딥러닝/클라우드

Chapter 09

Learning in Neural Network

Sejong Oh

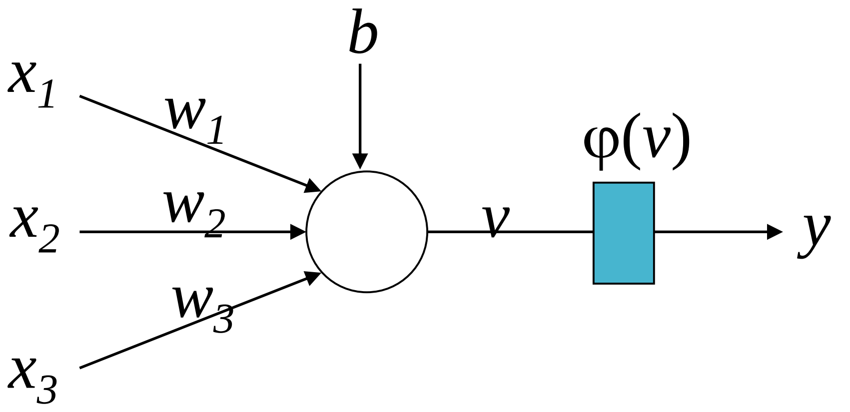
Bio Information technology Lab.

Contents

- Learning in Neural Network
- multi-output perceptron
- Cost function
- Update weight matrix

Summary

- Basic unit of neural network



$$v = w_1x_1 + w_2x_2 + w_3x_3 + b$$
$$= \boldsymbol{wx}^T + b$$

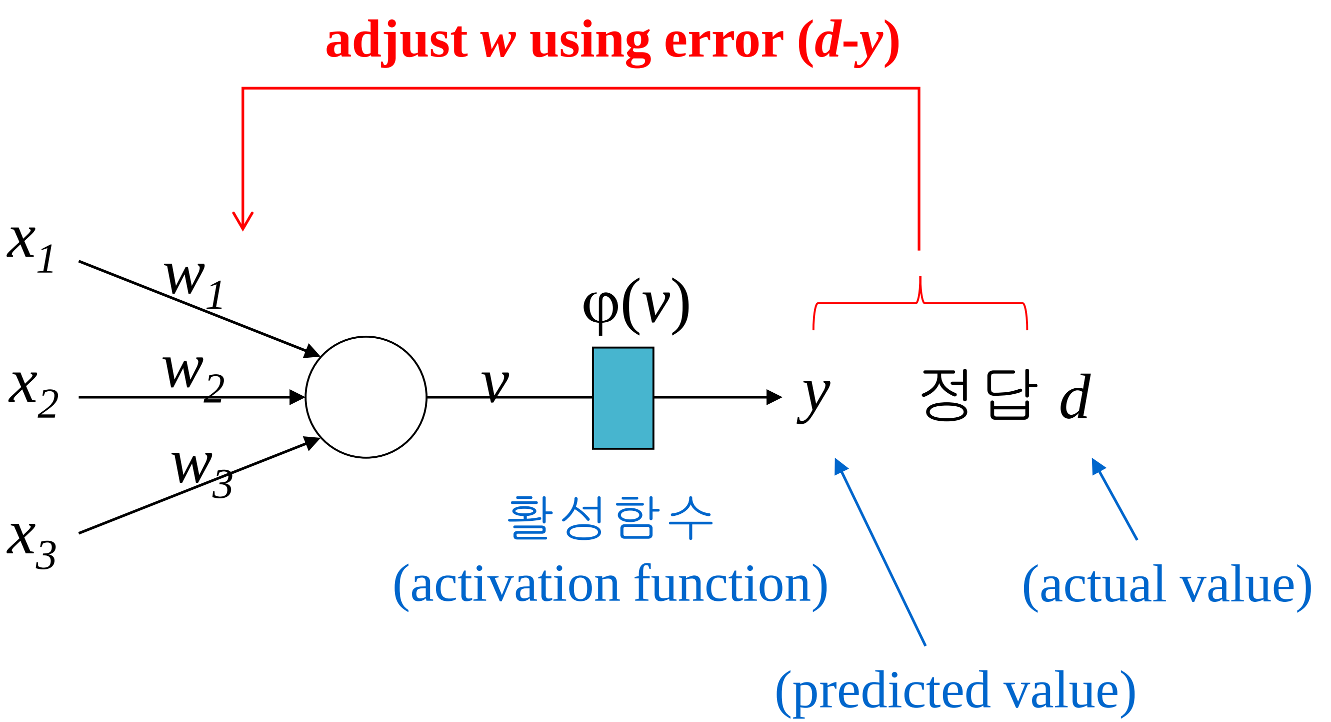
b : bias (편향)

v : weighted sum (가중합)

φ() : activation (활성함수)

1. Learning in Neural Network

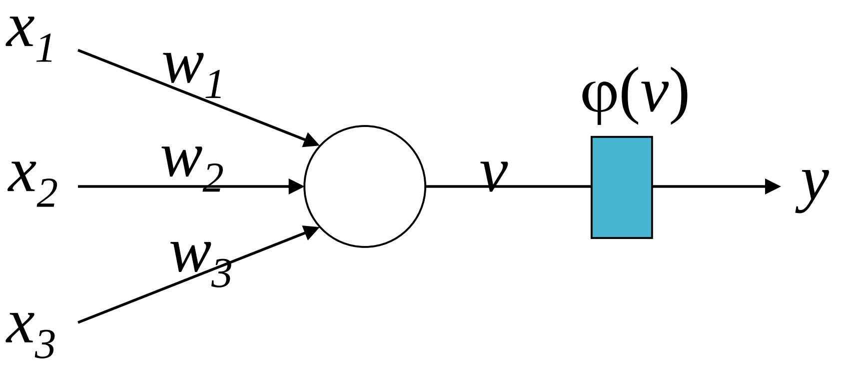
- Learning in single layer perceptron



오차가 충분히 줄어들 때 까지 이 과정을 반복한다.

1. Learning in Neural Network

- Calculate weighted sum v



scalar equation $v = w_1x_1 + w_2x_2 + w_3x_3$

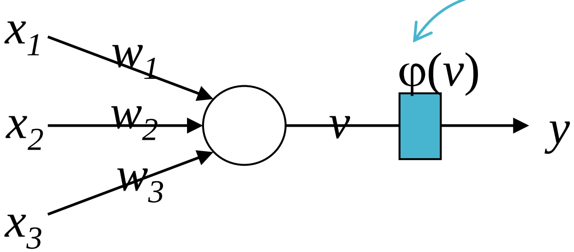
vector equation $v = \boldsymbol{w}^T\boldsymbol{x}$

matrix equation $v = (w_1, w_2, w_3) \bullet (x_1, x_2, x_3)^T = (w_1, w_2, w_3)\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$

1x3 matrix 3x1 matrix

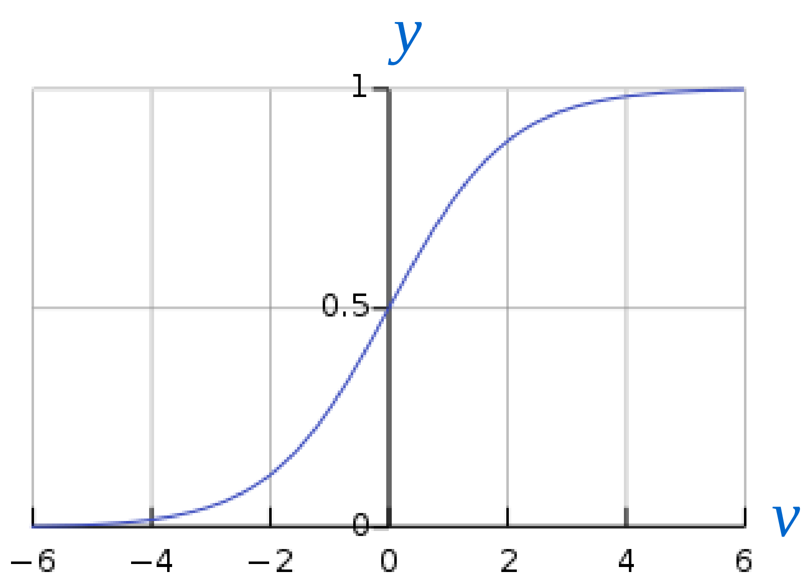
1. Learning in Neural Network

- Activation function $\varphi()$
  - 여러 함수가 사용될 수 있으며 **sigmoid 함수**가 대표적
  - 가중합 v 의 값을 0과 1 사이의 값으로 변환



$$f(x) = \frac{1}{1+\mathrm{e}^{-x}}$$

$e = 2.718281828459045235 36 \cdots$ (자연상수)



1. Learning in Neural Network

- Activation function $\varphi()$
  - **softmax 함수**
  - sigmoid 함수는 자신의 노드로 들어오는 신호의 가중합만 고려하여 출력값 조절 (good for binary-class problem)
  - Softmax 함수는 출력 노드가 여러 개 일 때 자신의 노드 뿐만 아니라 다른 노드로 들어오는 신호의 가중합도 고려 (good for multi-class problem)

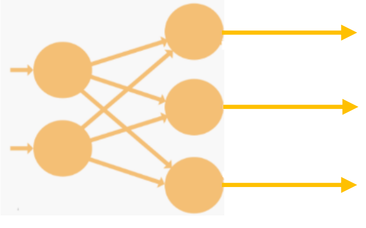
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

***K*: # of output node**

1. Learning in Neural Network

- softmax example

$$v = \begin{bmatrix} 2 \\ 1 \\ 0.1 \end{bmatrix} \Rightarrow \varphi(v) = \begin{pmatrix} \dfrac{e^2}{e^2+e^1+e^{0.1}} \\ \dfrac{e^1}{e^2+e^1+e^{0.1}} \\ \dfrac{e^{0.1}}{e^2+e^1+e^{0.1}} \end{pmatrix} = \begin{pmatrix} 0.6590 \\ 0.2424 \\ 0.0986 \end{pmatrix}$$



합이 1이기 때문에 각 출력에 대한 확률의 의미를 갖는다

1. Learning in Neural Network

- Python code
  - sigmoid

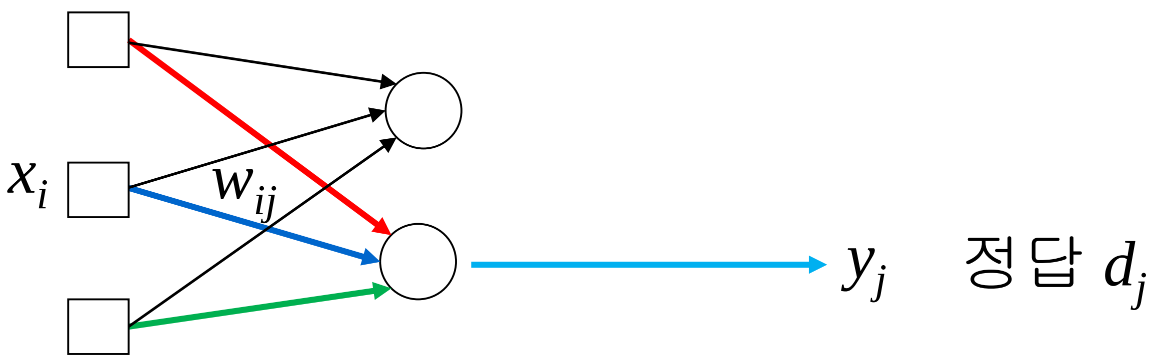
```python
import numpy as np

def SIGMOID(x):
   return 1/(1 + np.exp(-x))
```

