  
])
```



(1, 3, 3, 1)

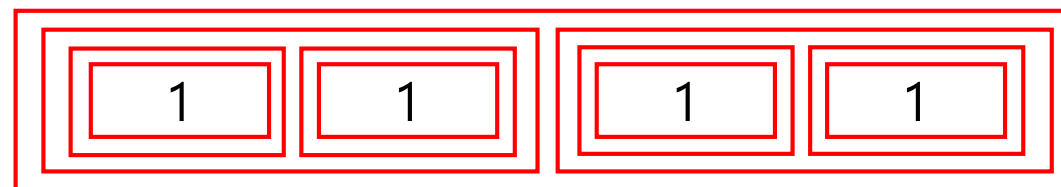

```
[[1.]],  
[[1.]]  
,
```

image

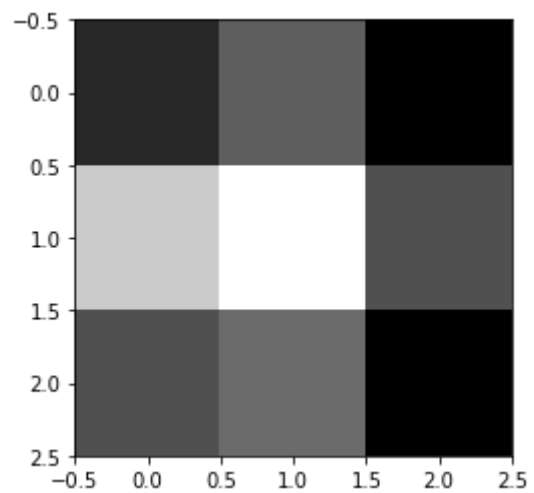
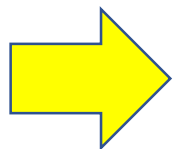
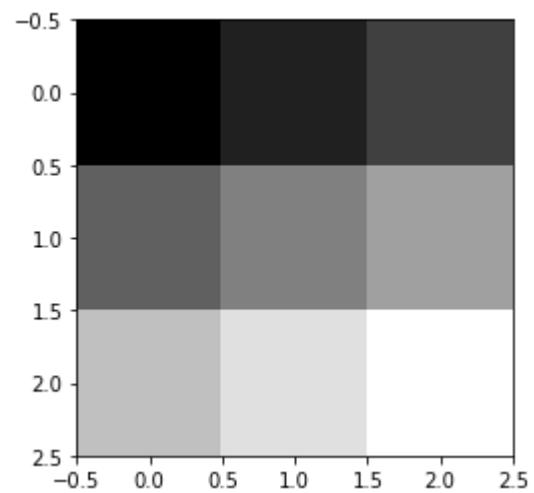


```
[[1.]],  
[[1.]]
```

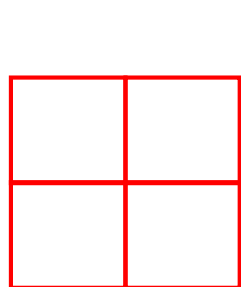
weight



```
(2, 2, 1, 1 )
```



```
conv2d = tf.keras.layers.Conv2D(filters=1, kernel_size=2, padding='valid',  
                                kernel_initializer=weight_init)(image)
```



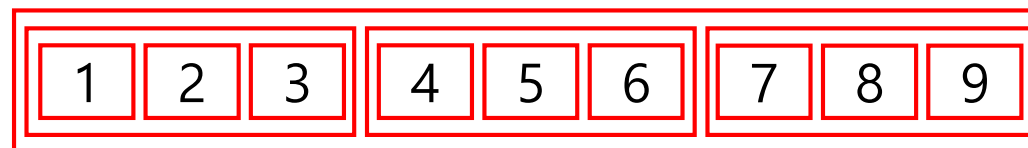
1	2	3
4	5	6
7	8	9

12	16
24	28

weight_init

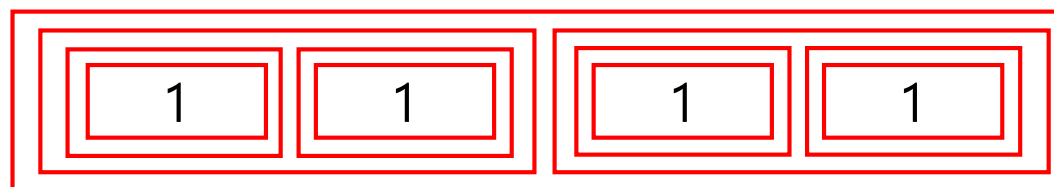
(1, 3, 3, 1)

image



(2, 2, 1, 1)

weight



