

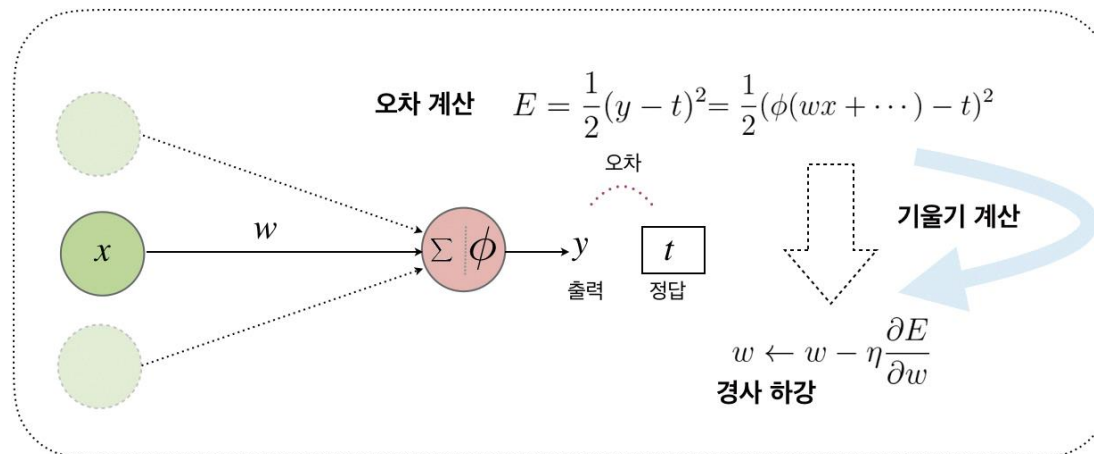


# 인공 신경망 기초

## Part3 오류역전파로 다층 퍼셉트론 학습

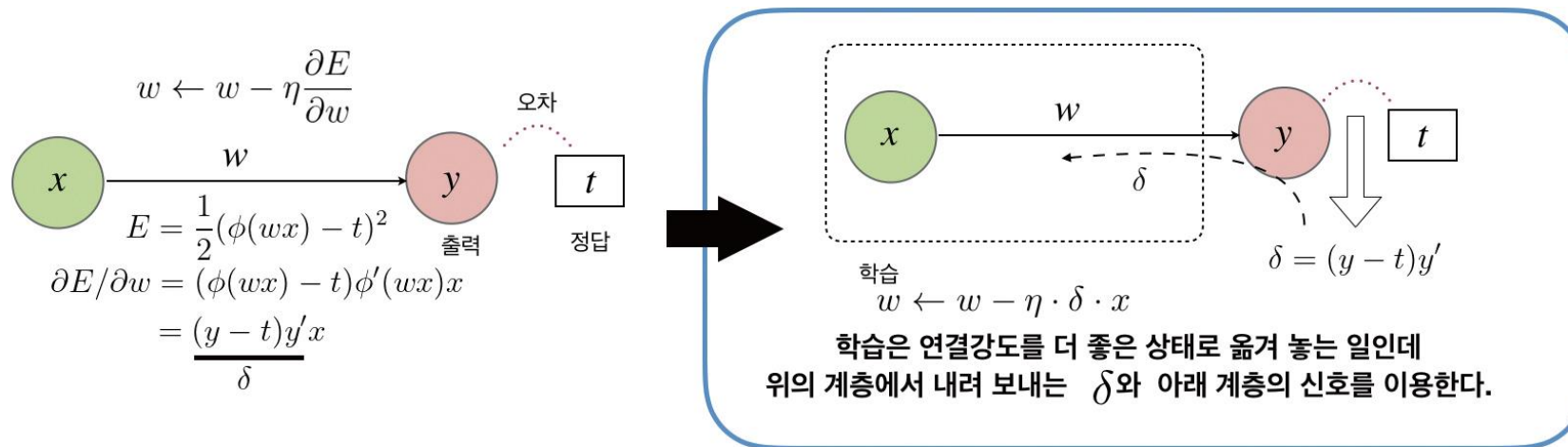
# 경사 하강법에 의한 연결강도 개선

- 인공 신경망에서 연결강도를 조정해 오차를 줄여 보자
- 오차 곡면의 기울기를 연결강도에 대해 구한 뒤에 이 기울기를 따라 내려가는 **경사 하강법** gradient descent을 사용하여 구현할 수 있음
- $x$ 가 연결강도  $w$ 로 출력 노드에 연결되어 있다고 하자.
- 출력 노드는  $x$ 를 비롯한 여러 노드에서 신호를 수신하여 합산하는 부분과 이 값을 활성화함수에 넣어 최종 출력을 결정하는 부분으로 나뉨
- 출력과 정답(목표값)의 차이를 제공하여 오차를 구하는 일반적인 방법을 사용 가능
- 출력 노드의 출력값  $y$ 가  $\phi(wx + \dots)$ 이므로, 오차  $E$ 는  $1/2 (\phi(wx + \dots) - t)^2$