  - softmax

```python
import numpy as np

def SOFTMAX(x):
    e_x = np.exp(x)
    return e_x / e_x.sum(axis=0)
```

sigmoid, softmax 외에도 여러 종류의 activation function 존재

- delta rule
  - 신경망의 출력값과 정답사이의 오차를 가지고 w 를 조정하는 방법중의 하나.
  - "어떤 입력노드가 출력노드의 오차에 기여했다면, 두 노드의 연결 가중치는 해당 입력 노드의 입력값(x_j)과 출력 노드의 오차(e_i) 에 비례하여 조절한다"
  - Special type of backpropagation
  - If cost function is mean of square error (MSE), it is gradient descent



$$e_j = d_j - y_j$$

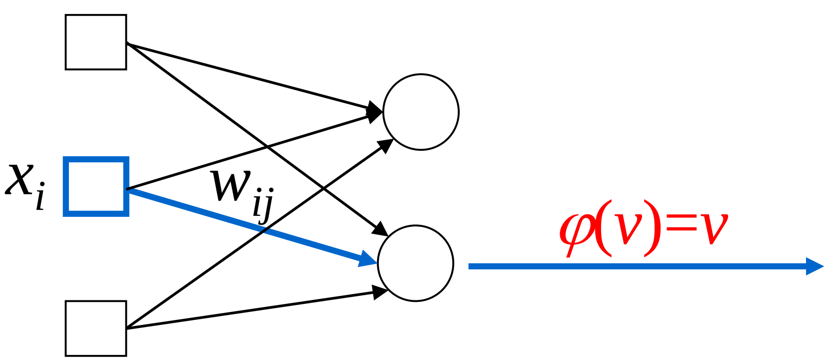
- delta rule : $\varphi(v) = v$ 일때

$$w \leftarrow w + \Delta w$$

$$\Delta w = \alpha \boldsymbol{e} \boldsymbol{x}$$

$$w_{ij} \leftarrow w_{ij} + \alpha e_j x_i$$

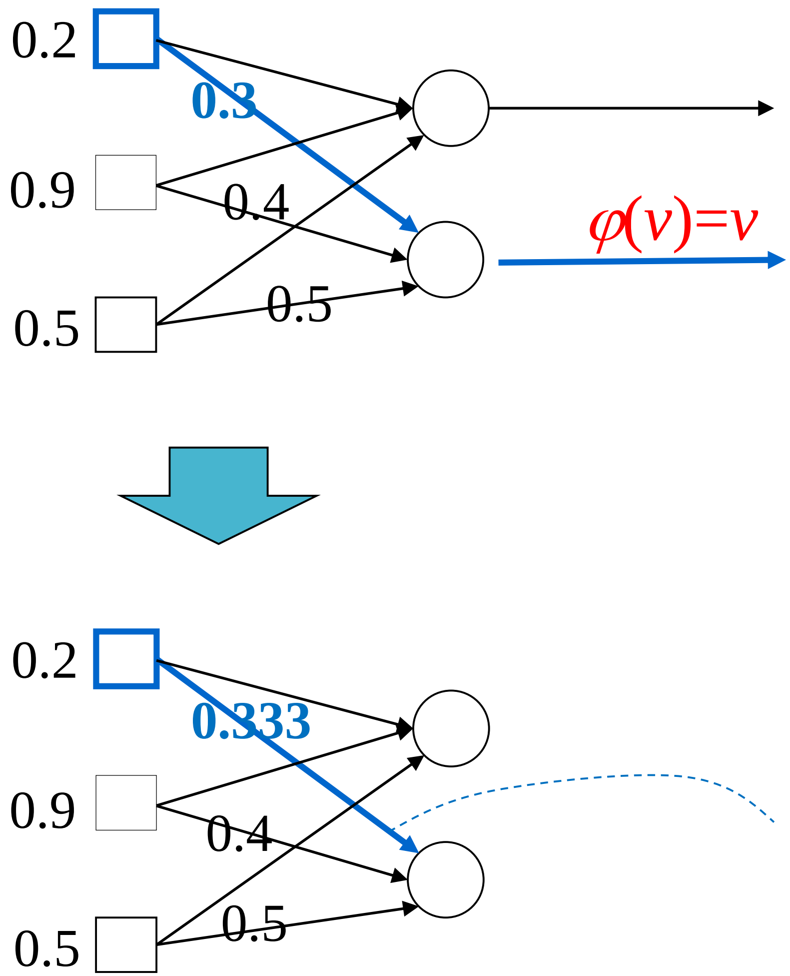
x_i : input of node i (i=1,2,3,..)
e_j : error of output node j (d_j - y_j)
w_{ij} : weight of output j and input i
α : learning rate ($0<\alpha \leq 1$)



y_j 정답 d_j

$e_j = d_j - y_j$

- delta rule example



$d_1=0$

학습률 $\alpha=0.5$

0.67 $d_2=1$

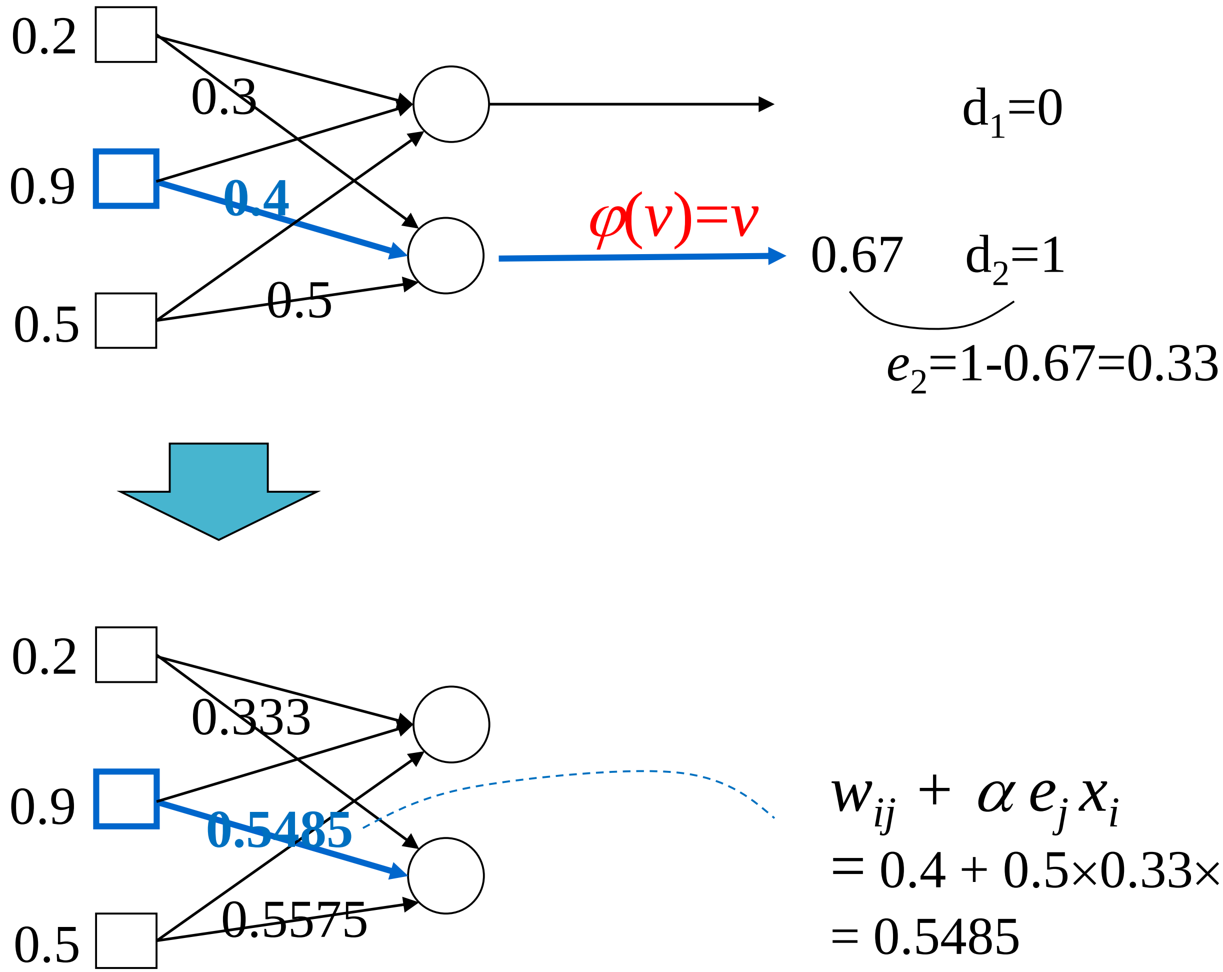
$e_2=1\text{-}0.67=0.33$

$$w_{ij} + \alpha\, e_j\, x_i$$
$$= 0.3 + 0.5\times0.33\times0.2$$
$$= 0.333$$

1. Learning in Neural Network

- delta rule example

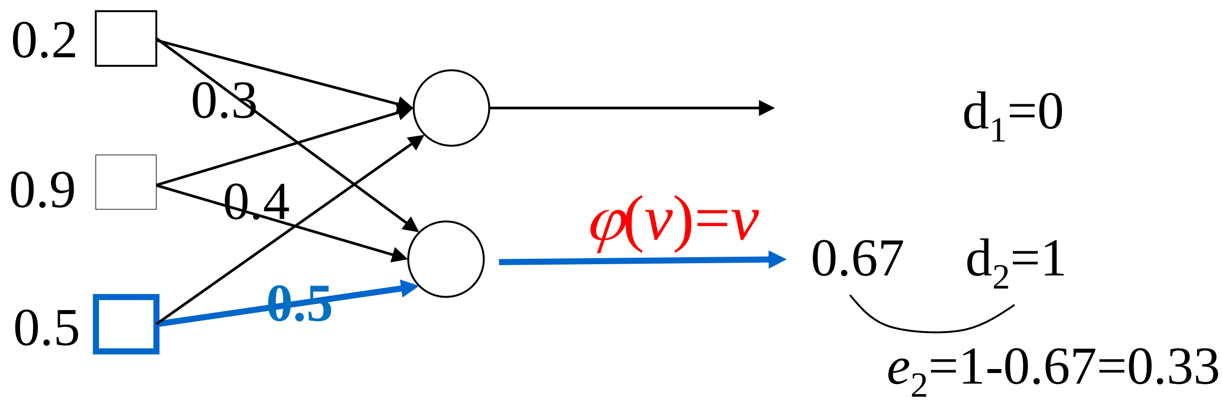
학습률 $\alpha=0.5$



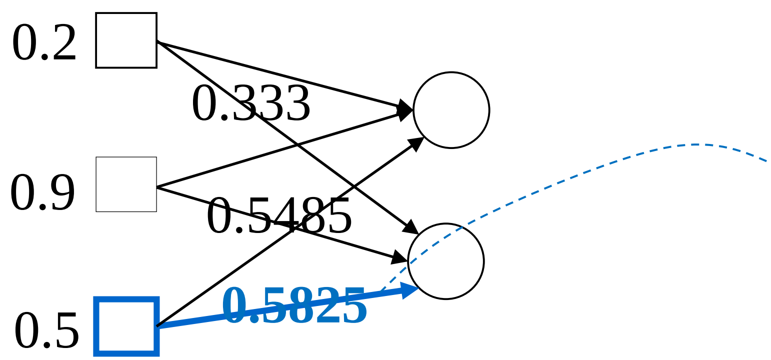
$$w_{ij} + \alpha\, e_j\, x_i$$
$$= 0.4 + 0.5\times0.33\times0.9$$
$$= 0.5485$$