```
conv2d = tf.keras.layers.Conv2D(filters=1, kernel_size=2, padding='same',
                                kernel_initializer=weight_init)(image)
```

[[12. 16. 9.]
 [24. 28. 15.]
 [15. 17. 9.]]

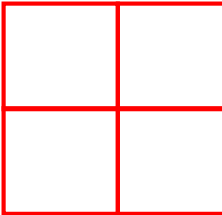
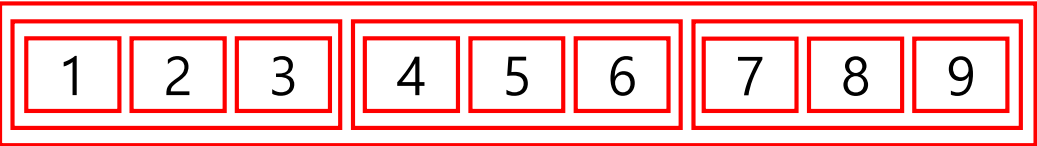
1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

12	16	9
24	28	15
15	17	9

weight_init

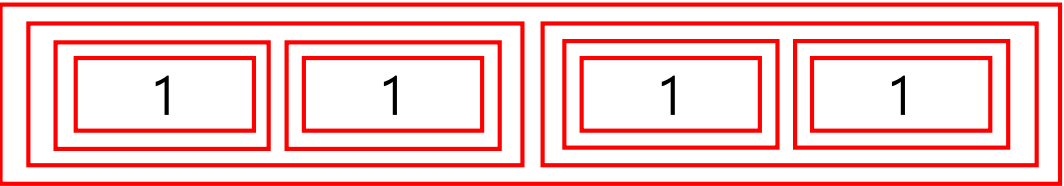
(1, 3, 3, 1)

image



(2, 2, 1, 1)

weight



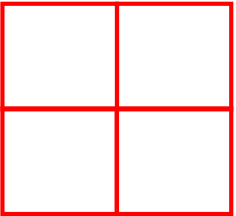
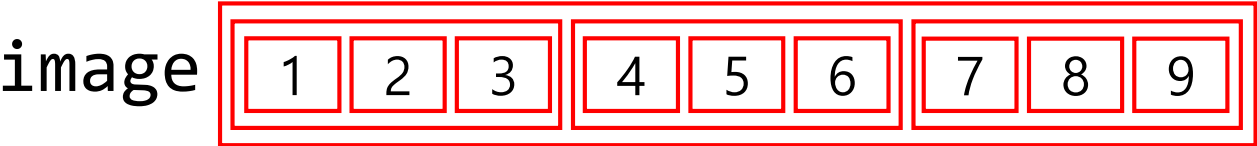
```
conv2d = tf.keras.layers.Conv2D(filters=3, kernel_size=2, padding='same',
                                kernel_initializer=weight_init)(image)
```

[[12. 16. 9.]
 [24. 28. 15.]
 [15. 17. 9.]]

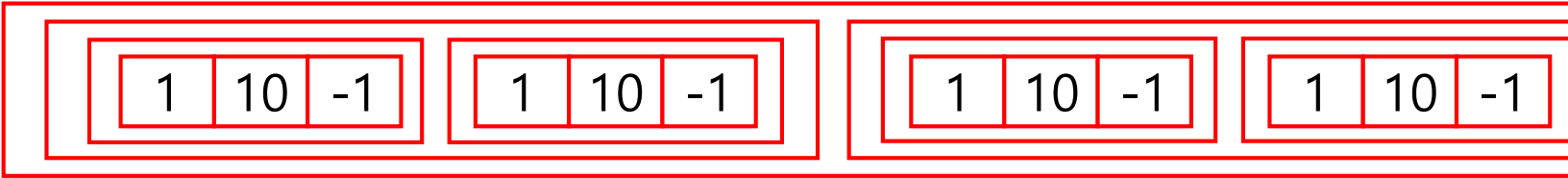
1	2	3	0	12	16	9	120	160	90	-12	-16
4	5	6	0	24	28	15	240	280	150	-24	-28
7	8	9	0	15	17	9	150	170	90	-15	-17
0	0	0	0								

weight_init

(1, 3, 3, 1)

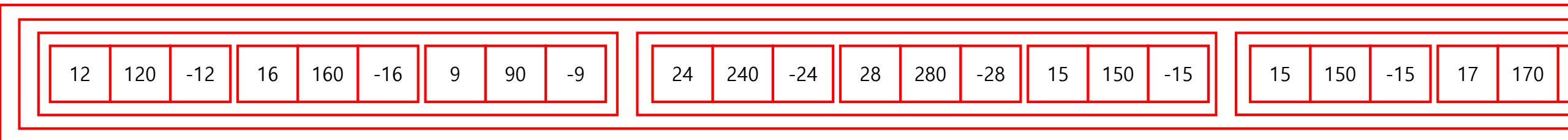


(2, 2, 1, 3) weight

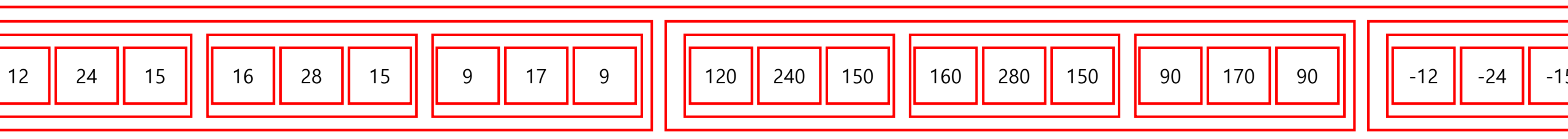


(1,3,3,3)

```
feature_maps = np.swapaxes(conv2d, 0, 3)
```



(3,3,3,1)



```
feature_maps = np.swapaxes(conv2d, 0, 3)
```

`[[12. 16. 9.]`
`[24. 28. 15.]`
`[15. 17. 9.]]`

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

12	16	9
24	28	15
15	17	9

120	160	90
240	280	150
150	170	90

-12	-16
-24	-28
-15	-17

`(1, 3, 3, 3)`

