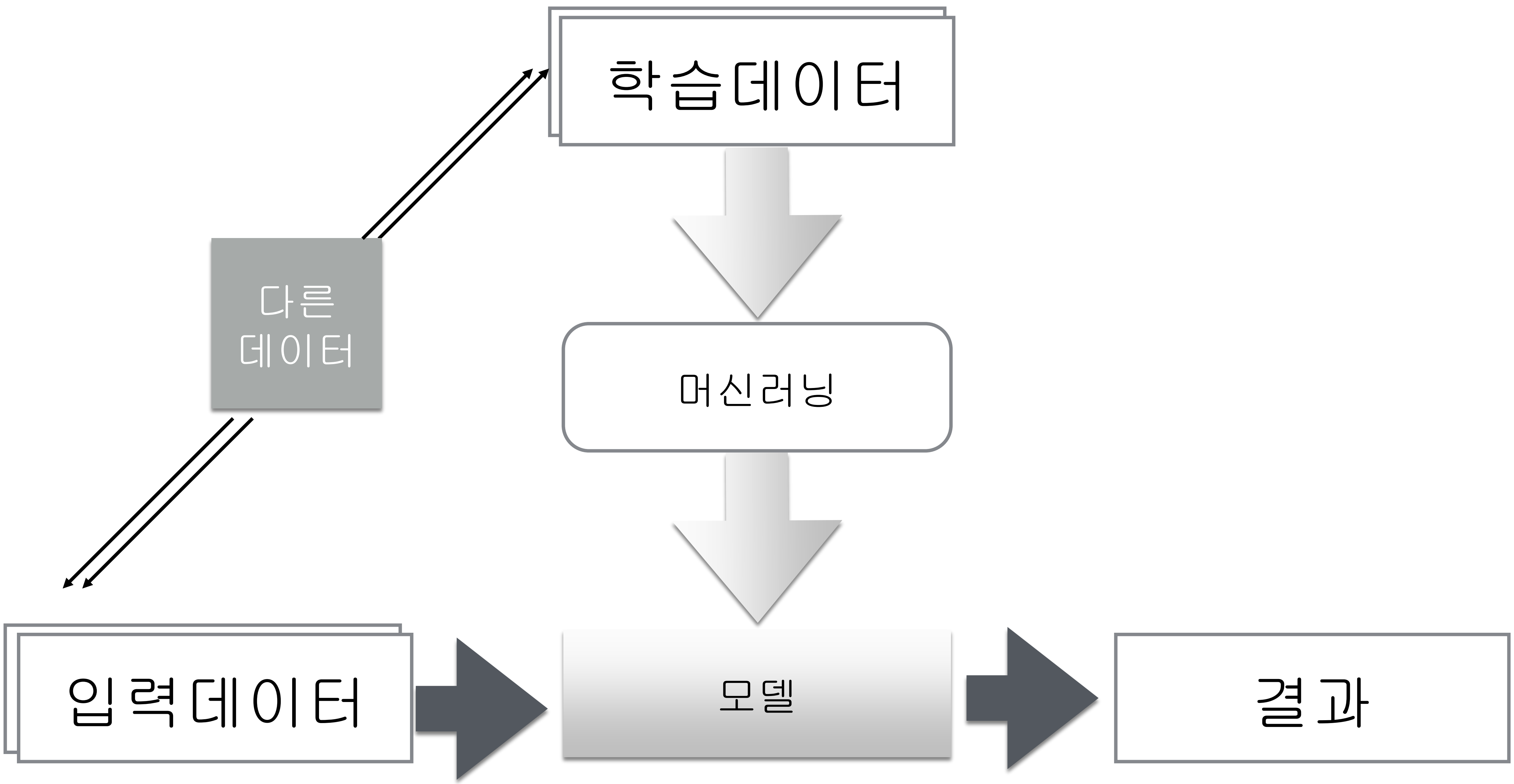
딥러닝 첫걸음

1. 머신러닝

인공지능, 머신러닝, 딥러닝

머신러닝



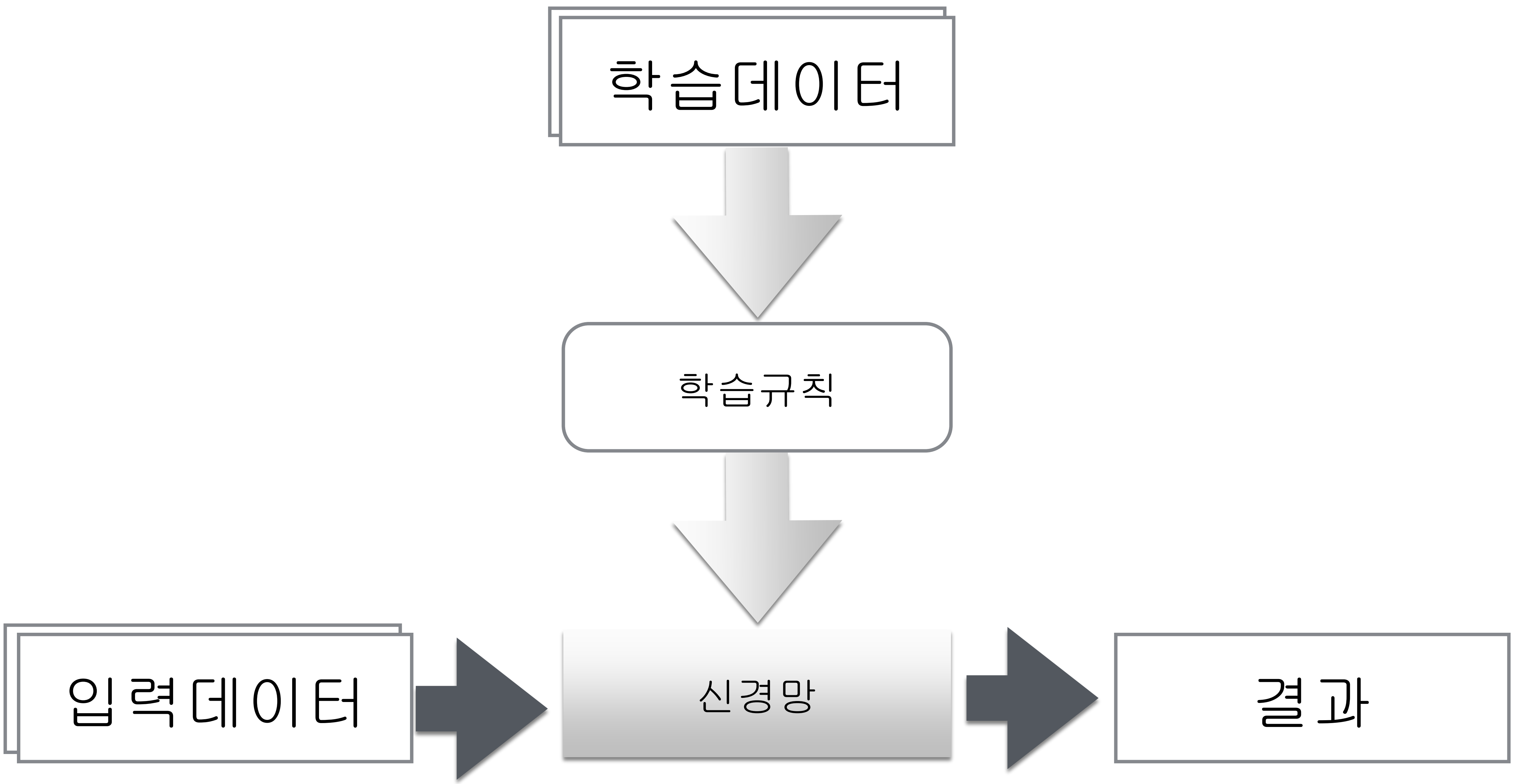
과적합, 정칙화, 검증

- 과적합(Overfitting) : Sample 추정에 지나치게 특화되어
  새로운 Sample 추정 성과가 나빠지는 현상
  - 지나치게 많은 설명변수를 사용할 경우 자주 발생
- 정칙화(Regularization) : 설명변수의 수를 줄이려는 시도
- 검정(Validation): Sample 을 추정용(train)/ 검정용(test) 으로 나누고, 추정용으로 학습한 모형의 성과를 검정용으로
  점검