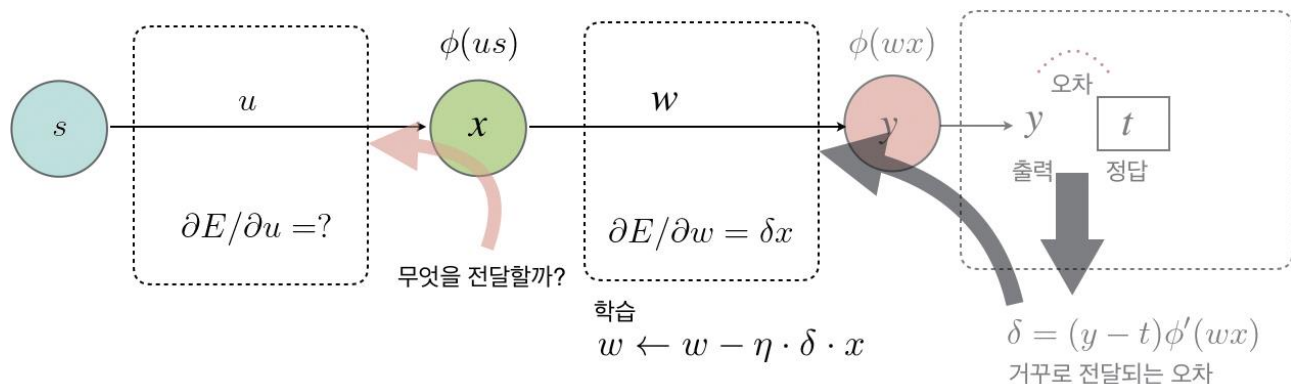


# 다층 퍼셉트론의 학습 - 오차를 아래로 전파하자

- 복잡한 문제를 해결하기 위해서는 일반적으로 신경망 연결을 여러 층으로 만들어야 하며 아래 그림과 같은 신경망이 연결된 구조에서 신호가  $x$ 를 거쳐  $y$ 로 전달되는데 이 방향을 **순전파** forward propagation라고 함
- 앞 절에서 오차를 줄이는 방향으로 연결강도를 변경하는 방법을 아래와 같이 출력 부분에서 확인할 수 있는 목표  $t$ 와 출력  $y$ 의 차이, 그리고 출력의 미분  $y'$ 가 연결망을 거꾸로 타고 전달되어 가중치를 조정하는 것으로 봄



- $s$ 를 거쳐  $x$ 를 통과한 다음 출력노드  $y$ 로 연결되는 다층 신경망을 고려해보자.
- 연결강도  $w$ 를 갱신하는 방법을 그 이전 단계에 있는 연결강도  $u$ 에도 전파할 수 있으며 이것이 **오차 역전파**<sup>error backpropagation</sup>의 기본 개념
  - 신경망 연구의 초창기에는 이 방법이 알려지지 않았기 때문에 민스키 등의 비판을 이겨낼 수 없었음



- $u$ 를 갱신하는 학습을 위해 우리는 오차 곡면의 기울기를  $u$ 기준으로 계산

$$\frac{\partial E}{\partial u} = \frac{\partial E}{\partial x} \cdot \frac{\partial x}{\partial u}$$

- $\partial E / \partial u$ 는 제곱 오차  $1/2 (\phi(wx) - t)^2$ 를  $w$ 가 아니라  $x$ 로 미분한 것이므로 앞에서 구한  $\partial E / \partial w$ 식에서  $w$ 와  $x$ 의 역할을 바꿈

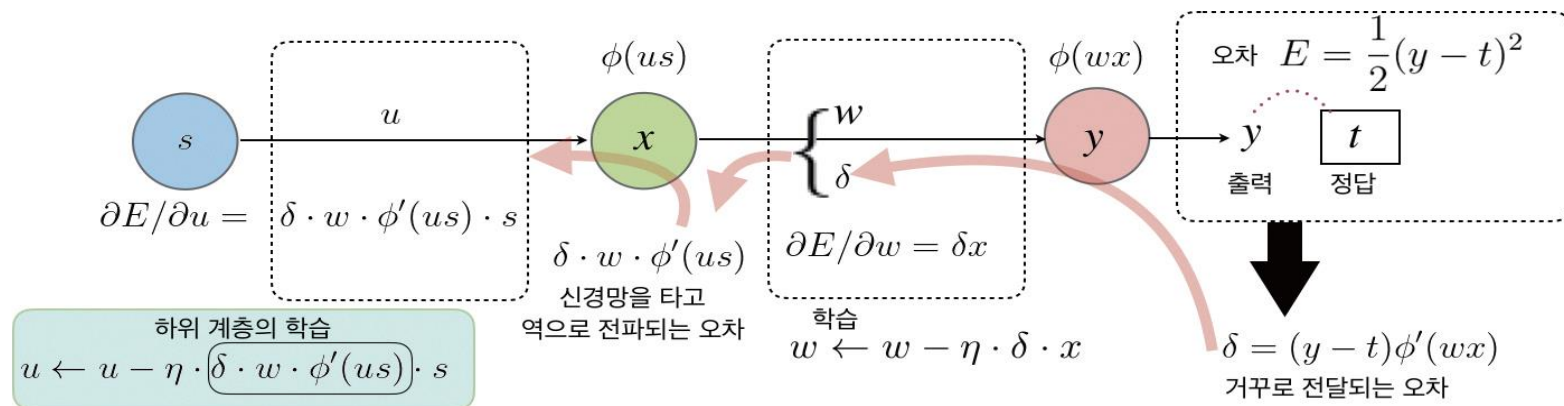
$$\partial E / \partial w = (y - t) y' x = \delta x$$

$$\partial E / \partial x = (y - t) y' w = \delta w$$

- 출력에 더 멀어진 단계의 가중치  $u$ 를 갱신하기 위한 오차의 미분을 계산하면  $x$ 값은 노드  $s$ 신호에 연결강도  $u$ 가 곱해진 뒤 활성화 함수를 통과한 결과인  $\phi(us)$ 이므로