1. Learning in Neural Network

- delta rule example



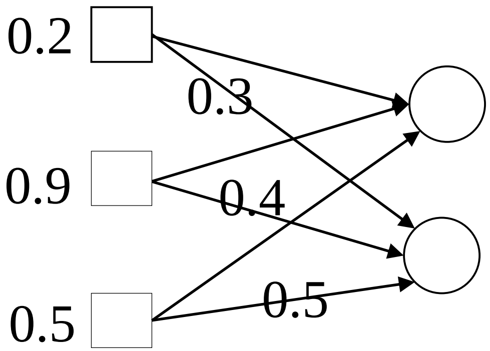
학습률 α=0.5



$$w_{ij} + \alpha\, e_j\, x_i$$
$$= 0.5 + 0.5\times0.33\times0.5$$
$$= 0.5825$$

1. Learning in Neural Network

- delta rule example



$\varphi(v)=v$

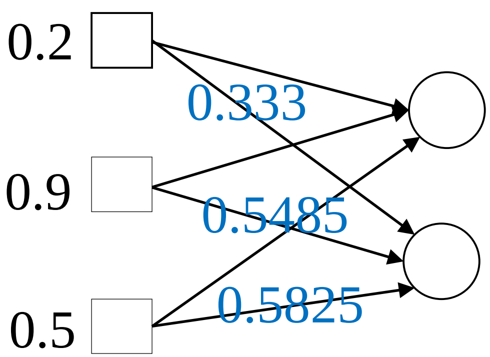
$d_1=0$

학습률 $\alpha=0.5$

0.67 $d_2=1$

$e_2=1-0.67=0.33$

에러가 감소함



$\varphi(v)=v$

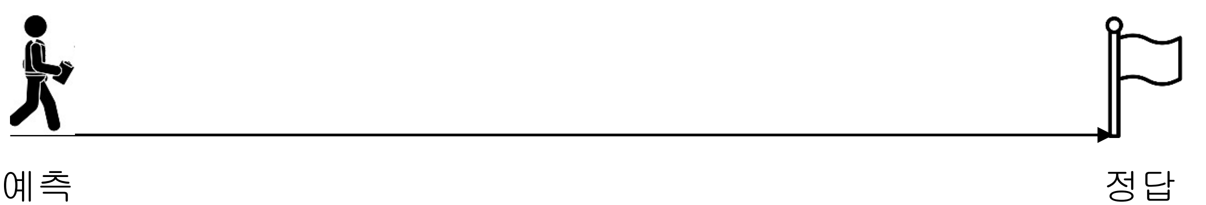
0.8515 $d_2=1$

$e_2=1-0.8515=0.1485$

1. Learning in Neural Network

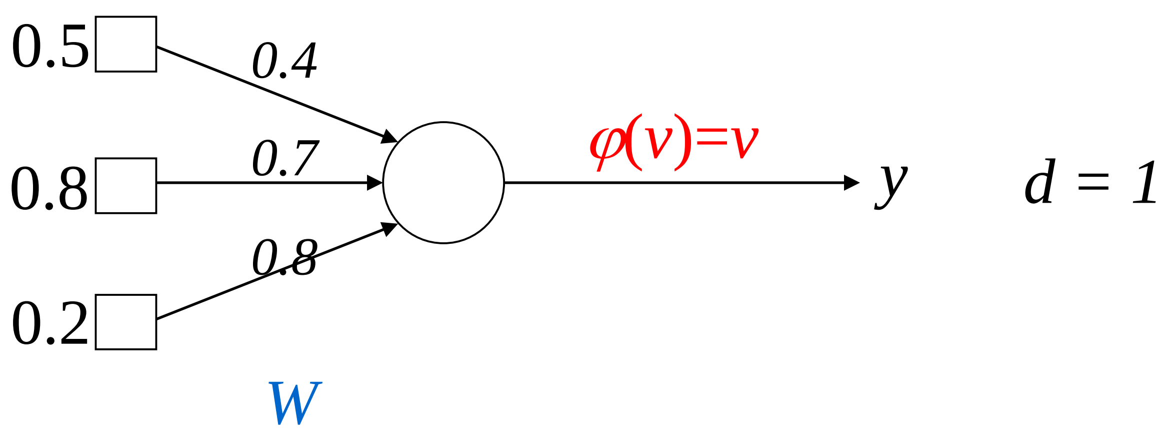
- learning rate
  - 학습률 (0< α ≤1)
  - α 값이 작으면 w 의 변동폭이 작아진다. 학습시간이 길어지는 대신 정답에 보다 근접할 수 있다. 또는 정답에 근접하기 전에 max iteration limit 에 걸려서 학습이 멈출 수 있다.
  - α 값이 크면 w 의 변동폭이 커진다. 학습시간이 짧아지는 대신 정답 부근에서 멈추어 정답에 접근이 안될 수 있다. 경우에 따라서는 정답에 수렴하지 않고 발산한다.

$$w_{ij} \leftarrow w_{ij} + \alpha e_j x_i$$



1. Learning in Neural Network

- Example 1
  - Assume below neural network
  - Learning rate α = 0.5, activation function is $\varphi(v)=v$
  - Update W using delta rule. (5 times)
  - Observe the alternation of W and error



1. Learning in Neural Network

- Example 1

initial

$\boldsymbol{w} = (0.4, 0.7, 0.8)$

Repeat 1

e =

$\Delta \boldsymbol{w} = (\quad,\quad,\quad)$

$\boldsymbol{w} = (\quad,\quad,\quad)$

Repeat 2

e =

$\Delta \boldsymbol{w} = (\quad,\quad,\quad)$

$\boldsymbol{w} = (\quad,\quad,\quad)$

Repeat 3

e =

$\Delta \boldsymbol{w} = (\quad,\quad,\quad)$

$\boldsymbol{w} = (\quad,\quad,\quad)$

Repeat 4

e =

$\Delta \boldsymbol{w} = (\quad,\quad,\quad)$

$\boldsymbol{w} = (\quad,\quad,\quad)$

Repeat 5

e =

$\Delta \boldsymbol{w} = (\quad,\quad,\quad)$

$\boldsymbol{w} = (\quad,\quad,\quad)$

$$w_{ij} \leftarrow w_{ij} + \underbrace{\alpha\, e_j x_i}_{\Delta w}$$

Error is decreasing ?

1. Learning in Neural Network

- Generalize delta rule

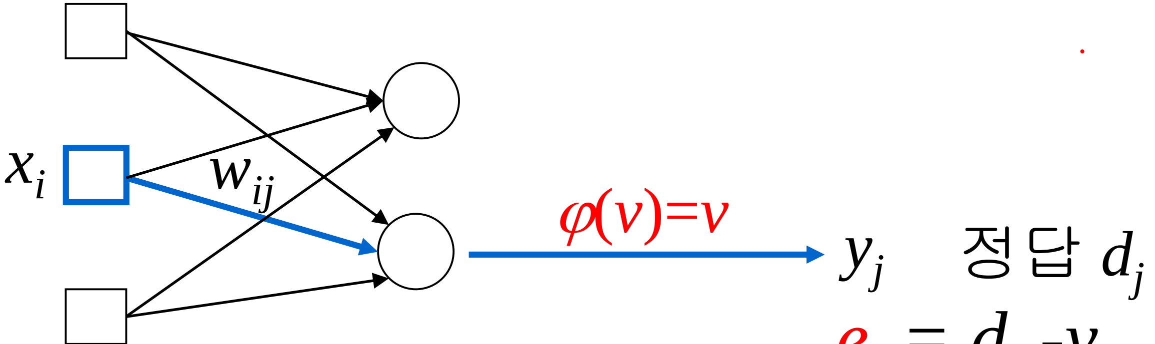
$$w \leftarrow w + \Delta w$$

$$\Delta w = \alpha \boldsymbol{e} \boldsymbol{x}$$

$$w_{ij} \leftarrow w_{ij} + \alpha e_j x_i$$

➡ 이 식은 활성함수가 $\varphi(v)=v$ 일때만 유효하다.

활성함수가 $\varphi(v)=v$ 가 아니라면? => 일반화된 식이 필요



$$e_j = d_j - y_j$$

1. Learning in Neural Network

- Generalize delta rule
  - Consider activation function

④ $w \leftarrow w + \Delta w$

③ $\Delta w = \alpha\, \delta\, x$

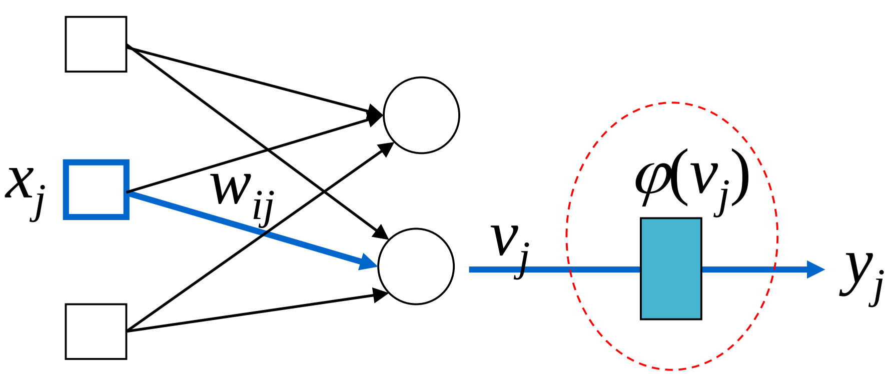
② $\delta = \varphi'(v)\, e$

① $e = d - y$

$$w_{ij} \leftarrow w_{ij} + \alpha \varphi'(v) e\, x_i$$

델타

$\varphi'()$: 출력노드 i 의 활성함수인 $\varphi()$ 의 도함수(Derivative)



정답 d_j

$$e_j = d_j - y_j$$

1. Learning in Neural Network

- delta rule with **constant** function

$$\varphi(v) = v \qquad = \qquad f(x) = x$$

$$\varphi'(v) = 1$$ 미분

$$\delta_i = \varphi'(v_i)\, e_i \quad = e_i$$

$$w_{ij} \leftarrow w_{ij} + \alpha \delta_j x_i$$

$$\boxed{w_{ij} \leftarrow w_{ij} + \alpha\, e_j x_i}$$

1. Learning in Neural Network

- delta rule with **sigmoid** function

$$\varphi(v) = \frac{1}{1 + e^{-v}}$$

$$\varphi'(v) = \varphi(v)(1-\varphi(v))$$

유도과정(미분) 생략

$$\delta_j = \varphi'(v_j)\, e_j = \varphi(v_j)(1-\varphi(v_j))\, e_j$$

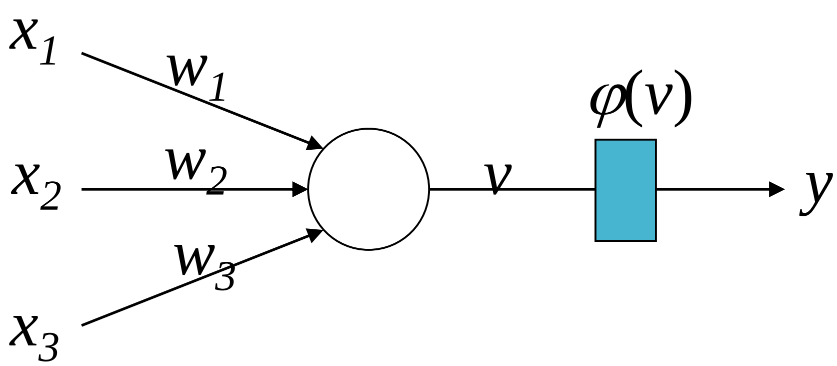
$$w_{ij} \leftarrow w_{ij} + \alpha \delta_j x_i$$

$$w_{ij} \leftarrow w_{ij} + \alpha\, \varphi(v_j)(1-\varphi(v_j))\, e_j x_i$$

* Sigmoid 미분: http://taewan.kim/post/sigmoid_diff/

1. Learning in Neural Network

- Delta rule with **sigmoid** function



$y = \varphi(v)$

$$w_{ij} \leftarrow w_{ij} + \alpha\, \varphi(v_j)(1-\varphi(v_j))\, e_j\, x_i$$

$$\Vert$$

$\varphi(x)$: sigmoid

$$w_{ij} \leftarrow w_{ij} + \underbrace{\alpha\, y(1-y)\, e_j\, x_i}_{\Delta w}$$