머신러닝 종류

- 학습 방식에 따라: 지도학습, 비지도학습, 강화학습
  - 지도학습: 정답이 있는 입력자료 사용하여 모형의 결과와 정답간의 차이를 줄이는 학습
  - 비지도학습: 정답이 없는 입력자료를 사용하여 패턴에 따라 분류하는 학습
  - 강화학습:?
- 모델의 쓰임새?: 회귀(Regression), 분류(Categorization)
  - 회귀: 추세예측/분류: 범주예측

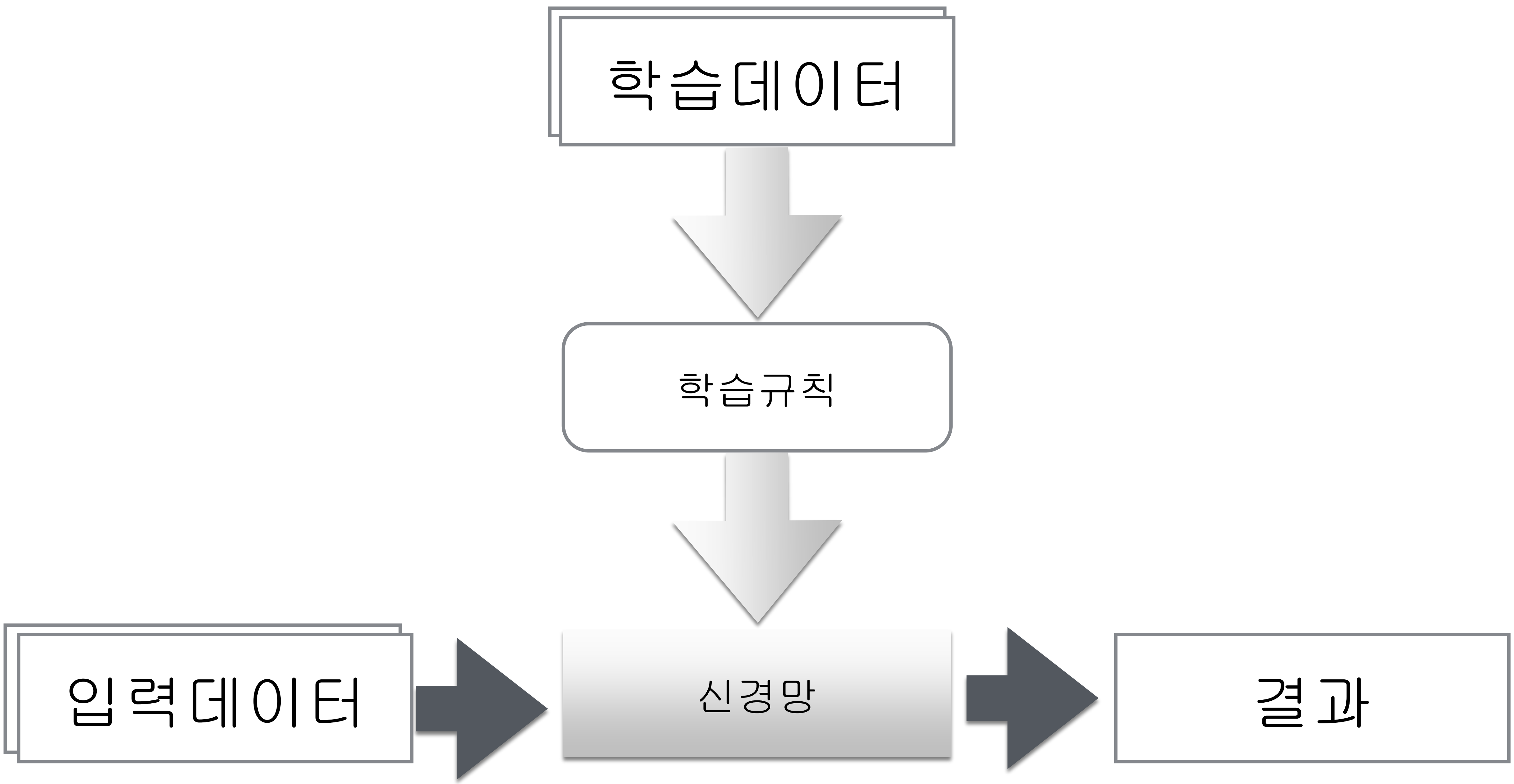
딥러닝



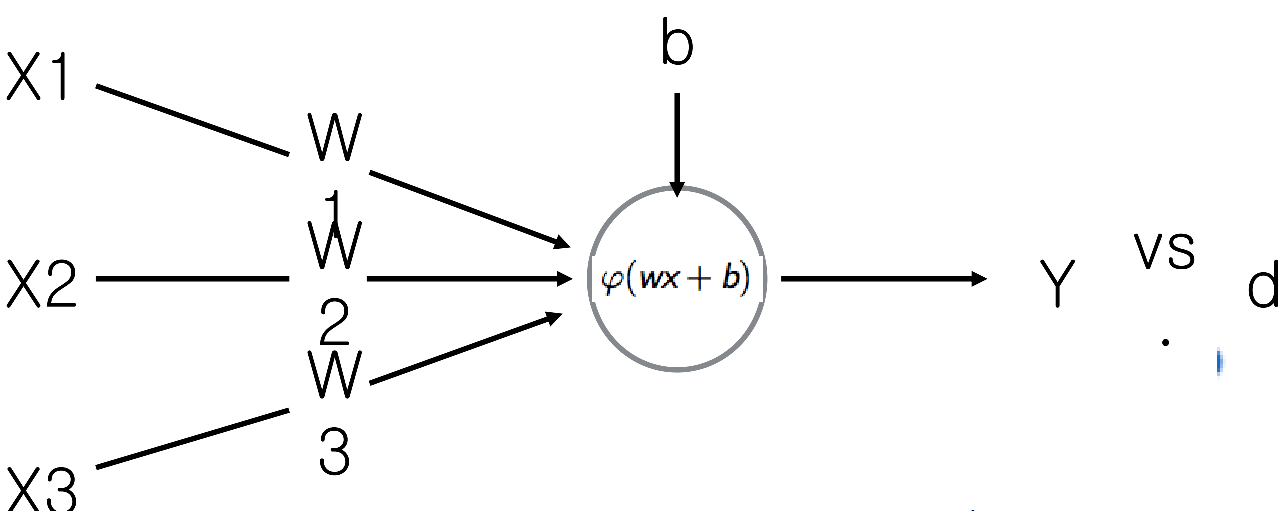
딥러닝 첫걸음

2. 신경망

딥러닝



Node



Activation function: $\varphi(v) = v, \quad \frac{1}{1+exp(-v)}, \ldots$

$$\begin{aligned}
\text{weight}: w &= [w_1, w_2, w_3] \\
\text{input}: x &= [x_1, x_2, x_3]^T \\
\\
\text{weighted sum}: v &= w_1x_1 + w_2x_2 + w_3x_3 + b \\
& wx + b \\
\text{output}: y &= \varphi(v) = \varphi(wx+b) \\
\text{Activation function}: \varphi(v) &= v, \quad \frac{1}{1+exp(-v)}, \ldots
\end{aligned}$$