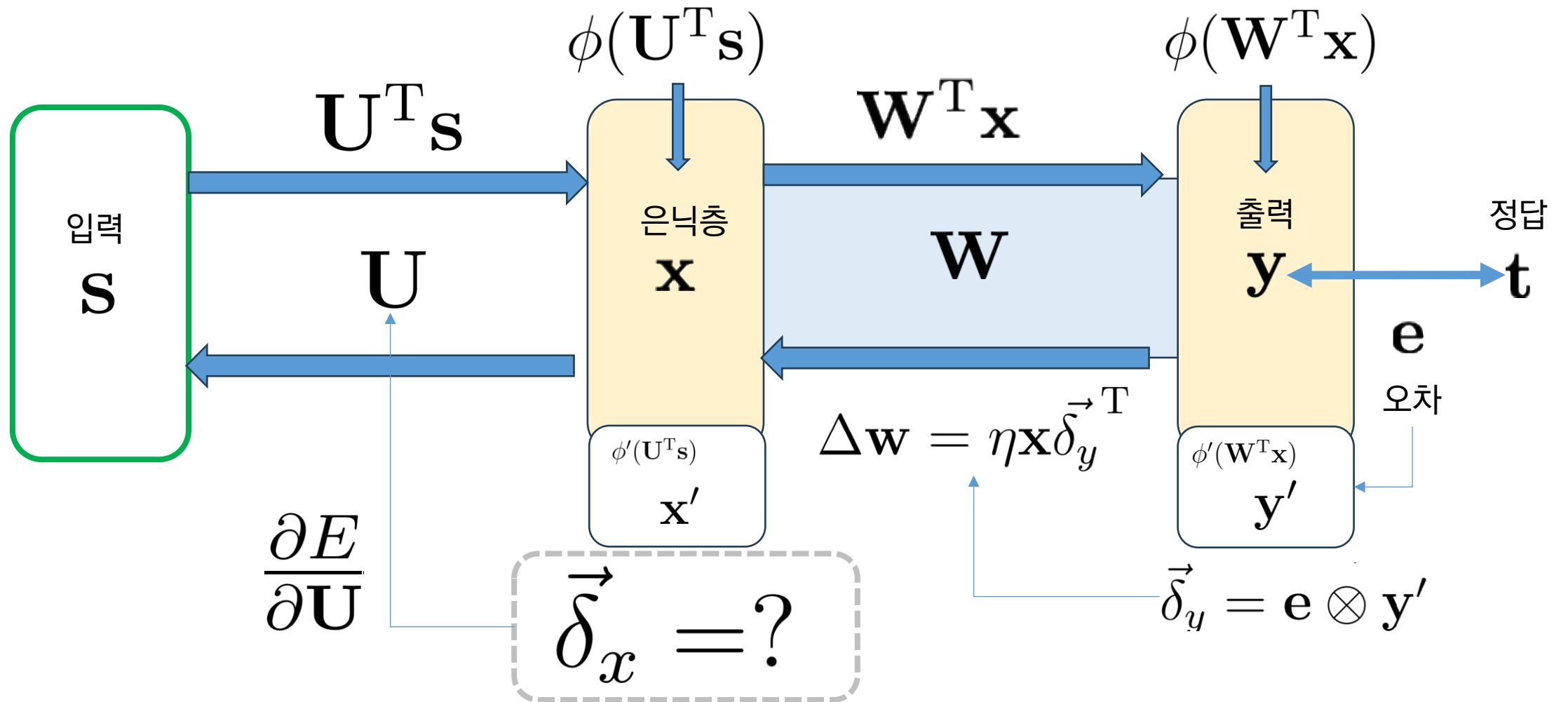
$$\frac{\partial E}{\partial u} = \frac{\partial E}{\partial x} \frac{\partial x}{\partial u} = \delta \cdot w \cdot \frac{\partial \phi(us)}{\partial u} = \delta \cdot w \cdot \phi'(us) \cdot s$$

- **순전파**forward propagation 단계에서는 입력의 신호가 연결강도로 증폭되고 활성화 함수를 통과하여 출력 노드로 전달되는 과정
- 학습 과정은 연결강도를 수정하는 것이기 때문에, 출력단의 오차가 역방향으로 **활성화 함수의 미분 함수**를 거친 뒤에 연결강도에 의해 증폭되어 입력단까지 전달되는 과정으로, 이러한 신경망 학습 방법을 **오차 역전파**error backpropagation 알고리즘



- 역전파 알고리즘은 1980년대 후반 발표
  - 데이비드 럼멜하트David Rumelhart, 제프리 힌튼Geoffrey Hinton, 로널드 윌리엄스Ronald Williams가 발표
- 이와 같은 아이디어는 폴 워보스Paul Werbos의 1974년도 박사학위 논문에서 제안됨
  - 순서가 있는 미분ordered derivative으로 제안된 것

- 오차를 한 층 더 내릴 수 있을까?