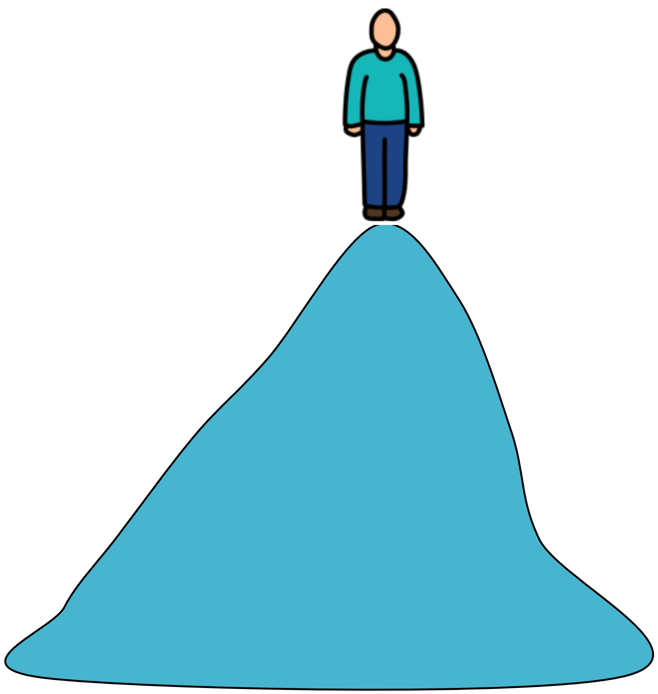
1. Learning in Neural Network

- delta
  - 연결 가중치 값을 update 한다. 목표는 update 된 W 에 의해 산출되는 y 와 d 의 오차가 줄어들게 하는 것
  - 경사하강법(gradient descent)에 의해 오차가 줄어들도록 할 수 있다.
  - 오차가 줄어드는 방향으로 delta 를 update 하려면 미분값이 관여 (기울기)

이론적 설명 : https://js1jj2sk3.tistory.com/103

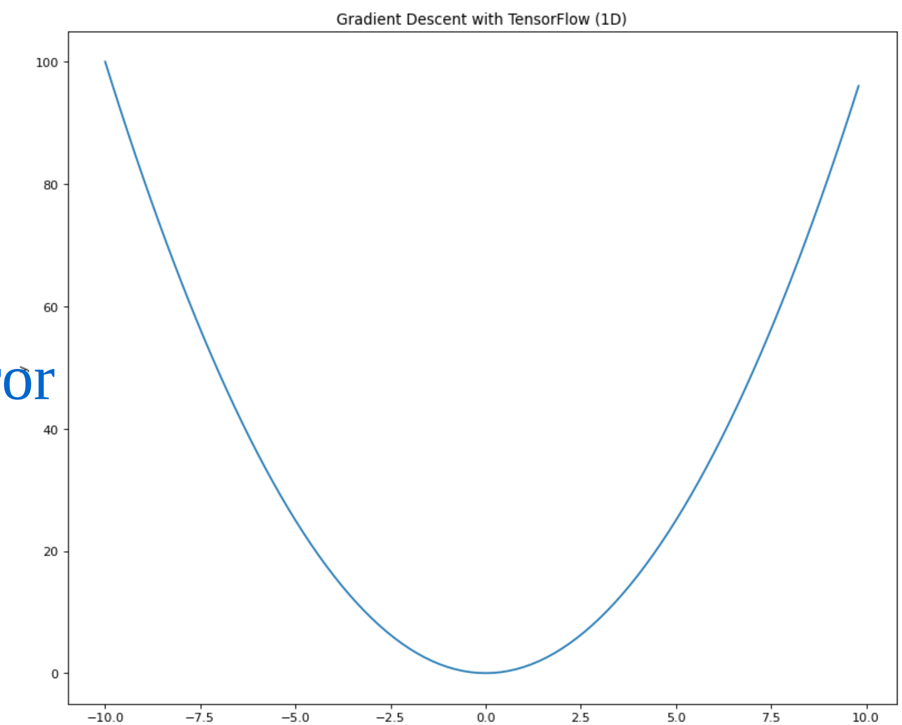
1. Learning in Neural Network

- Gradient descent(경사하강법)

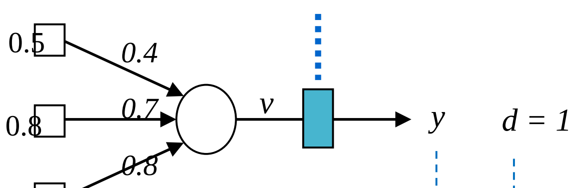


MSE error function
(mean(Σ(d-y)^2)

error



w

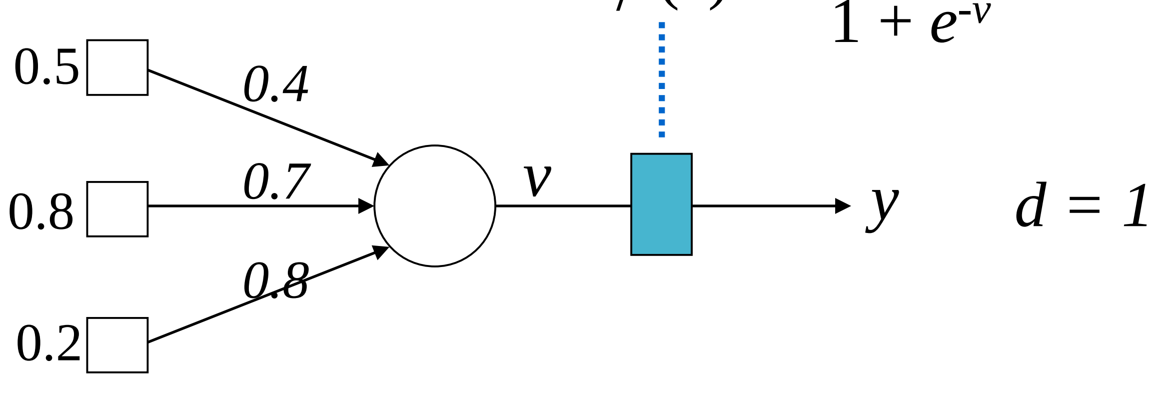


Error를 계산하는
여러가지 방법이 있음

1. Learning in Neural Network

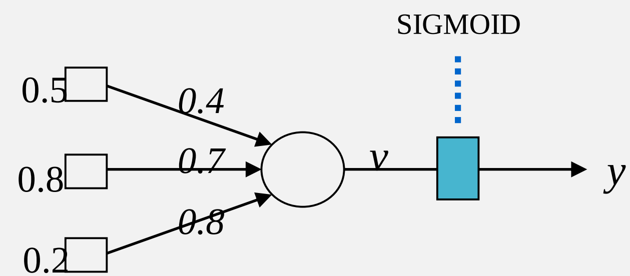
- Example 2
  - Assume below neural network
  - Learning rate α = 0.5, activation function is sigmoid
  - error = d -y
  - Update W using delta rule. (50 times)
  - Observe the alternation of W and error
  - Implement by python

$$\varphi(v) = \frac{1}{1 + e^{-v}}$$



1. Learning in Neural Network

```
## simple delta rule

x = np.array([0.5,0.8,0.2])          # input
w = np.array([0.4,0.7,0.8])          # weight
d = 1                                # 정답
alpha = 0.5

# update w
for i in range(50):
    v = np.sum(w * x)
    y = SIGMOID(v)
    e = d - y
    print("error",i,e)
    w =                                      # update w
```



$$w_{ij} \leftarrow w_{ij} + \alpha\, y(1-y)\, e_i\, x_j$$

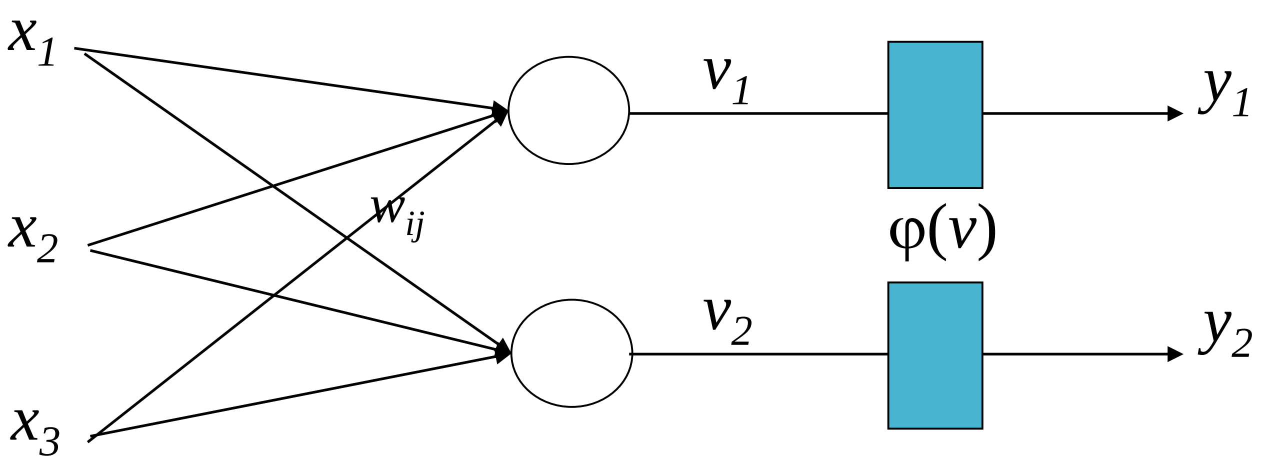
1. Learning in Neural Network

```
error 0 0.2849578942990102
error 1 0.2794887691927339
error 2 0.2742491010755598
error 3 0.26922614783872123
error 4 0.26440792063416385
error 5 0.25978315219123826
error 6 0.25534126252533806
error 7 0.25107232327280227
error 8 0.2469670215879135
error 9 0.24301662429965365
error 10 0.23921294283737404
```

```
error 38 0.1706974890847862
error 39 0.1691127718338511
error 40 0.16756581934176817
error 41 0.1660552575823464
error 42 0.16457977788241818
error 43 0.16313813316490478
error 44 0.16172913444016468
error 45 0.16035164752747189
error 46 0.15900458998988964
error 47 0.15768692826710384
error 48 0.15639767499198376
error 49 0.15513588647773924
```

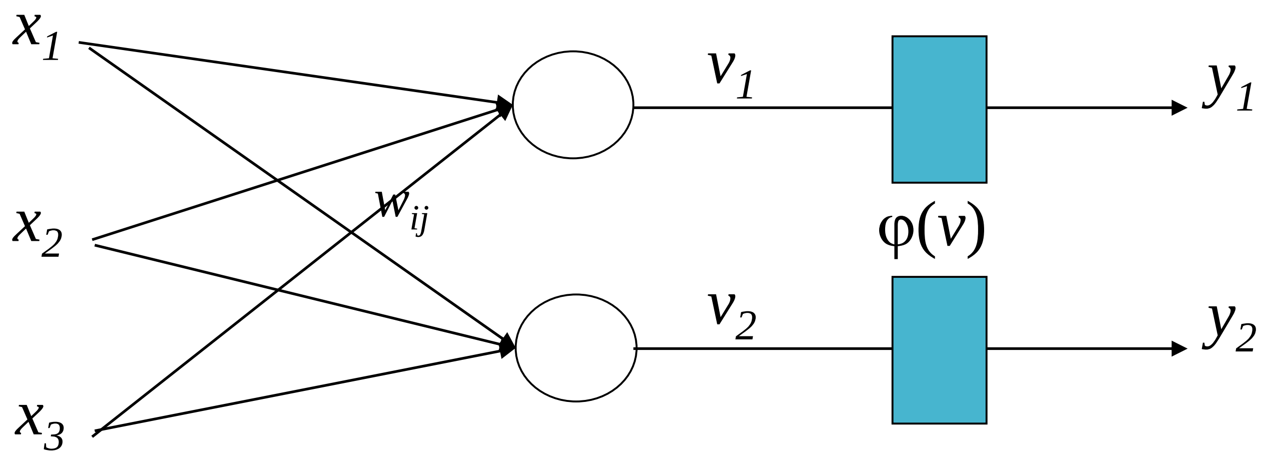
2. multi-output perceptron

- Design of multi-output perceptron
  - Computation of neural network can be expressed by matrix operation
  - Input, weight, v, output 을 어떻게 배열로 표현할 수 있을까?