신경망 지도학습

1. 가중치 초기화

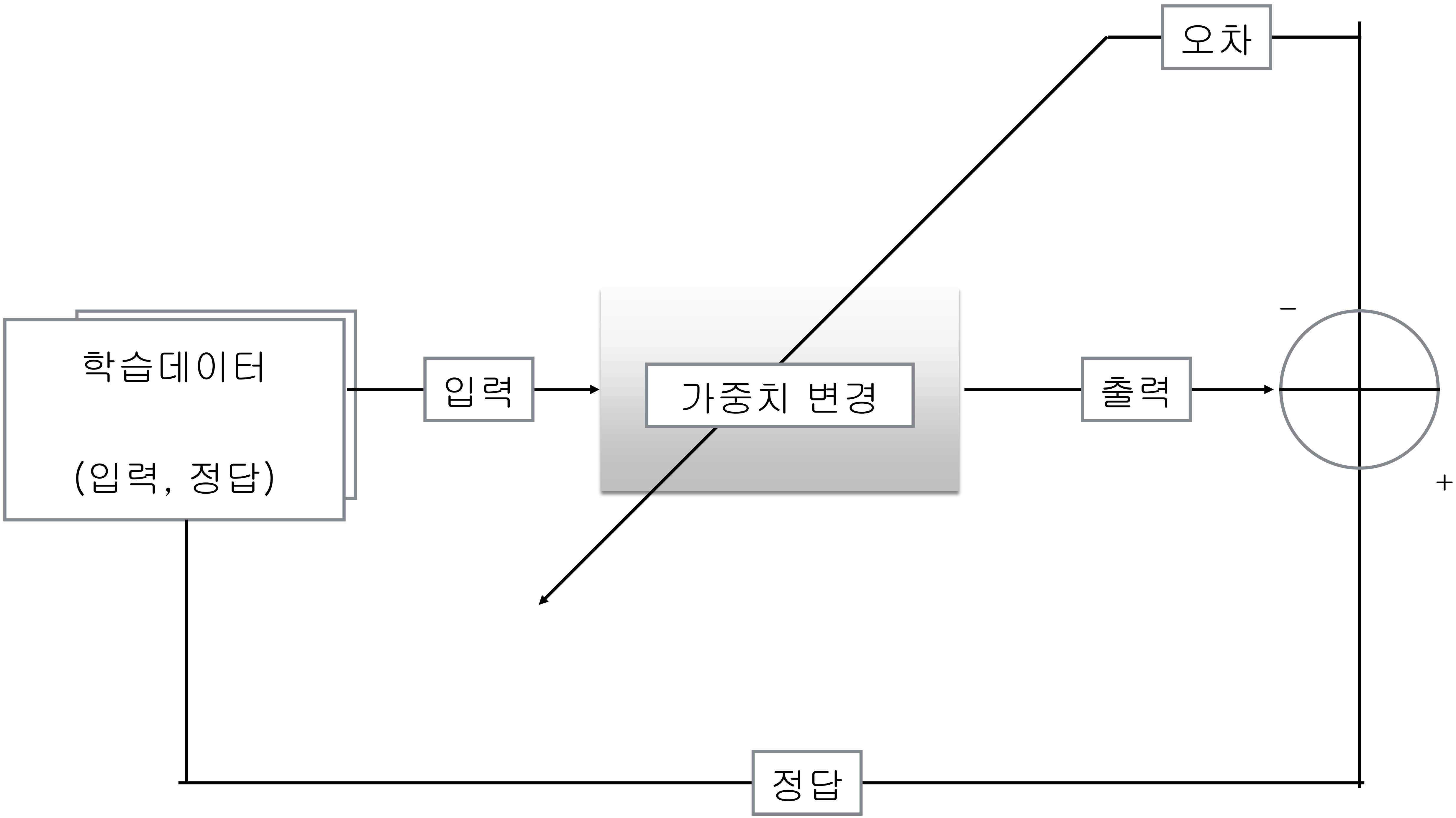
2. 입력 데이터 => 결과

3. 오차 = 결과 - 정답

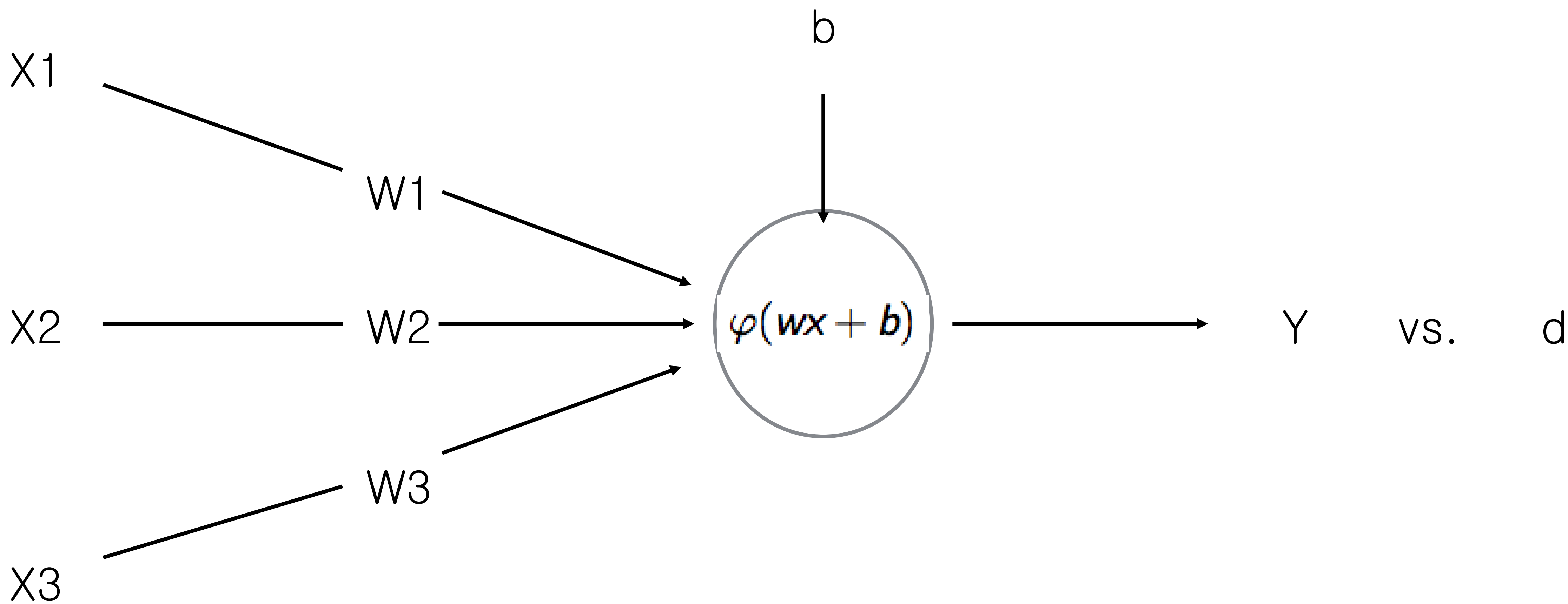
4. 가중치 조정 : 오차 축소

5. 2-4. 반복

신경망 지도학습

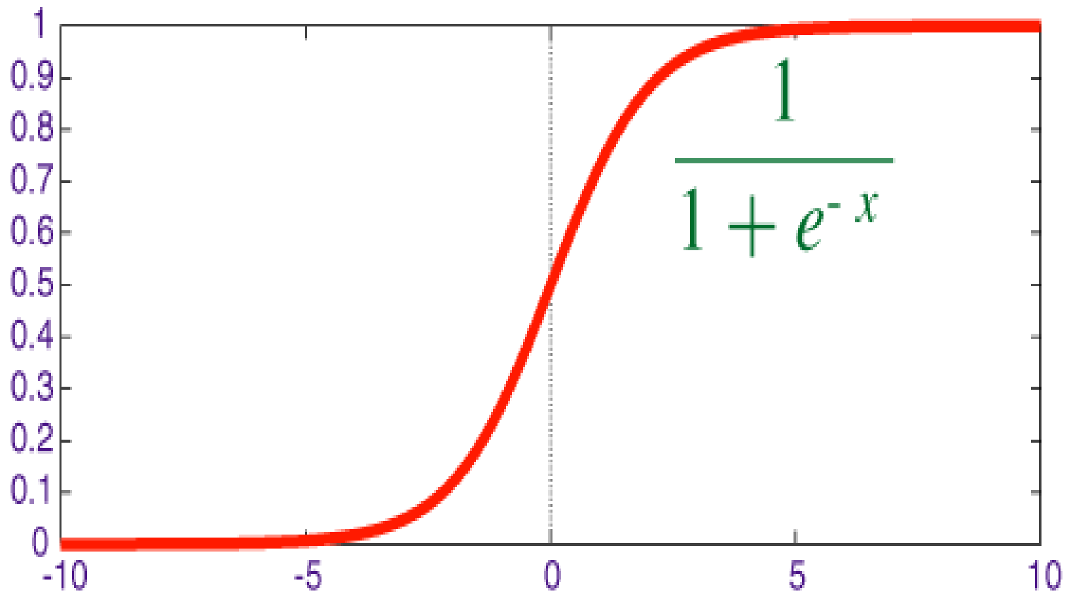


단층 신경망



Activation function: $\varphi(v) \;=\; v, \;\; \frac{1}{1 + exp(-v)}, \ldots$

Sigmoid function



$$
\begin{aligned}
\varphi(v) &= \frac{1}{1+exp(-v)} \\
\varphi'(v) &= -1 \times \frac{-exp(-v)}{(1+exp(-v))^2} = \times \frac{exp(-v)}{(1+exp(-v))^2} \\
&= \frac{1}{1+exp(-v)} \times \frac{exp(-v)}{1+exp(-v)} \\
&= \frac{1}{1+exp(-v)} \times \left[1 - \frac{1}{1+exp(-v)}\right] \\
&= \varphi(v)(1-\varphi(v))
\end{aligned}
$$