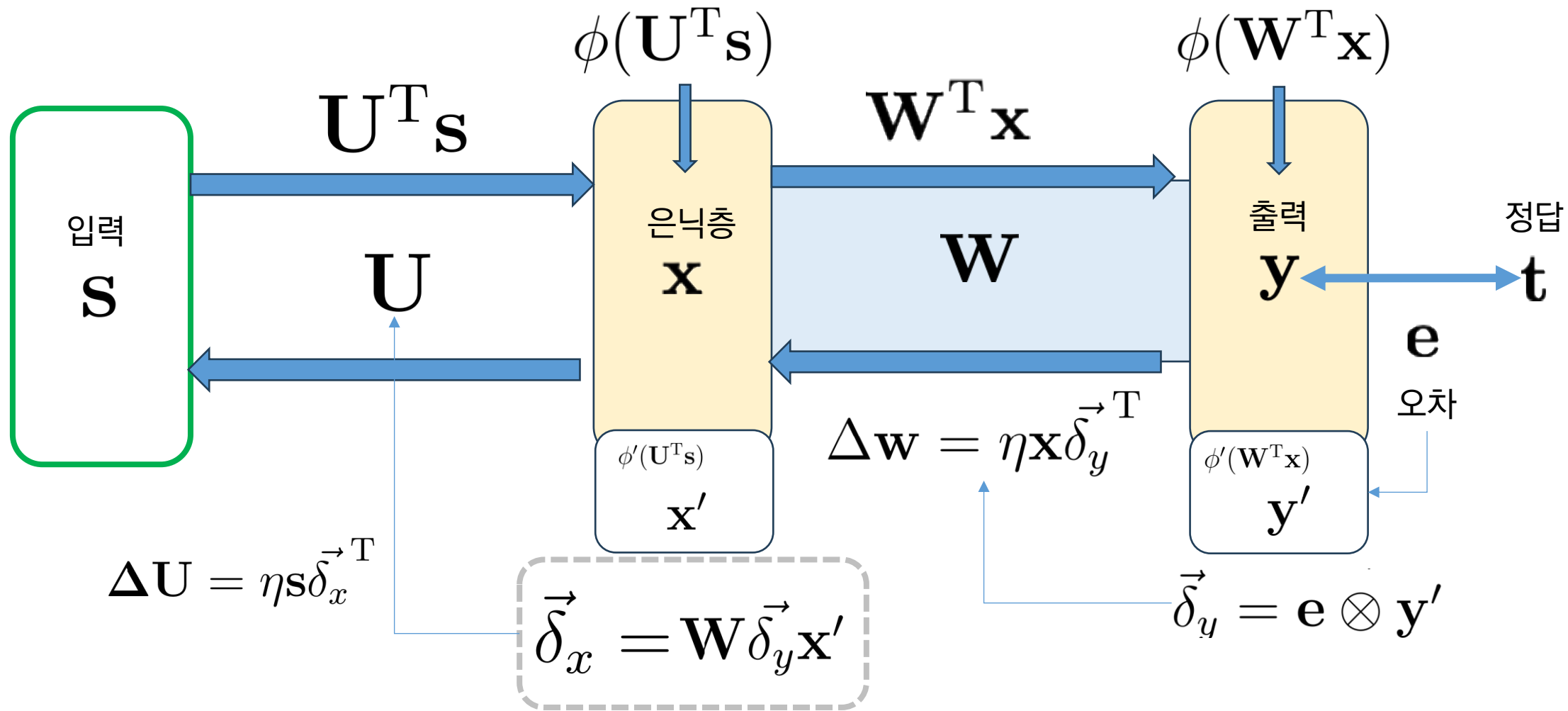
- 오차를 한 층 더 내릴 수 있을까?

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{x} \vec{\delta}^T \quad : \quad \frac{\partial E}{\partial \mathbf{x}} = \mathbf{W} \vec{\delta}$$

$$\frac{\partial E}{\partial \mathbf{U}} = \frac{\partial E}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{U}} = \frac{\partial E}{\partial \mathbf{x}} \frac{\partial \phi(\mathbf{U}^T \mathbf{s})}{\partial \mathbf{U}} = \mathbf{W} \vec{\delta} \phi'(\mathbf{U}^T \mathbf{s})_s$$



$$\frac{\partial E}{\partial \mathbf{U}} = \frac{\partial E}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{U}} = \frac{\partial E}{\partial \mathbf{x}} \frac{\partial \phi(\mathbf{U}^T \mathbf{s})}{\partial \mathbf{U}} = \boxed{\mathbf{W} \vec{\delta} \phi'(\mathbf{U}^T \mathbf{s})}_s$$



# Pseudo-code로 살펴보는 순전파 / 역전파

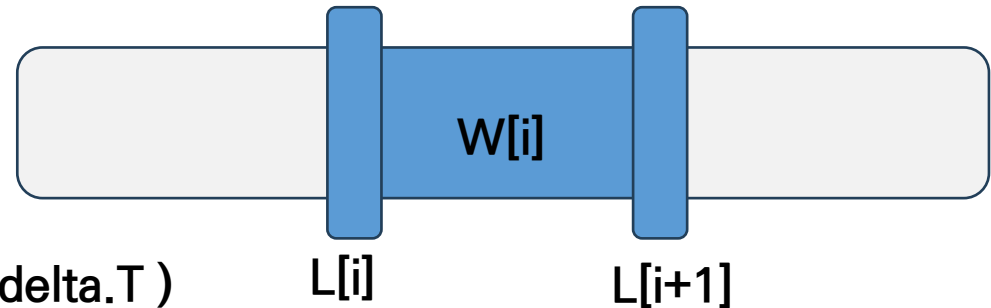
계층  $L[i]$ 와  $L[i+1]$  사이에  $W[i]$ 로 연결

(순전파)

```
For i in range(start, end):  
     $L[i+1] = \text{phi}(W[i].T @ L[i])$   
     $L\_derivative[i+1] = \text{phi\_derivative}(W[i].T @ L[i])$ 
```