2. multi-output perceptron

- Design of multi-output perceptron

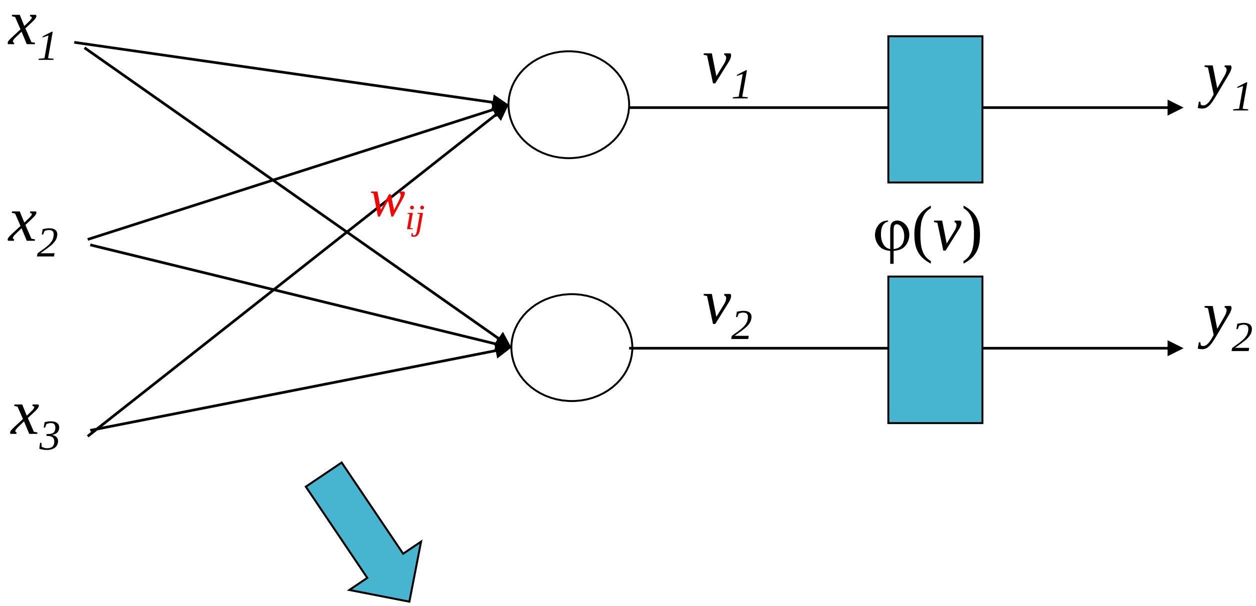


$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \qquad \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

3x1 2x1 2x1

2. multi-output perceptron

- Design of multi-output perceptron

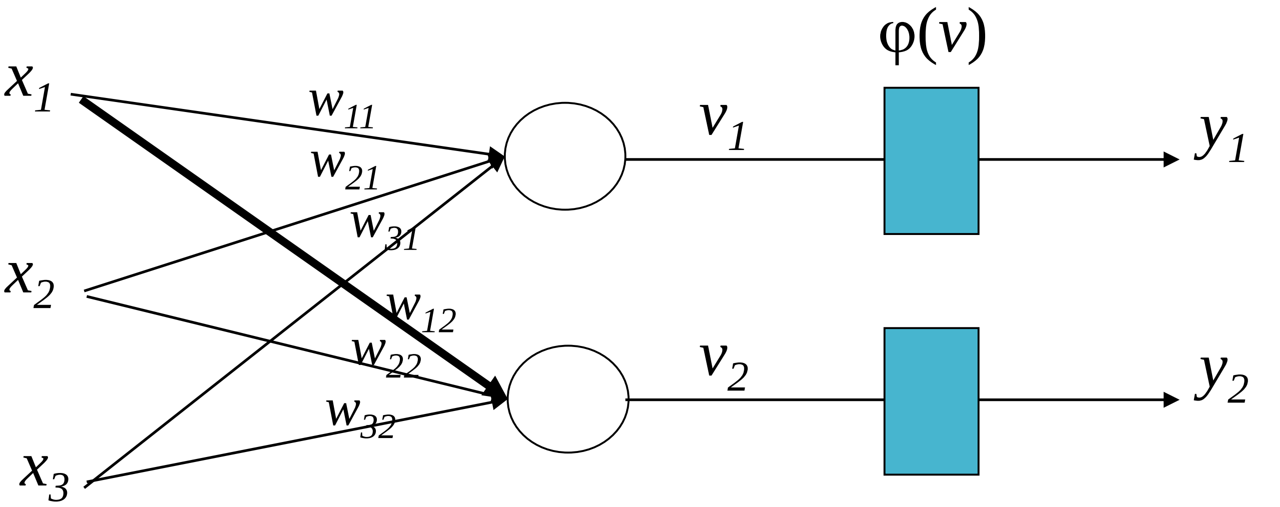


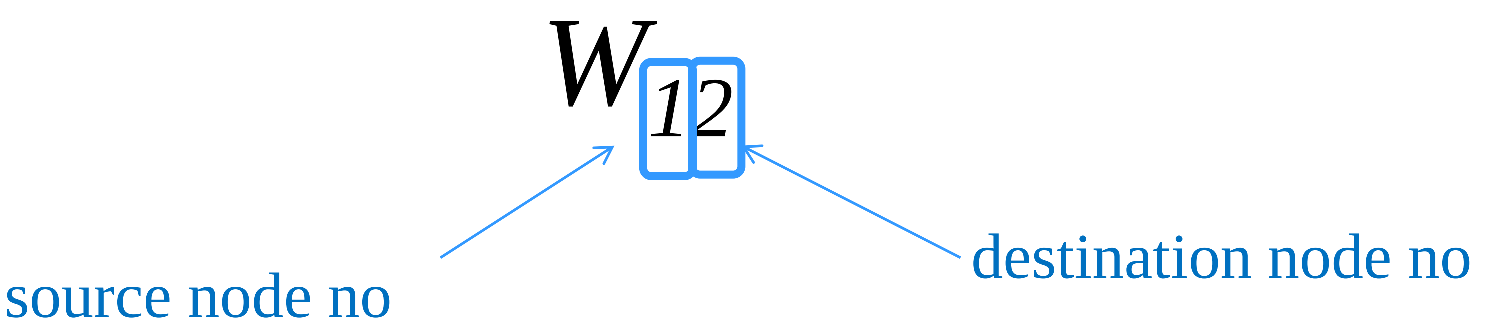
$$\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

3x2   2x3

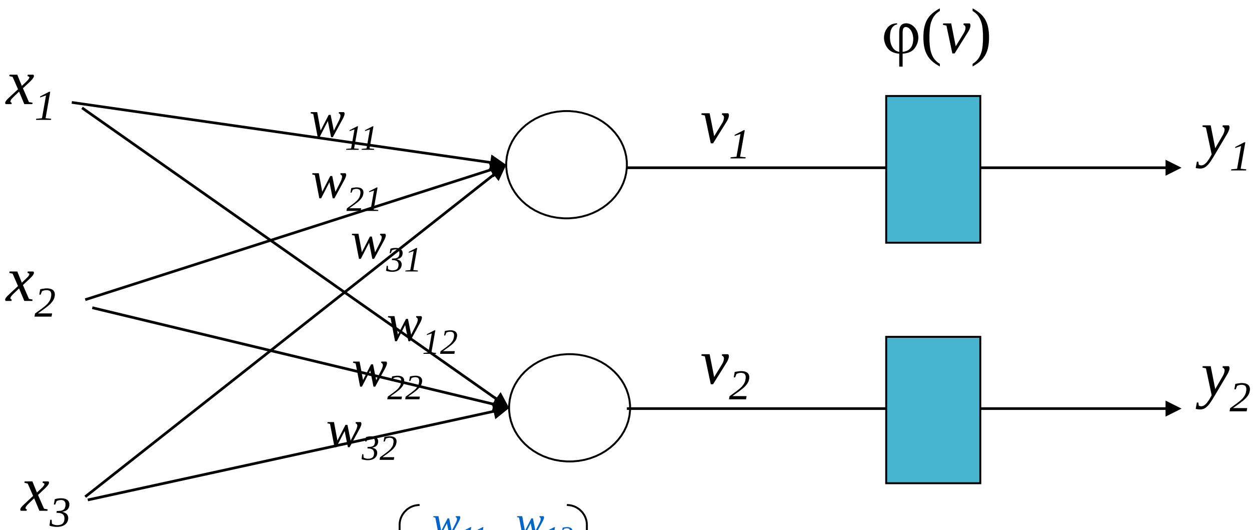
2. multi-output perceptron

- Design of multi-output perceptron





2. multi-output perceptron



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

$$\mathbf{v} = \mathbf{W}^T\mathbf{x}$$

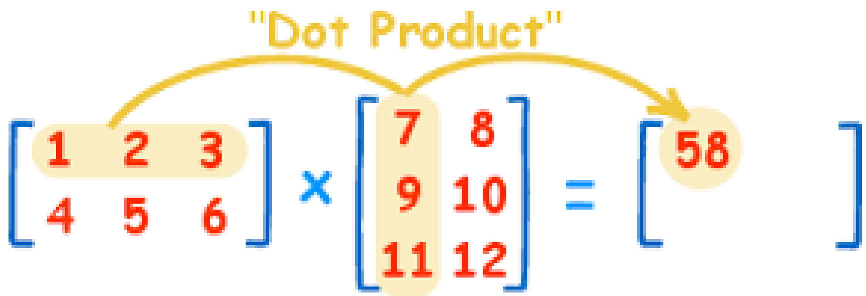
$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \varphi\left(\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}\right)$$

2x1 2x3 3x1

$\otimes$ matrix multiplication

2. multi-output perceptron

- Matrix multiplication



[2, 3] x [3,2] = [2,2]

- Matrix multiplication in python

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

```
In [26]: a
Out[26]:
array([[1, 2, 3],
       [4, 5, 6]])

In [27]: b
Out[27]:
array([[ 7,  8],
       [ 9, 10],
       [11, 12]])

In [28]: c
Out[28]:
array([[ 58,  64],
       [139, 154]])
```

2. multi-output perceptron

Begin of Neural Network is a matrix operation.
End of Neural Network is the matrix operation.