단층신경망 학습: 델타 규칙

1. 가중치 초기화:

$$w^0 = [w_{11}^0, w_{12}^0, w_{13}^0]$$

2. 입력데이터 $\rightarrow$ 결과:

$$\begin{aligned}
x &= [x_1, x_2, x_3]^T \\
v_1^0 &= w_{11}^0 x_1 + w_{12}^0 x_2 + w_{13}^0 x_3 + b^0 = w^0 x + b^0 \\
y_1^0 &= \varphi(v^0) = \varphi(w^0 x + b^0)
\end{aligned}$$

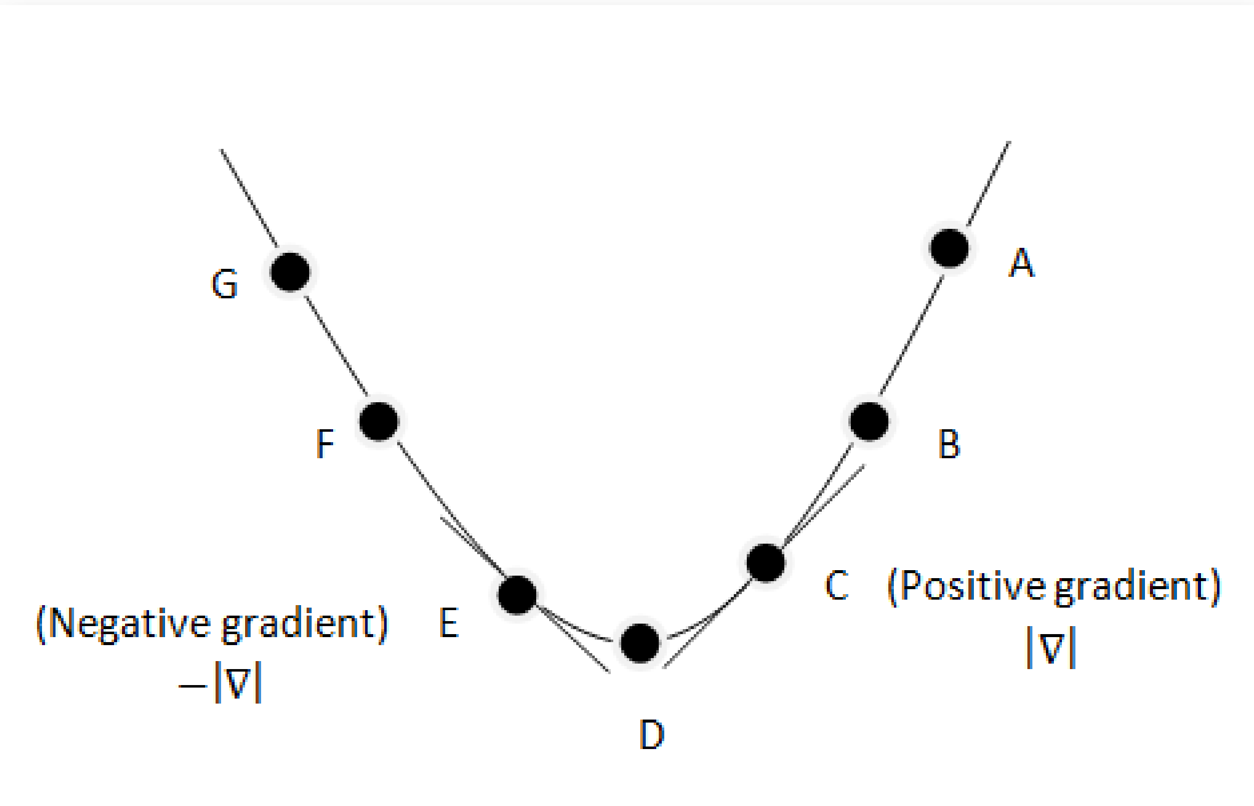
3. 오차 = 결과 - 정답

$$e_1^0 = d_1 - y_1^0$$

4. 가중치 조정: 오차 축소

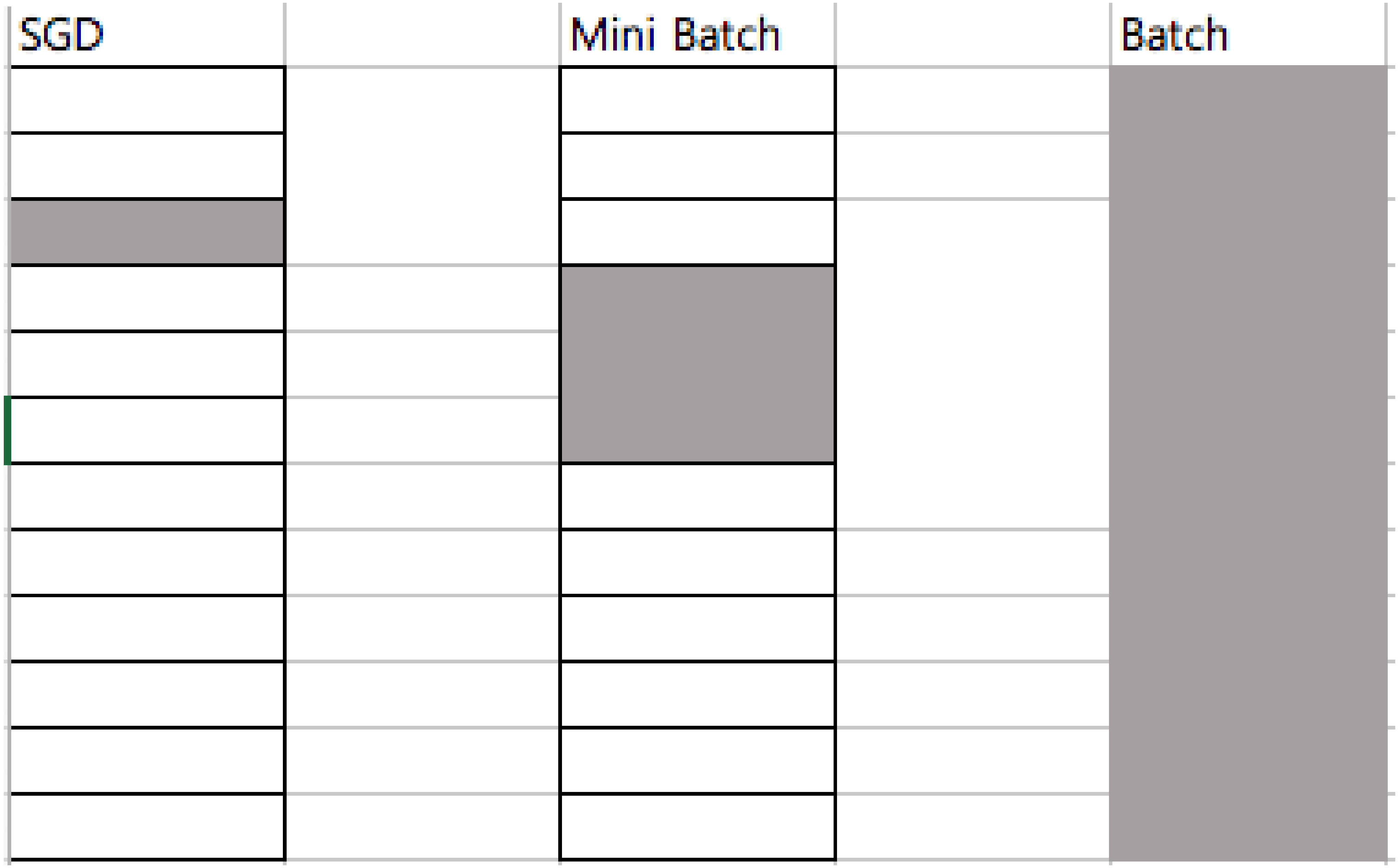
$$\begin{aligned}
\delta_1^0 &= \varphi'(v_1^0) e_1^0 = e_1^0 \qquad (\varphi'(v_1^0) = 1) \\
w_{11}^1 &= w_{11}^0 + \alpha \delta_1^0 x_1 \\
w_{12}^1 &= w_{11}^0 + \alpha \delta_1^0 x_2 \\
w_{13}^1 &= w_{11}^0 + \alpha \delta_1^0 x_3
\end{aligned}$$