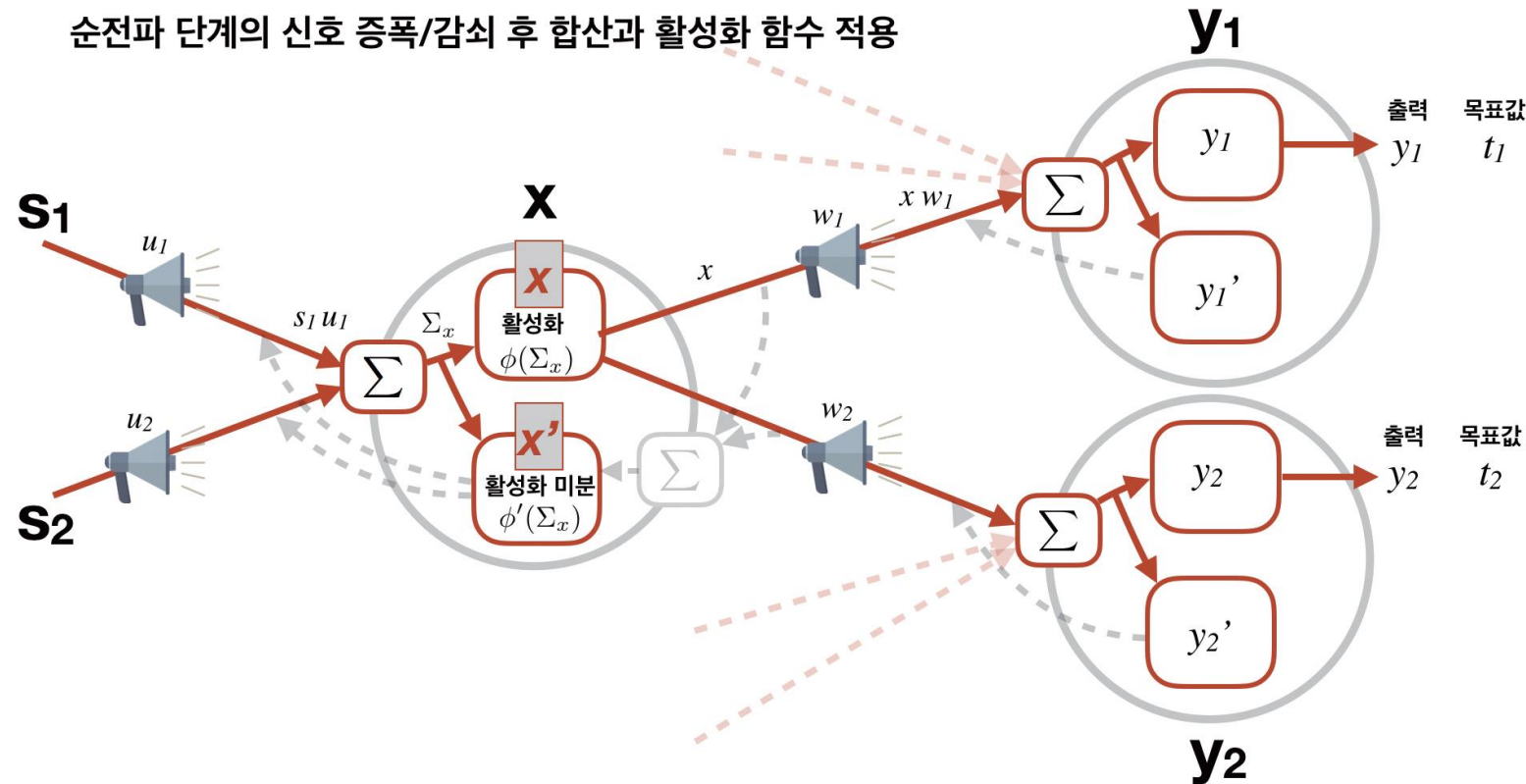
(역전파)

```
 $e = (L[\text{end}] - \text{target})$   
For i in range(end, start, -1):  
     $\text{delta} = e * L\_derivative[i]$   
     $W[i-1] += \text{learning\_rate} * (L[i-1] @ \text{delta}.T)$   
     $e = W[i-1] @ \text{delta}$ 
```