2. multi-output perceptron

- One-hot encoding
  - Class 가 a, b, c 3개 일 때 다음과 같이 coding

```
a ──→ 1   0   0
b ──→ 0   1   0
c ──→ 0   0   1
```

  - One-hot encoding in python

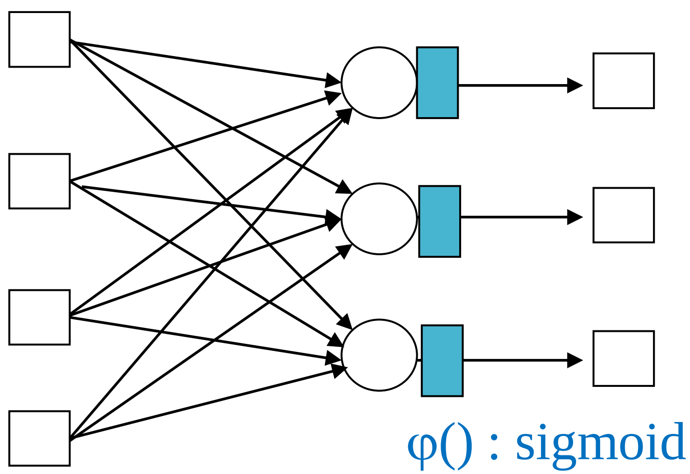
```
import numpy as np

target = np.array([0,1,2])
num = np.unique(target, axis=0)
num = num.shape[0]

encoding = np.eye(num)[target]
```

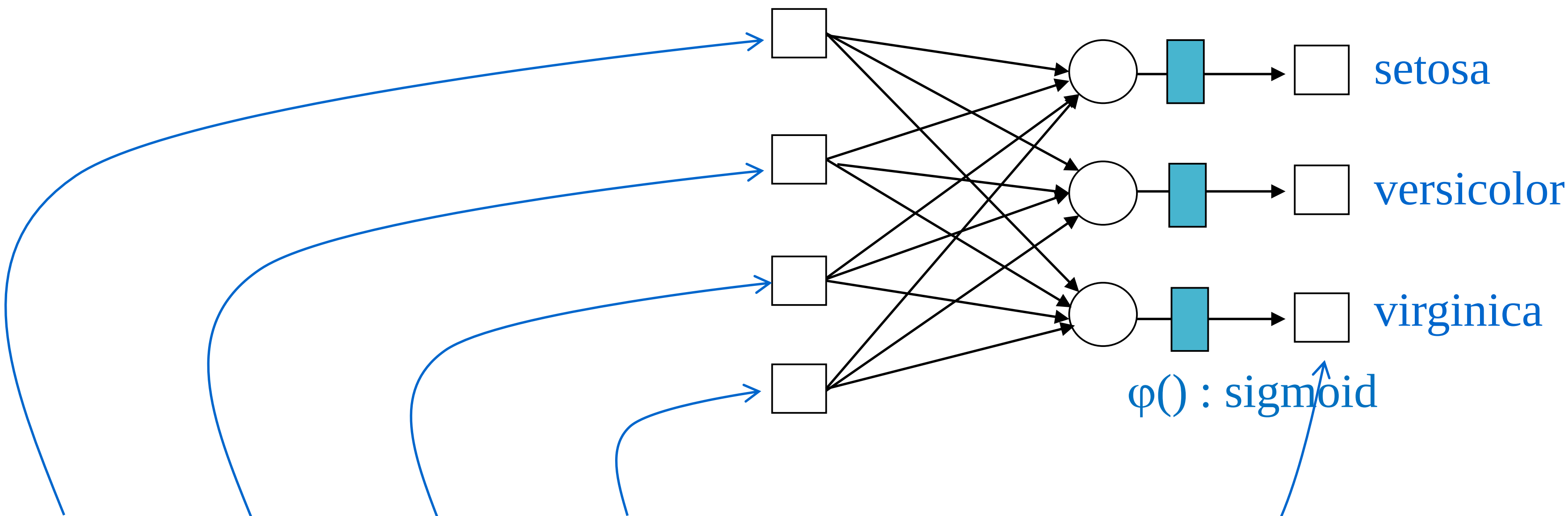
```
In [57]: target
Out[57]: array([0, 1, 2])

In [58]: encoding
Out[58]:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

keras package supports more simple function.

Practice 1

- Implement single layer neural network to predict 'Species' in iris dataset
  - input node: 4, output node: 3
  - Learning rate $\alpha = 0.01$, activation function: sigmoid
  - Initialize weight W with random value between [-0.5, 0.5] (Use runif() function)
  - Repeat time (update W) : 1000

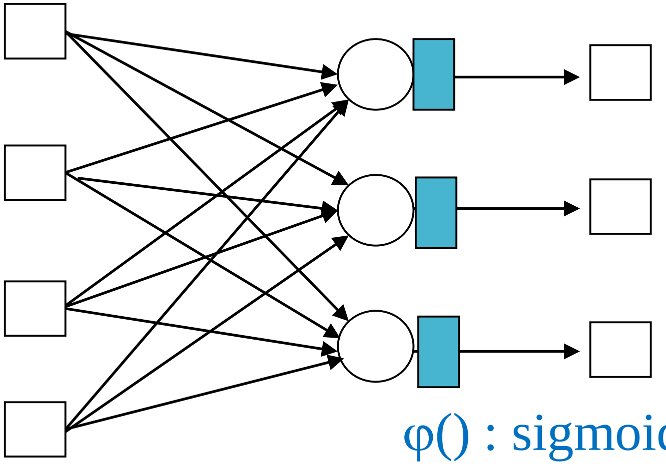




Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

Practice 1

- Implement delta rule



$$\underline{w_{ij}} \leftarrow \underline{w_{ij}} + \underline{\alpha\, y(1-y)e_j}\ \underline{x_i}$$

4x3 4x3 1x3 1x4

How to ?

4x3

Practice 1

```
from sklearn import datasets
import random
import numpy as np

## prepare dataset #####################################
iris = datasets.load_iris()
X = iris.data
target = iris.target

# one hot encoding
num = np.unique(target, axis=0)
num = num.shape[0]
y = np.eye(num)[target]


## Training (get W) ####################################
W = SLP_SGD(X, y,  alpha=0.01, rep=1000)
```

```
error 910 0.00021806650091218496
error 911 0.00020964783902748899
error 912 0.00020123197946933977
error 913 0.0001928189294116 2374
error 914 0.00018440869597426598
error 915 0.00017600128622234182
error 916 0.00016759670716616956
error 917 0.00015919496576205042
error 918 0.0001507960689137907
error 919 0.00014240002347071698
error 920 0.00013400683622930473
error 921 0.0001256165139337101
error 922 0.00011722906327556182
error 923 0.00010884449089496966
error 924 0.00010046280338050125
error 925 9.208400726794241e-05
error 926 8.370810904380817e-05
error 927 7.533511514265974e-05
error 928 6.696503195043983e-05
error 929 5.859786580085976e-05
error 930 5.023362297924364e-05
error 931 4.187230972076128e-05
error 932 3.351393221244966e-05
error 933 2.5158496592176765e-05
```

Error 다시 증가

```
error 996 -0.0004950674258682777
error 997 -0.0005032238319262472
error 998 -0.0005113769886394467
error 999 -0.0005195268925034524
```

```
In [154]: W
Out[154]:
array([[ 0.30149912,  1.21107872, -2.82156052],
       [ 1.89267237, -2.12463262, -2.57351699],
       [-2.40522645,  0.16317877,  3.93583929],
       [-1.10204224, -2.23254871,  3.73092153]])
```

Practice 1

```
## Test #############################################
pred = np.zeros(X.shape[0])
for i in range(X.shape[0]):
    v =  np.matmul(X[i,:],W)
    y = SIGMOID(v)

    pred[i] = np.argmax(y)
    print("target, predict", target[i], pred[i])

print("accuracy :", np.mean(pred==target))
```

```
target, predict 0 0.0
target, predict 1 1.0
target, predict 1 1.0
target, predict 1 1.0
target, predict 1 1.0
target, predict 1 1.0
target, predict 1 2.0
target, predict 1 2.0
target, predict 1 1.0
```

```
In [156]: print("accuracy :", np.mean(pred==target))
accuracy : 0.9133333333333333
```

Practice 1

```
# SLP function ##########################################
def SLP_SGD(tr_X, tr_y, alpha, rep):
    #initialize w
    n = tr_X.shape[1] * tr_y.shape[1]
    random.seed = 123
    w = random.sample(range(1,100), n)
    w = (np.array(w)-50)/100
    w = w.reshape(tr_X.shape[1],-1)

    # update w
    for i in range(rep):
        for k in range(tr_X.shape[0]):
            x = tr_X[k,:]
            v = np.matmul(x, w)
            y = SIGMOID(v)
            e = tr_y[k,:] - y
            w =