Gradient Descent



$$
\begin{aligned}
J = (d-y)^2 &= (d-\varphi(v))^2 = (d-\varphi(wx))^2 \\
\frac{\partial J}{\partial w_i} &= -2(d-\varphi(wx))\varphi'(wx)x_i \\
&= -2e\varphi'(wx)x_i = -2\delta x_i \\
\alpha\delta x_i &\propto -\frac{\partial J}{\partial w_i}
\end{aligned}
$$

SGD vs. Mini Batch v. batch



training function

```
function W = Deltasgd (W,X,D)
alpha=0.9;
N=4;
  for k=1:N

2. 입력데이터 -> 결과
  x=X(k,:)';
  d=D(k);
 v=W*x;
 y=Sigmoid(v);

3. 오차 = 결과 - 정답
 e=d-y;

4. 가중치 조정 ( k loop 안에서)
 delta=y*(1-y)*e;
 dW=alpha*delta*x;
  W(1)=W(1)+dW(1);
  W(2)=W(2)+dW(2);
  W(3)=W(3)+dW(3);
 end
end
```

```
function W = Deltabatch(W,X,D)
alpha=0.9;
dWSum=zeros(3,1);

N=4;
 for k=1:N

2. 입력데이터 -> 결과
  x=X(k,:)';
  d=D(k);
  v=W*x;
 y=Sigmoid(v);

3. 오차 = 결과 - 정답
 e=d-y;

4. 가중치 조정
 delta=y*(1-y)*e;
 dW=alpha*delta*x;
 dWSum=dWSum+dW;
 end
(가중치 조정은  k 루프가 끝난 다음)
 dWAvg=dWSum/N;

  W(1)=W(1)+dWAvg(1);
  W(2)=W(2)+dWAvg(2);
  W(3)=W(3)+dWAvg(3);
 end
```

Sigmoid function: Sigmoid.m

```
function y=Sigmoid(x)

 y=1./(1+exp(-x));

end
```

SGD, batch training function

```
function W = Deltasgd (W,X,D)
% setting learning rate
 alpha=0.9;
% applying delta method for each
observation
 N=4;
  for k=1:N
%% input and outputdata on kth
observation
  x=X(k,:)';
  d=D(k);
%% obtain weighted sum of input
  v=W*x;
%% obtain output
  y=Sigmoid(v);
%% obtain error
  e=d-y;
%% obtain delta =d of activation function *
error
  delta=y*(1-y)*e;
%% obtain adjustment term
  dW=alpha*delta*x;
%% update weight: add adjustment terms to
weight.
  W(1)=W(1)+dW(1);
  W(2)=W(2)+dW(2);
  W(3)=W(3)+dW(3);
 end
end
```

```
function W = Deltabatch(W,X,D)
% setting learning rate
 alpha=0.9;
%set up a variable to save sum of weight adjustments
 dWSum=zeros(3,1);
%applying delta method one time for all sample
 N=4;
%% input and outputdata on kth observation
  for k=1:N
  x=X(k,:)';
  d=D(k);
%% obtain weighted sum of input
  v=W*x;
%% obtain output
  y=Sigmoid(v);
%% obtain error
  e=d-y;
%% obtain delta =d of activation function * error
  delta=y*(1-y)*e;
%% obtain weight adjustment term
  dW=alpha*delta*x;
%% add up weight adjustment term
  dWSum=dWSum+dW;
 end
% take average of sum of weight adjustment
  dWAvg=dWSum/N;
% weight adjustment
  W(1)=W(1)+dWAvg(1);
  W(2)=W(2)+dWAvg(2);
  W(3)=W(3)+dWAvg(3);
 end
```

TestDelat**.m (학습)

```
%Clearing memory
clear all
%Declaring input and output data
 X=[0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1, 1;];
 D=[0,0,1,1];
%initialize weight
 W=2*rand(1,3)-1;
%train model
 for epoch=1:10000
 W=Deltasgd(W,X,D);
 %W=DeltasgdM(W,X,D);
 end
%obtain estimate
 y=Sigmoid(X*W')
 %N=4;
 %for k=1:N
 %x=X(k,:)';
 %v=W*x;
 %y=Sigmoid(v)
 %end
```

```
clear all
 X=[0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1,
1;];
 D=[0,0,1,1];
 rand('seed',1);
 W=2*rand(1,3)-1;

 for epoch=1:1000000
 %W=Deltasgd(W,X,D);
 %W=Deltabatch(W,X,D);
 W=DeltabatchM(W,X,D);
 end

y=Sigmoid(X*W')
 %N=4;
 %for k=1:N
 %x=X(k,:)';
 %v=W*x;
 %y=Sigmoid(v)
 %end
```

실습 1. Matlab => R

- Sigmoid.m
- Deltasgd.m
- Deltabatch.m
- Testdeltabatch.m
- Testdeltasgd.m
- .m 에서 과업 특성 파악 => 과업 list 작성 => R code