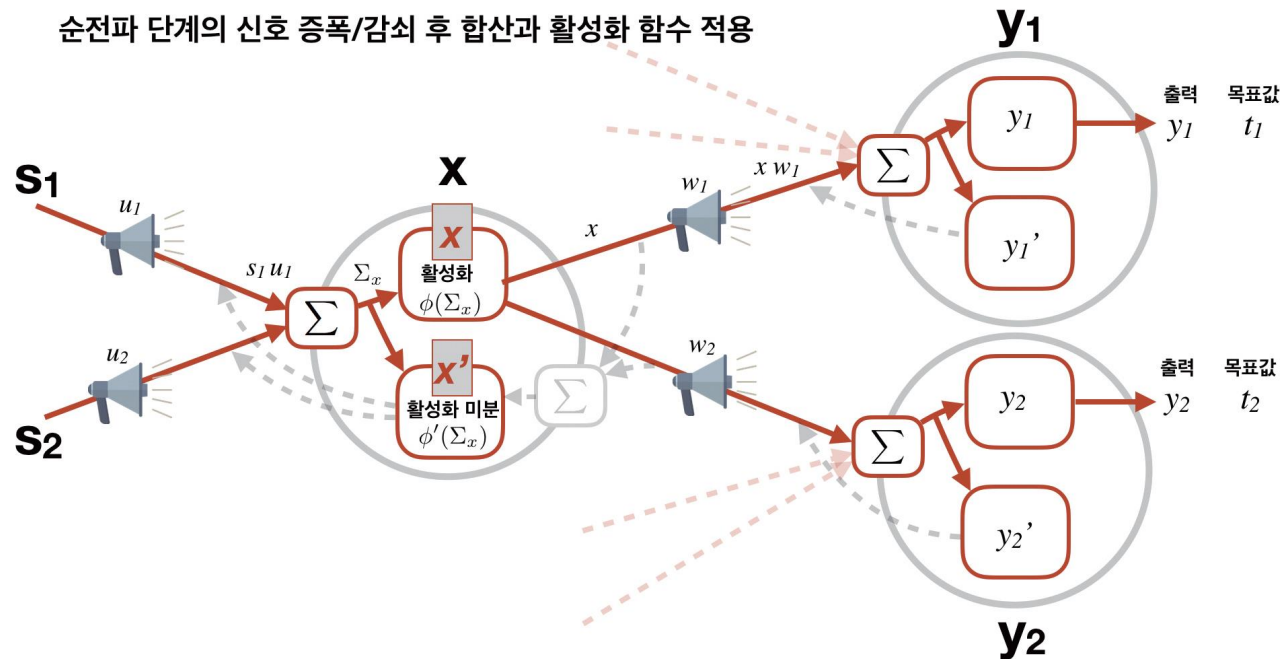


## 7.8 역전파 알고리즘에 대한 직관적 이해

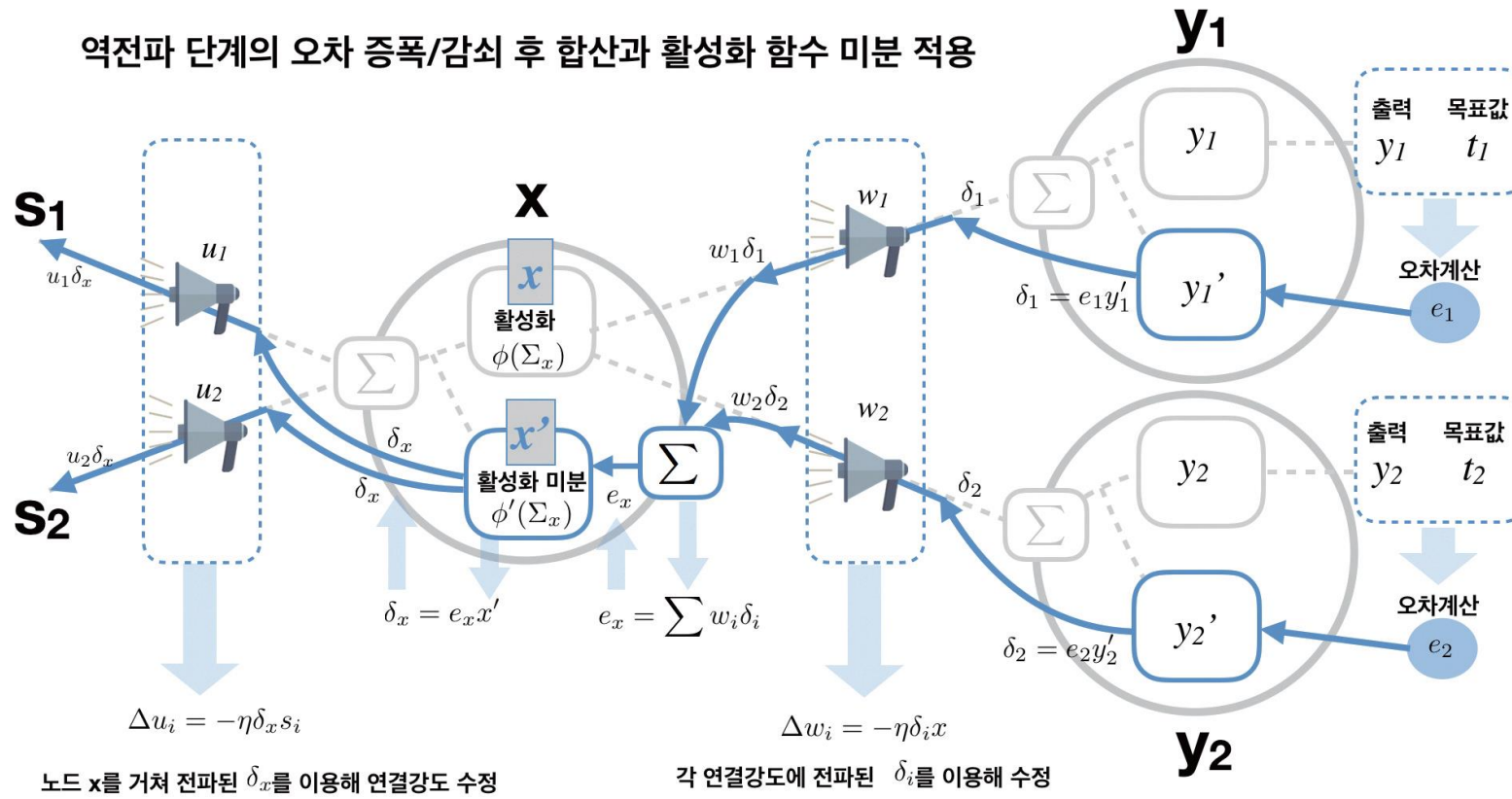
- 역전파 알고리즘을 이용한 연결강도 갱신 학습법을 이해하기 위해 아래와 같이 순전파 경로와 역전파 경로를 모두 표시한 연결망을 살펴보자



- 노드  $x$ 에는 두 개의 신호가 전달되며  $s_1$ 과  $s_2$ 를 연결강도  $u_1$ 과  $u_2$ 에 각각 곱한 뒤 모두 합한 것
  - 그림에서  $\Sigma_x$ 로 표시
  - 신경세포는 이렇게 모아진 신호가 임계치를 넘으면 출력을 하는 활성화 함수를 가짐
- 활성화 함수를  $\phi()$ 라 하면 노드  $x$ 는  $\phi(\Sigma_x)$ 를 계산하여 이 값을  $x$ 로 출력하는 것
- 주의해서 볼 값은  $x'$ 
  - 순전파 과정에서는 다음 노드로 전달되지는 않는 값
  - 활성화 함수의 미분  $x' = \phi'(\Sigma_x)$ 값으로 나중에 사용하기 위해 미리 계산해 두는 값
  - 이러한 일은 그 다음 단계인  $y_1$ 과  $y_2$  노드처럼 신경망 전체에서 동일하게 이루어짐

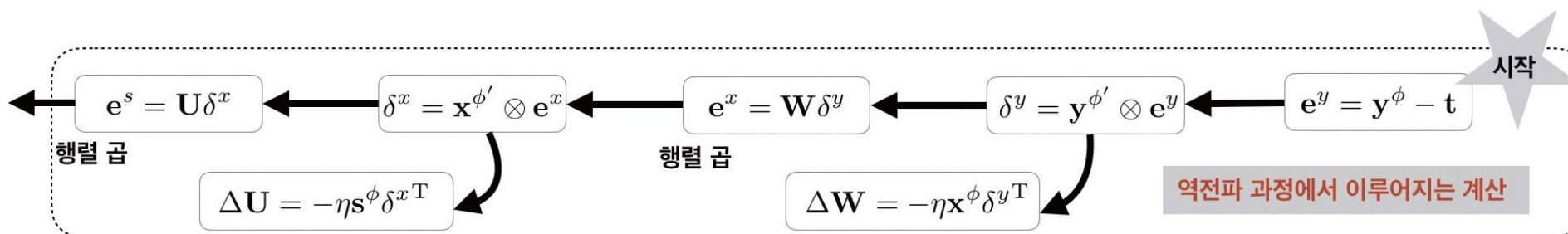


- 역전파 알고리즘은 순전파 과정에서 활성화 함수를 적용할 때마다 활성화 함수의 미분을 구해 두기만 한다면, 오차를 계산하여 순전파와 동일한 신호의 증폭과 합산으로 계산 다만 그 방향이 반대일 뿐