        print("error",i,np.mean(e))
    return w
```

$$w_{ij} \leftarrow w_{ij} + \alpha\, y(1-y)\, e_j\, x_i$$

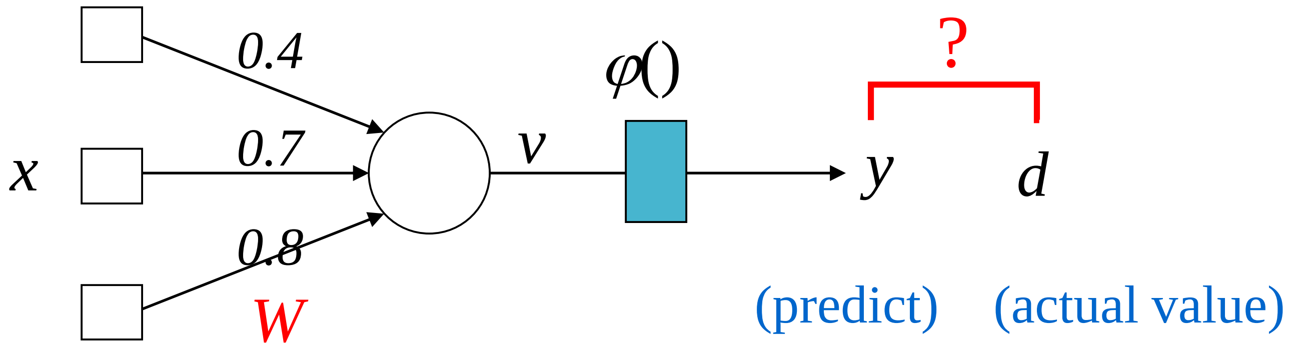
Exercise 1

- Practice 1에서 α 값을 0.05, 0.1, 0.5 로 하여 테스트 하여 보시오
  - 에러가 줄어드는 추세를 비교하여 보자
  - 최종 예측 accuracy 가 어떻게 되는지 비교하여 보자

- α 값은 0.01 로 하고 repeat time 을 200, 400, 600 으로 하여 테스트 하여 보시오
  - 최종 예측 accuracy 가 어떻게 되는지 비교하여 보자

3. Cost function

- Cost function
  - Also called loss function
  - Way to measure error



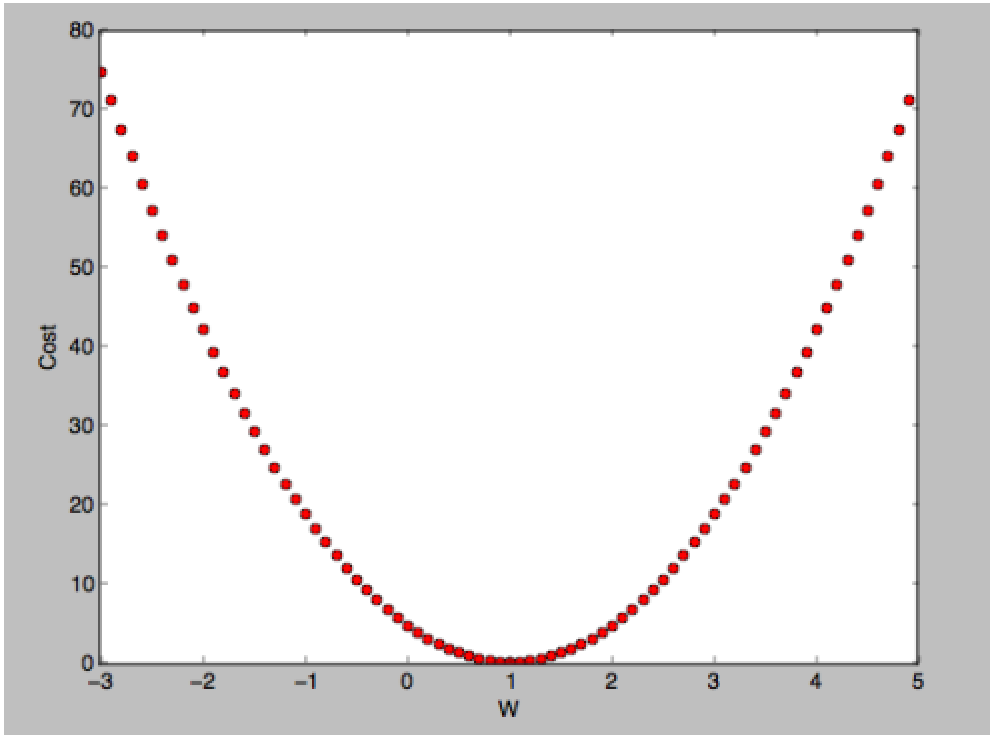
- Typical cost function for neural network
  - Sum of square error (MSE: Mean of square error)
  - Cross entropy

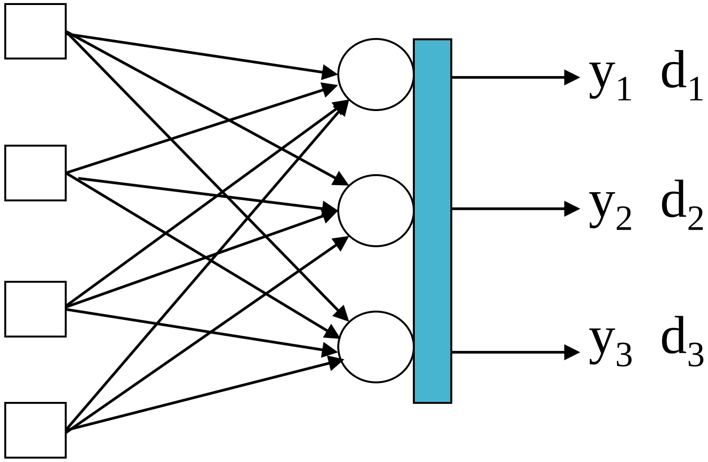
3. Cost function

- Mean of Square Error

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - d^{(i)})^2$$

m: number of output nodes





3. Cost function

- Mean of Square Error

$$cost(\mathrm{W}) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - d^{(i)})^2$$

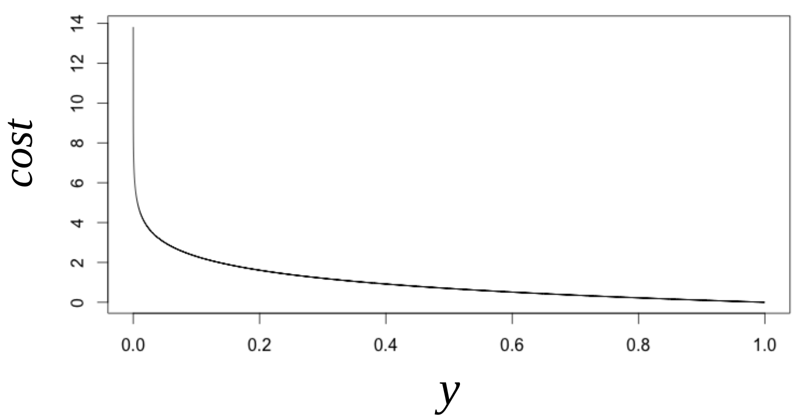
$$y = \begin{pmatrix} 0.2 \\ 0.1 \\ 0.3 \end{pmatrix} \qquad d = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

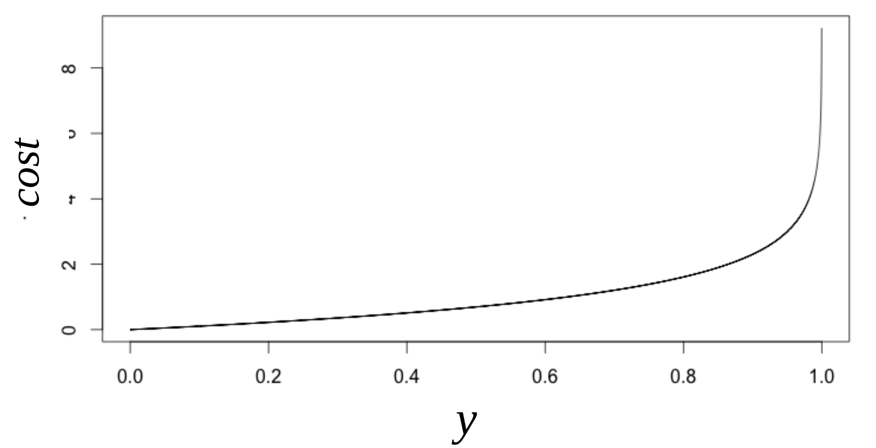
$$\begin{aligned} cost(w) &= (1/3)*((0.2-0)^2+(0.1-1)^2+(0.3-0)^2) \\ &= 0.94/3 \\ &= 0.31 \end{aligned}$$

3. Cost function

- Cross entropy

$$cost(W) = \sum_{i=0}^{m} \underbrace{d_{(i)}(-\log(y_{(i)}))}_{\text{removed when d=0}} + \underbrace{(1-d_{(i)})(-\log(1-y_{(i)}))}_{\text{removed when d=1}}$$





d=1 d=0

Cross entropy is more sensitive to error than sum of square error

3. Cost function

- Cross entropy

$$cost(W) = \sum_{i=0}^{m} \underbrace{d_{(i)}\left(-\log\left(y_{(i)}\right)\right)}_{\text{removed when d=0}} + \underbrace{\left(1 - d_{(i)}\right)\left(-\log\left(1 - y_{(i)}\right)\right)}_{\text{removed when d=1}}$$

$$y = \begin{pmatrix} 0.2 \\ 0.1 \\ 0.3 \end{pmatrix} \qquad d = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