실습 2. Matlab => R

- SGD와 Batch 비교
  - SGDvsBatch.m => SGDvsBatch. R

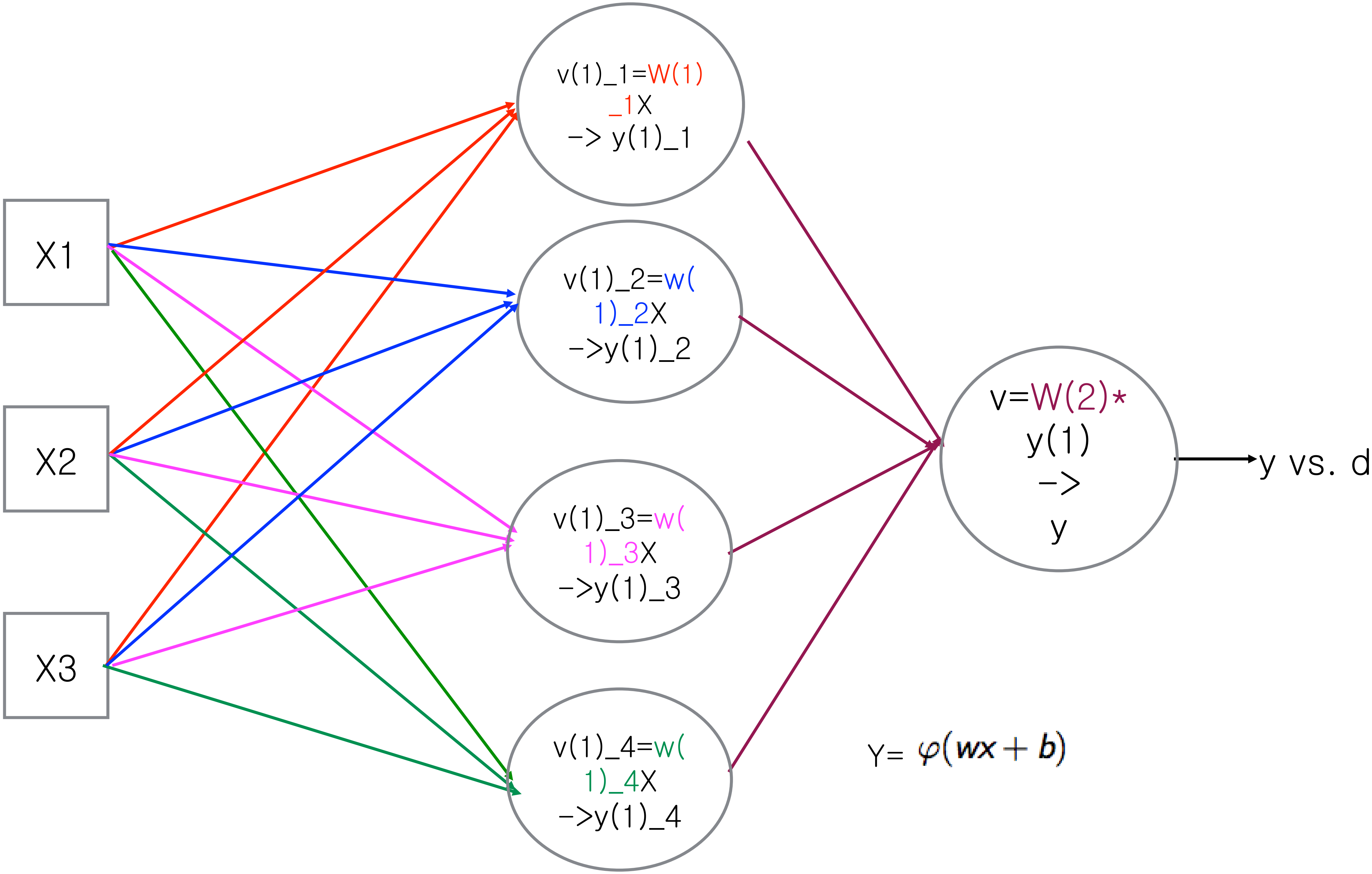
SGDvsBatch.m

```
 clear all

X=[0 0 1;
   0 1 1;
   1 0 1;
   1 1 1;
   ];

 D=[0
    0
    1
    1];

E1=zeros(1000,1);
E2=zeros(1000,1);

W1=2*rand(1,3)-1;
W2=W1;

for epoch = 1:10000 %train
  W1=Deltasgd(W1, X, D);
  W2=Deltabatch(W2, X, D);

  es1=0;
  es2=0;
  N=4;

for k =1:N
   x=X(k,:)';
   d=D(k);

   v1=W1*x;
   y1=Sigmoid(v1);
   es1=es1+(d-y1)^2;

   v2=W2*x;
   y2=Sigmoid(v2);
   es2=es2+(d-y2)^2;
  end

  E1(epoch)=es1/N;
  E2(epoch)=es2/N;

end


 plot(E1,'r')
hold on
plot (E2,'b:')
xlabel( 'Epoch')
ylabel('Ave Training Error')
legend('SGD','Batch')
```