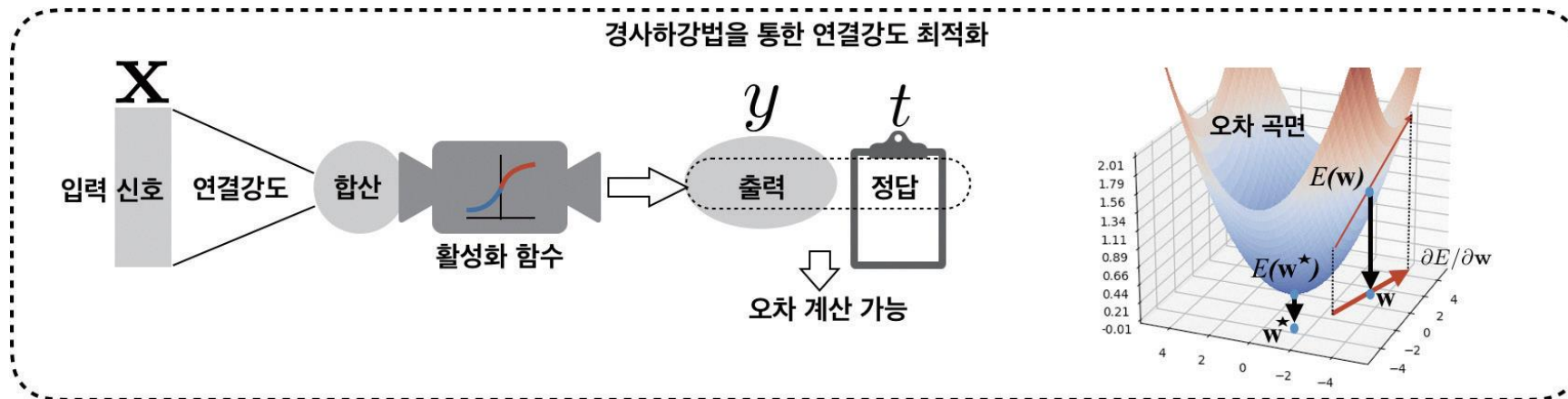
- 역전파 알고리즘을 적용하는 조건
  1. 오차를 계산하는 방법이 존재
  2. 활성화 함수는 미분이 가능한 함수
- 역전파 방법은 계속해서 아래 계층으로 내려갈 수 있음
  - 다층 퍼셉트론의 계층을 여러 층으로 쌓아도 모두 학습이 가능하다는 것을 의미
  - 인공 신경망 연구의 획기적인 발전을 이루는데 큰 기여

- 역전파 과정의 신호 전달을 행렬 연산으로 정리
  - 아래 그림의 오른쪽에서 왼쪽으로 진행하는 계산



## 7.10 역전파 알고리즘의 요약

- 다층 퍼셉트론의 학습을 가능하게 한 신경망 분야의 중요한 알고리즘
- 기본은 물론 경사 하강법이며 그림처럼 입력 계층  $x$  가 연결강도를 통해 출력 계층으로 연결되어 연결 강도에 의해 모아진 신호가 활성화 함수를 통과하여  $y$ 로 출력





- 연결강도 최적화는 연결강도 공간에서 오차의 기울기를 구해 그 반대 방향으로 파라미터를 움직여가는 것
- 기울기는 아래와 같이 오차 함수의 미분, 활성화 함수의 미분에 입력을 곱한 것으로 표현