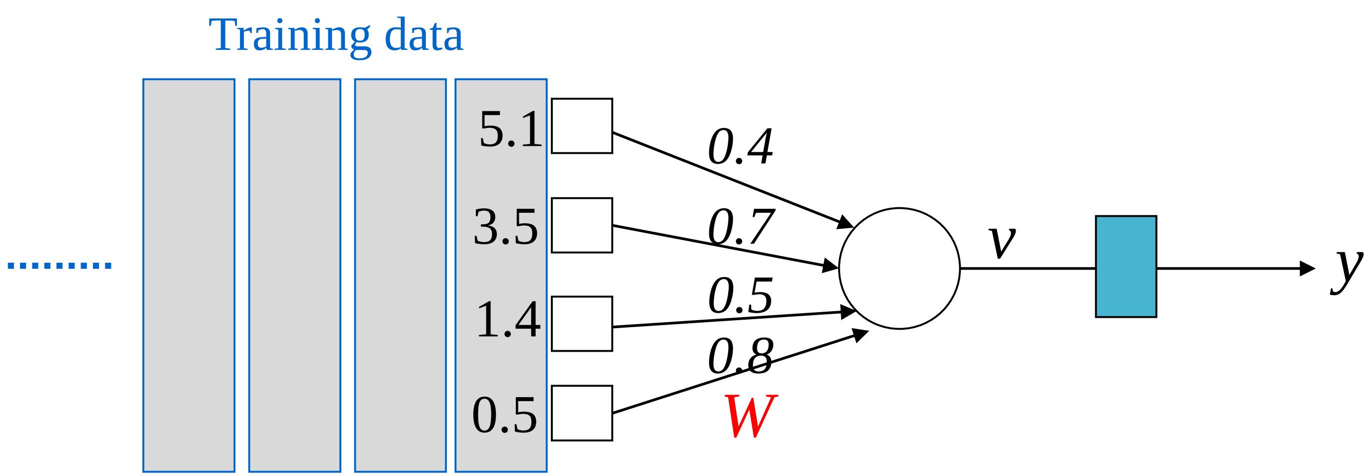
```
cost(w) = (1-0)*(-log(1-0.2))
          + 1*(-log(0.1))
          + (1-0)*(-log(1-0.3))
        = 2.882404
```

4. Update weight matrix

- One time update for one instance of dataset
  ⇨ Big size dataset may require long learning time

- Any other idea?

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4

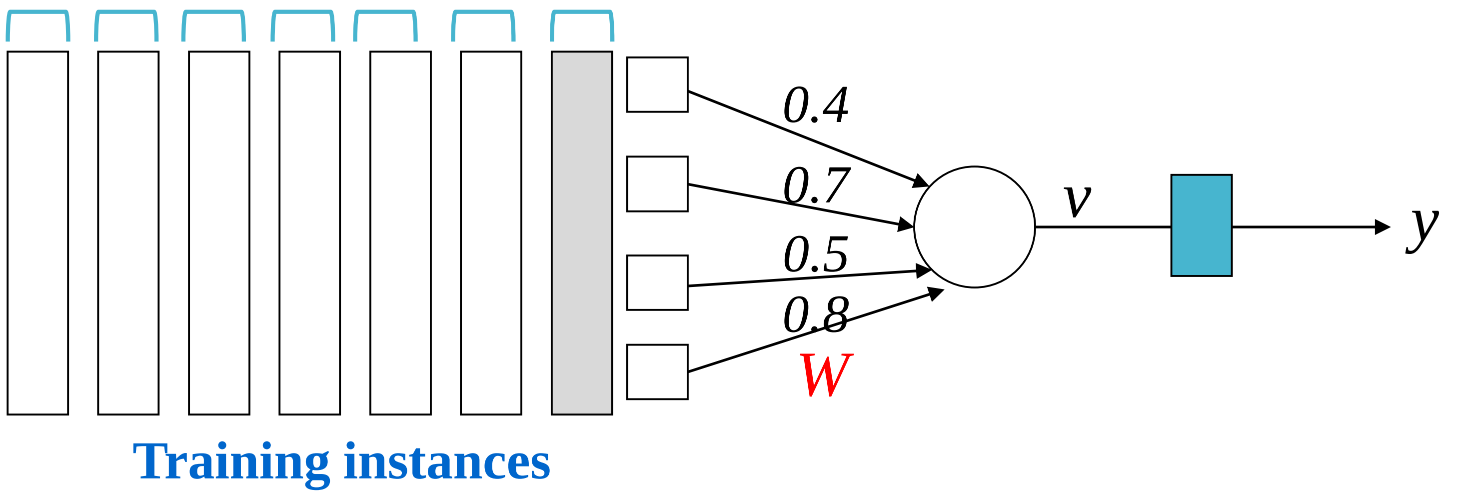


4. Update weight matrix

- weight 갱신 방법
  - Stochastic 경사하강법(SGD; stochastic gradient descent)
    - 하나의 학습 데이터마다 오차를 계산하여 신경망의 가중치를 update
  - 배치(batch)
    - 모든 학습 데이터의 오차에 관해 가중치 갱신값을 계산 다음 이들의 평균값으로 가중치를 한번 update
    - 학습데이터가 많으면 평균 계산에 시간이 많이 걸리고 가중치 갱신도 느려, 학습에 시간이 많이 걸림
  - 미니배치 (mini batch)
    - SGD 와 배치 방식의 중간
    - 전체 학습 데이터에서 일부 데이터만 골라 배치 방식으로 학습

4. Update weight matrix

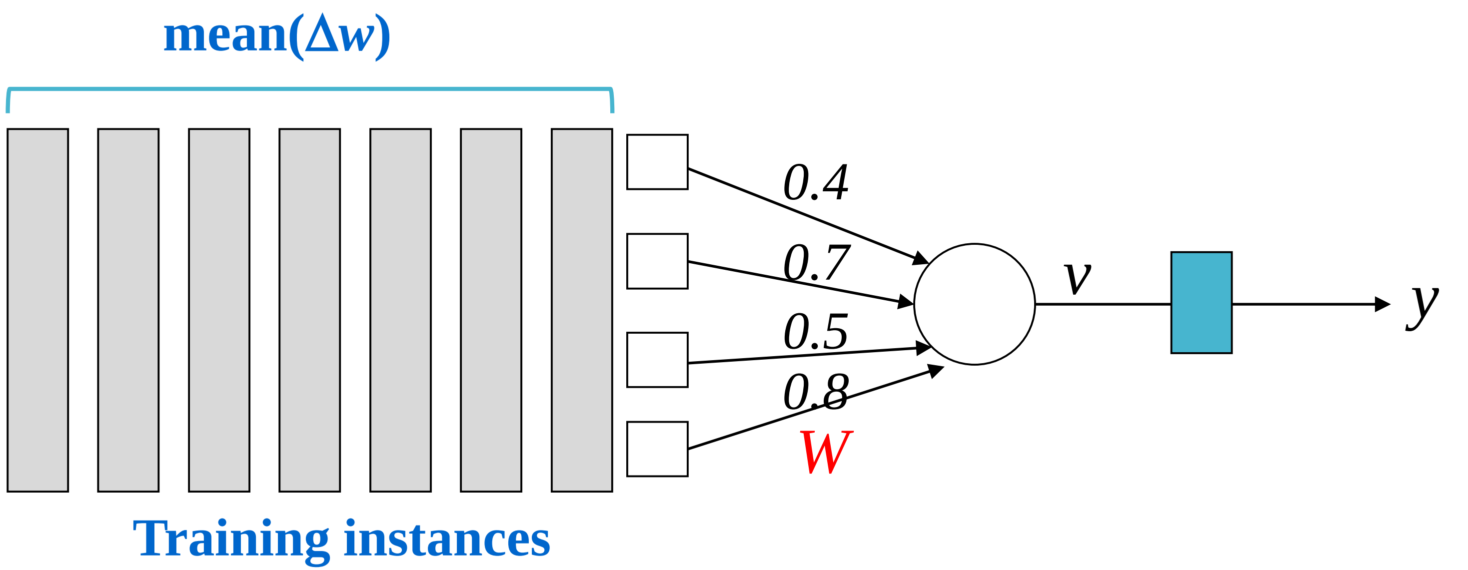
- 연결 가중치 갱신 방법
  - Stochastic 경사하강법(SGD)



학습데이터가 하나 입력 될 때 마다
가중치 (**w = w +** Δ***w***) 갱신

4. Update weight matrix

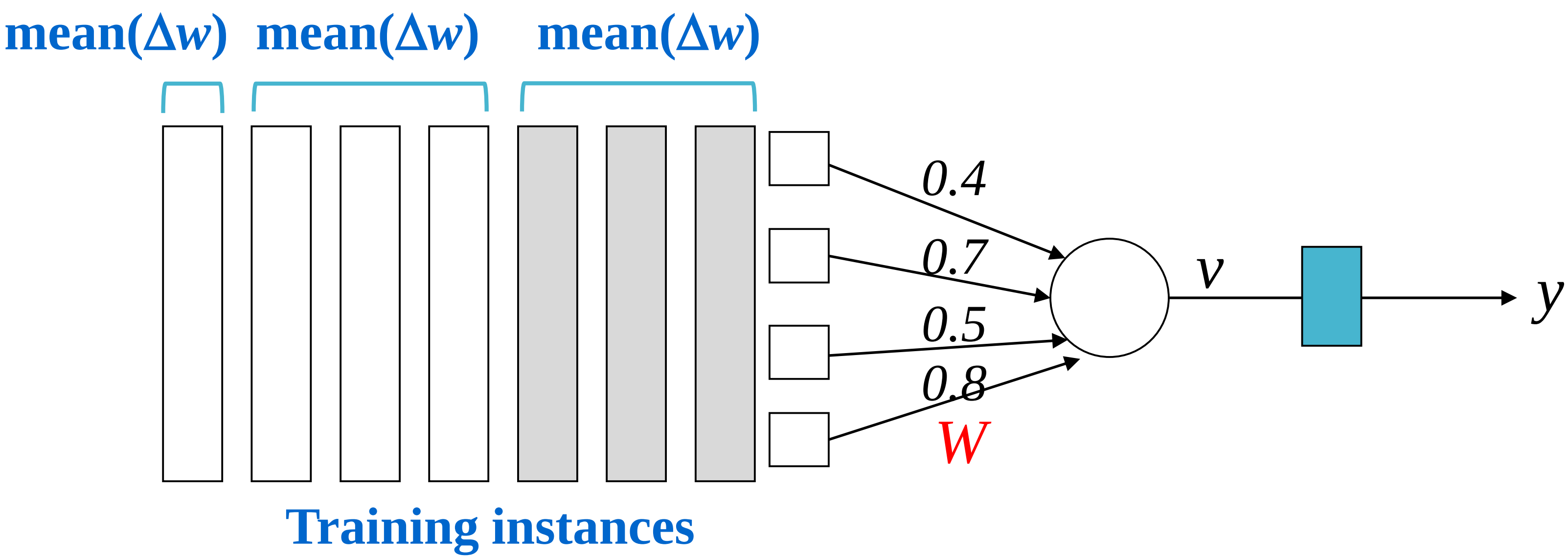
- 연결 가중치 갱신 방법
  - 배치(batch)



모든 학습데이터의 가중치 갱신값(Δ*w*)을 구한 뒤 그 값들의 평균값으로 가중치 갱신 **(w = w + mean(Δ*w*))**

4. Update weight matrix

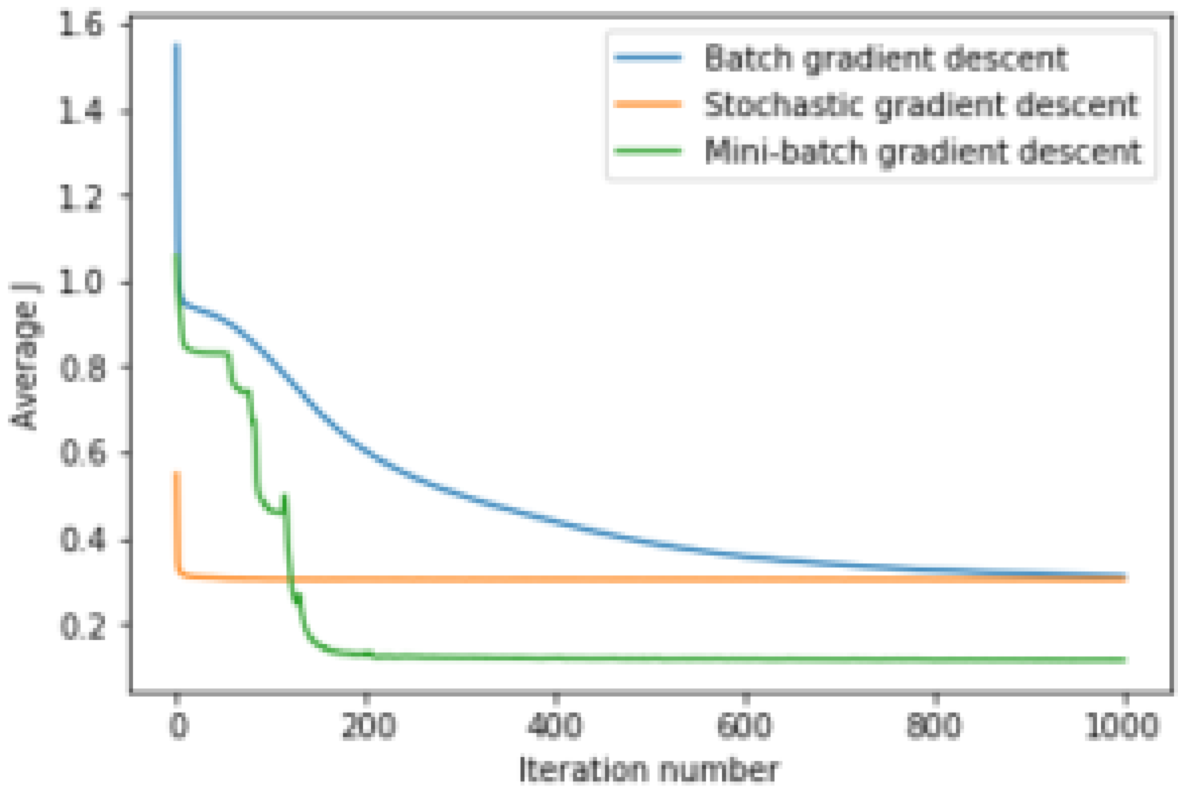
- 연결 가중치 갱신 방법
  - 미니 배치(mini batch)



전체 학습 데이터를 일정 크기로 나누어 배치 방식으로 학습

4. Update weight matrix

- SGD vs batch vs mini batch
  - SGD 방식이 batch 방식보다 오차가 빨리 줄어든다
  - mini batch 방식이 오차를 가장 많이 줄여준다

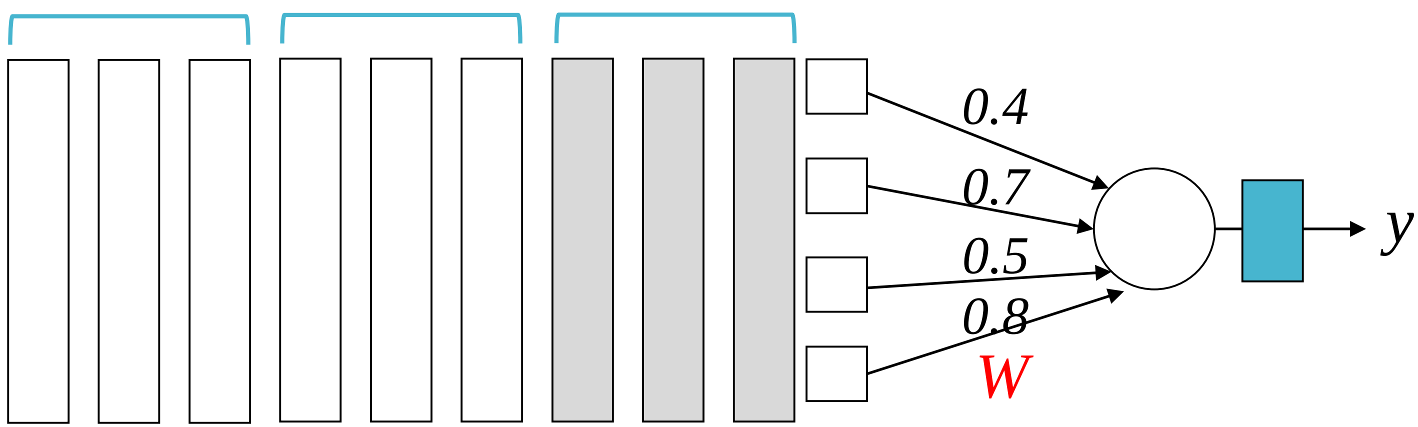


(http://adventuresinmachinelearning.com/stochastic-gradient-descent/)

4. Update weight matrix

- Epoch
  - epoch : 전체 학습 데이터를 한번씩 모두 학습시킨 횟수

  - 예) 9 개의 데이터를 세부분으로 나누어 (mini batch) 10회 학습시킨 경우
    - Batch size = 3, epoch = 10



Key wards

- Activation function
  - Sigmoid, softmax
- Learning rate
- Delta rule
- Cost function / loss function
  - MSE, Cross entropy
- SGD, mini batch, batch
- Epoch