딥러닝 첫걸음

3. 다층 신경망의 학습

다층신경망

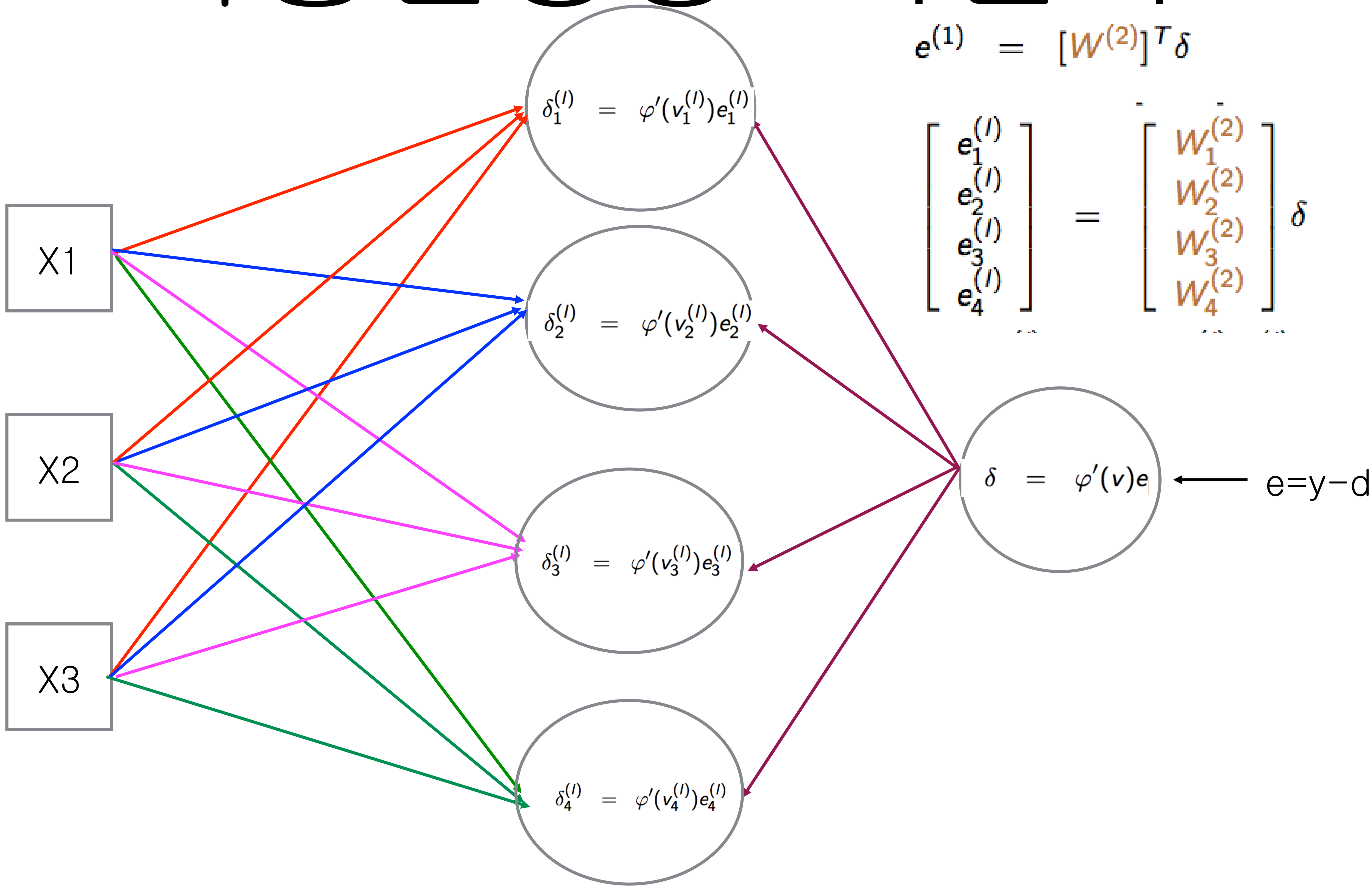


Y= $\varphi(\boldsymbol{wx}+\boldsymbol{b})$

다층신경망: 역전파

$$e^{(1)} = [W^{(2)}]^T \delta$$

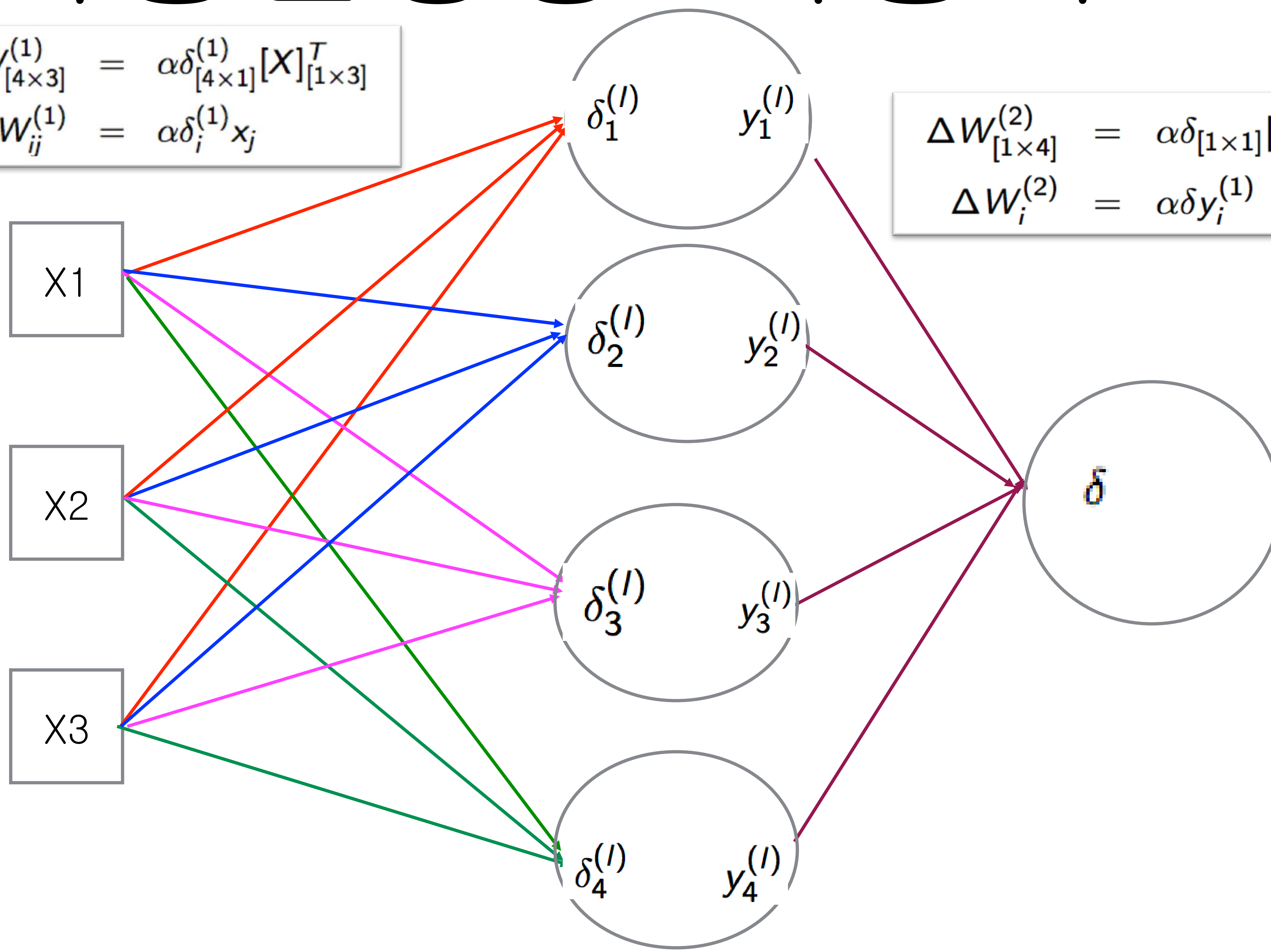
$$\begin{bmatrix} e_1^{(l)} \\ e_2^{(l)} \\ e_3^{(l)} \\ e_4^{(l)} \end{bmatrix} = \begin{bmatrix} W_1^{(2)} \\ W_2^{(2)} \\ W_3^{(2)} \\ W_4^{(2)} \end{bmatrix} \delta$$



다층신경망: 가중치조정

$$\Delta W^{(1)}_{[4\times 3]} = \alpha \delta^{(1)}_{[4\times 1]} [X]^T_{[1\times 3]}$$
$$\Delta W^{(1)}_{ij} = \alpha \delta^{(1)}_i x_j$$

$$\Delta W^{(2)}_{[1\times 4]} = \alpha \delta_{[1\times 1]} [y^{(1)}]^T_{[1\times 4]}$$
$$\Delta W^{(2)}_i = \alpha \delta y^{(1)}_i$$



다층신경망 학습

STEP 1 가중치 초기화

$$
W^{(1)}_{[4\times 3]} = \begin{bmatrix} w^{(1)}_{11} & w^{(1)}_{12} & w^{(1)}_{13} \\ w^{(1)}_{21} & w^{(1)}_{22} & w^{(1)}_{23} \\ w^{(1)}_{31} & w^{(1)}_{32} & w^{(1)}_{33} \\ w^{(1)}_{41} & w^{(1)}_{42} & w^{(1)}_{43} \end{bmatrix} \qquad W^{(2)}_{[1\times 4]} = [w^{(2)}_1, w^{(2)}_2, w^{(2)}_3, w^{(2)}_4]
$$

STEP 2 입력데이터 $\rightarrow$ 결과

1. 입력층

$$X_{[3\times 1]} = [x_1, x_2, x_3]^T$$

2. 은닉층

$$
\begin{aligned}
v^{(1)}_{[4\times 1]} &= W^{(1)}_{[4\times 3]} X_{[3\times 1]} \\
y^{(1)}_{[4\times 1]} &= \varphi(v^{(1)}) \\
v^{(l+1)}_{[1\times 1]} &= W^{(l+1)}_{[1\times 4]} y^{(l)}_{[4\times 1]} \\
y^{(l+1)}_{[1\times 1]} &= \varphi(v^{(l+1)})
\end{aligned}
$$

3. 출력층

$$
\begin{aligned}
v_{[1\times 1]} &= W^{(L)}_{[1\times 4]} y^{(L-1)}_{[4\times 1]} \\
y_{[1\times 1]} &= \varphi(v)
\end{aligned}
$$

다층신경망 학습 (2)

STEP3 오차, δ

1. 출력층

$$\begin{aligned} e &= y - d \\ \delta &= \varphi'(v) e_{[1\times 1]} \end{aligned}$$

2. 은닉층

$$\begin{aligned} e^{(l)}_{[4\times 1]} &= [W^{(l+1)}]^T_{[4\times 1]} \delta^{(l+1)}_{[1\times 1]} \\ \delta^{(l)}_{[4\times 1]} &= Diag[\varphi'(v^{(l)})_{[4\times 1]}]_{[4\times 4]} e^{(l)}_{[4\times 1]} \end{aligned}$$

STEP4 가중치조정

$$\begin{aligned} \Delta W^{(l)}_{i,j} &= \alpha \delta^{(l)}_i y^{(l-1)}_j \\ \Delta W^{(L)}_{[1\times 4]} &= \alpha \delta^{(L)}_{[1\times 1]} [y^{(L-1)}]^T_{[1\times 4]} \\ \Delta W^{(l)}_{[4\times 3]} &= \alpha \delta^{(l)}_{[4\times 1]} [X]^T_{[1\times 3]} \\ W'^{(l)} &= W^{(l)} + \Delta W^{(l)} \end{aligned}$$

training function: BackpropXOR.m

```
function [W1 W2] =
BackpropXOR(W1,W2, X,D)
alpha =0.9;
N=4;
for k=1:N
  % Load data. Single obs.
  x=X(k,:)';
  d=D(k);
```

2. 입력 데이터 => 결과

```
  % generate outputt
  %% Layer 1
  v1=W1*x;
  y1=Sigmoid(v1);
  %% Layer 2
  v=W2*y1;
  y=Sigmoid(v);
```

3. Delta, error

```
% generate  error and delta
  %% Layer 2
  e=d-y;
  delta=y.*(1-y).*e;
  %% Layer 1
  e1=W2'*delta;
  delta1=y1.*(1-y1).*e1;
```

4.가중치 조정

```
genrate weight adjustment
  %% Layer 1
  dW1 = alpha*delta1*x';
  W1=W1+dW1;
  %% Layer 2
  dW2 = alpha*delta*y1';
  W2=W2+dW2;
  end
End
```

TestbackpropXOR .m

```
clear all

X=[0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1, 1;];
 D=[0,1,1,0];

1. 가중치 초기화
W1=2*rand(4,3)-1;

 W2=2*rand(1,4)-1;

2. 학습

for epoch=1:1000000
 [W1 W2]=BackpropXOR(W1,W2,X,D);
 %W=DeltasgdM(W,X,D);
 end
 y1=Sigmoid(X*W1');
 y=Sigmoid(y1*W2')

3. 추정
 N=4;
 for k=1:N

 x=X(k,:)';
 v1=W1*x;
 y1=Sigmoid(v1);
 v=W2*y1;
 y=Sigmoid(v)
 end
```

실습 . Matlab => R

- TestbackpropXOR.m
- BoxpropXOR.m

- .m 에서 과업 특성 파악 => 과업 list 작성 => R code

딥러닝 첫걸음

3. 다층 신경망의 학습 - 수렴 속도 제고

수렴 속도 향상 방법

- Momentum : BackPropMmt.m, TestBackPropMmt.m
- Cross Entropy Loss function: BackPropCE.m, TestBackPropCE.m
- .m 에서 과업 특성 파악 => 과업 list 작성 => R code

Monentum: 가중치조정 방식

$$
\begin{aligned}
\Delta W_{i,j}^{(l)} &= \alpha \delta_i^{(l)} y_j^{(l-1)} \\
W_{i,j}'^{(l)} &= W_{i,j}^{(l)} + \Delta W_{i,j}^{(l)} \\
\Rightarrow & \\
\Delta W_{i,j}^{(l)} &= \alpha \delta_i^{(l)} y_j^{(l-1)} \\
m_{i,j}^{(l)} &= \Delta W_{i,j}^{(l)} + \beta m_{i,j,(-1)}^{(l)} \\
W_{i,j}'^{(l)} &= W_{i,j}^{(l)} + m_{i,j}^{(l)} \\
m_{i,j,(-1)}^{(l)} &= m_{i,j}^{(l)}
\end{aligned}
$$

실습 . Matlab => R

- TestbackpropMmt.m
- BaxpropMmt.m
- .m 에서 과업 특성 파악 => 과업 list 작성 => R code

BackpropMmt.m

```
function [W1 W2] = BackpropMmt(W1,W2,
X,D)
alpha =0.9;
beta=0.9;
mmt1=zeros(size(W1));
mmt2=zeros(size(W2));
N=4;
for k=1:N
  % Load data. Single obs.
  x=X(k,:)';
  d=D(k);

  % generate outputt
  %% Layer 1
  v1=W1*x;
  y1=Sigmoid(v1);
  %% Layer 2
  v=W2*y1;
  y=Sigmoid(v);
  % generate  error and delta
  %% Layer 2
  e=d-y;
  delta=y.*(1-y).*e;
  %% Layer 1
  e1=W2'*delta;
  delta1=y1.*(1-y1).*e1;
  % genrate weight adjustment
  %% Layer 1
  dW1 = alpha*delta1*x';
  mmt1=dW1+beta*mmt1;
  W1=W1+mmt1;
  %% Layer 2
  dW2 = alpha*delta*y1';
  mmt2=dW2+beta*mmt2;
  W2=W2+mmt2;
  end
end
```

TestbackpropMmt.m

```
clear all

X=[0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1, 1;];
 D=[0,1,1,0];
 W1=2*rand(4,3)-1;

 W2=2*rand(1,4)-1;
 %W1=0.0001*W1
 %W2=0.0001*W2
 for epoch=1:10000
 %[W1 W2]=BackpropXOR(W1,W2,X,D);
 [W1 W2]=BackpropMmt(W1,W2,X,D);
 %W=DeltasgdM(W,X,D);
 end
 y1=Sigmoid(X*W1');
 y=Sigmoid(y1*W2')

 N=4;
 for k=1:N

 x=X(k,:)';
 v1=W1*x;
 y1=Sigmoid(v1);
 v=W2*y1;
y=Sigmoid(v)
end
```

Cross entropy loss function

$$J = \frac{1}{2}(d-y)^2 \qquad \text{SSE: Sum of Squared Residual}$$

$$J = -d\ln(y) - (1-d)\ln(1-y) \qquad \text{CE:Cross Entropy}$$

- Cross Entropy를 사용하면 출력층의 delta 산출방식이 변화

$$\begin{aligned} e &= y - d \\ \delta &= \varphi'(v)e \end{aligned} \quad \Rightarrow \quad \begin{aligned} e &= y - d \\ \delta &= e \end{aligned}$$

$$
\begin{aligned}
J &= -d\ln(y) - (1-d)\ln(1-y) \qquad \text{CE:Cross Entropy} \\
\frac{\partial J}{\partial w_i} &= \left[-\frac{d}{y} - (1-d)\frac{-1}{1-y}\right]\frac{\partial y}{\partial w_i} \\
&= \left[-\frac{-d(1-y)+y(1-d)}{y(1-y)}\right]\frac{\partial y}{\partial w_i} = -\left[\frac{y-d}{y(1-y)}\right]\frac{\partial y}{\partial w_i} \\
&= -\left[\frac{y-d}{y(1-y)}\right]\varphi'(wx)x_i \\
&= -\left[\frac{y-d}{y(1-y)}\right]y(1-y)x_i = -ex_i
\end{aligned}
$$

실습 . Matlab => R

- TestbackpropCE.m
- BackpropCE.m
- .m 에서 과업 특성 파악 => 과업 list 작성 => R code

실습. Matlab => R

- SSE와 CE 비교
  - CEvsSEE.m => CEvsSEE. R

BackpropCE.m

```
function [W1 W2] = BackpropCE(W1,W2,
X,D)
alpha =0.9;
N=4;
for k=1:N
  % Load data. Single obs.
  x=X(k,:)';
  d=D(k);

  % generate outputt
  %% Layer 1
  v1=W1*x;
  y1=Sigmoid(v1);
  %% Layer 2
  v=W2*y1;
  y=Sigmoid(v);
  % generate  error and delta
  %% Layer 2
  e=d-y;
  delta=e;
  %% Layer 1
  e1=W2'*delta;
  delta1=y1.*(1-y1).*e1;
  % genrate weight adjustment
  %% Layer 1
  dW1 = alpha*delta1*x';
  W1=W1+dW1;
  %% Layer 2
  dW2 = alpha*delta*y1';
  W2=W2+dW2;
  end
end
```

TestBackpropCE.m

```
clear all

X=[0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1, 1;];
 D=[0,1,1,0];
 W1=2*rand(4,3)-1;
 W2=2*rand(1,4)-1;

 for epoch=1:10000
 %[W1 W2]=BackpropXOR(W1,W2,X,D);
 [W1 W2]=BackpropCE(W1,W2,X,D);
 %W=DeltasgdM(W,X,D);
 end
 %y=Sigmoid(X*W')
 N=4;
 for k=1:N
 x=X(k,:)';
 v1=W1*x;
 y1=Sigmoid(v1);
 v=W2*y1;
 y=Sigmoid(v)
end
```

실습. Matlab => R

- SSE와 CE 비교
  - CEvsSEE.m => CEvsSEE. R

CEvsSEE.m

```
clear all

X=[0 0 1;
   0 1 1;
   1 0 1;
   1 1 1;
   ];

 D=[0
    0
    1
    1];

E1=zeros(1000,1);
E2=zeros(1000,1);

W11=2*rand(4,3)-1;%Cross Entropy
W12=2*rand(1,4)-1;

W21=W11;%SSE
W22=W12;


for epoch = 1:10000 %train
  [W11, W12]=BackpropCE(W11,W12,X, D);
  [W21, W22]=BackpropXOR(W21,W22, X, D);

  es1=0;
  es2=0;
  N=4;
```

```
 for k =1:N
  x=X(k,:)';
  d=D(k);

  v1=W11*x;
  y1=Sigmoid(v1);
  v=W12*y1;
  y=Sigmoid(v);
  es1=es1+(d-y)^2;

  v1=W21*x;
  y2=Sigmoid(v1);
  v=W22*y2;
  y=Sigmoid(v);
  es2=es2+(d-y)^2;
 end

 E1(epoch)=es1/N;
 E2(epoch)=es2/N;

end


 plot(E1,'r')
hold on
plot (E2,'b:')
xlabel( 'Epoch')
ylabel('Ave Training Error')
legend('Cross Entropy','SSE')
```