연결강도 최적화 : 오차 곡면의 기울기 반대방향으로 연결강도를 옮겨 놓는 것

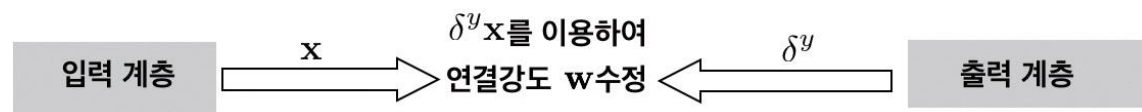
연결 강도를 수정하는 방향은 아래 기울기의 반대 방향

오차의 기울기 = 오차 미분 x 활성화 미분 x 입력

$\delta^y$

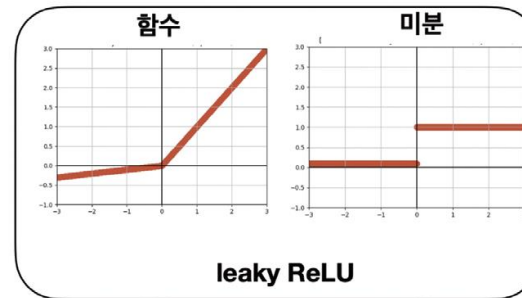
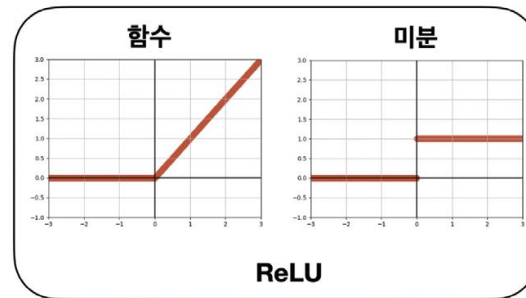
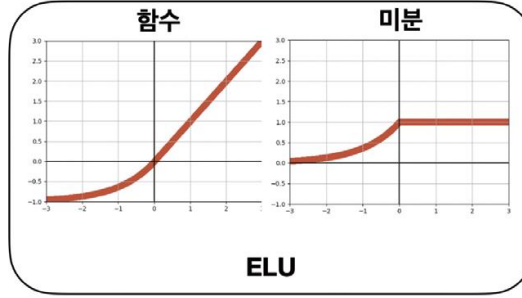
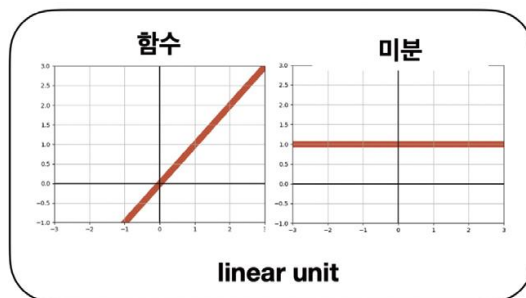
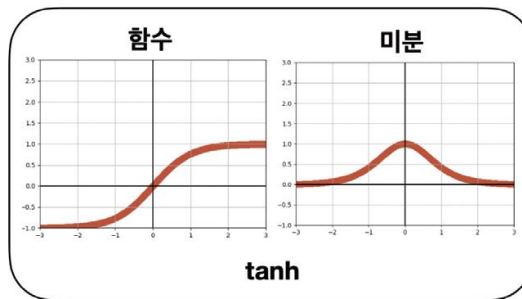
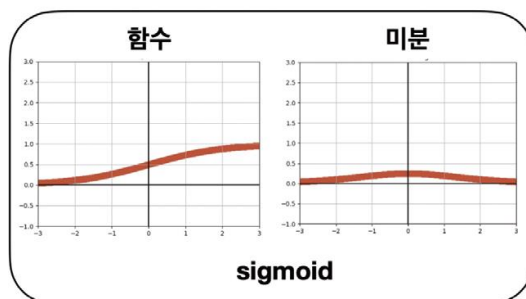
연결강도 수정을 위해 y 계층에서 내려오는 신호

- 오차 기울기를 결정하는 오차 함수의 미분과, 활성화 함수의 미분은 출력 계층에서 알 수 있는 값이고,  $x$  는 입력 계층의 값이므로 두 계층은 다음과 같은 정보를 이용해 협력하여 연결 강도를 수정



- 이러한 동작을 모든 계층에 일반화하면  $y$ 와  $x$  가 협력한 것과 똑같은 방식으로  $x$  계층과 그 아래에 있는 계층에도 협력하여 연결강도를 수정

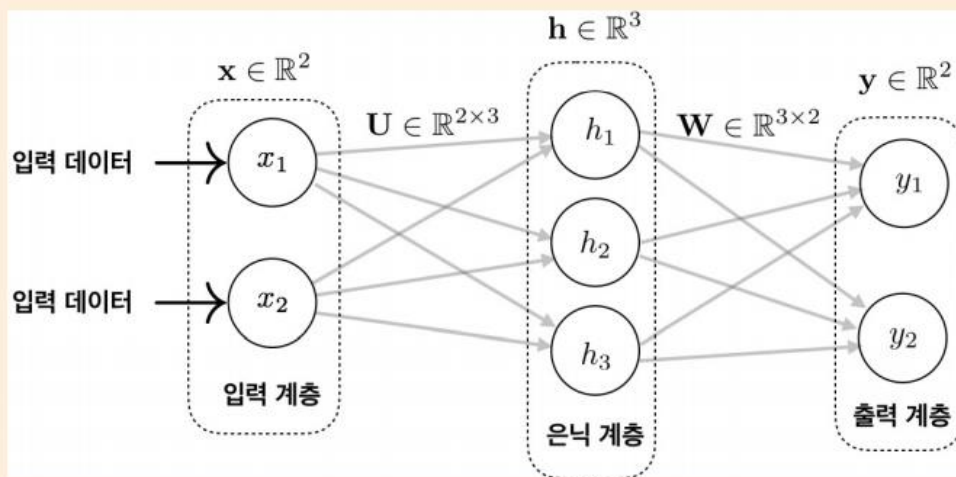
- 역전파 알고리즘은 오차 계산 함수와 활성화 함수만 결정되면 이들의 미분 함수들을 이용하여 계속해서 층별  $\delta$  벡터를 구하는 일
- 시그모이드 함수를 활성화 함수로 사용해 보았는데, 그 외에도 다양한 활성화 함수를 도입할 수 있음



# LAB<sup>7-5</sup> XOR 연산이 가능한 다층 퍼셉트론 만들기

## 실습 목표

역전파 모델로 아래와 같은 다층 퍼셉트론을 훈련시켜 XOR 연산을 수행하게 하라. 이 퍼셉트론의 출력은  $y_1$  이 켜지면 참,  $y_2$  가 켜지면 거짓을 나타낸다.



## 힌트

앞에서 살펴본 바와 같이 각 계층으로 유입되는 오차를 미분치에 곱해서 델타 벡터를 만들고, 이를 이용하여 연결강도를 수정한다. 그리고 이 델타 벡터는 가중치를 타고 내려가게 하면 된다.

교재의 해답 코드를 하나하나 단계별로 수행해  
봅니다

1. 넘파이와 맷플롯립을 임포트하여 수치 데이터를 다루고 가시화할 수 있게 한다.

```
import numpy as np
import matplotlib.pyplot as plt
```

2. 연결강도  $U$ 와  $W$ 를 난수를 발생시켜 준비한다. 실습 목표에 나타나 있는 퍼셉트론을 구현하기 위해서는 각각 (2,3)과 (3,2)의 크기를 갖는다. 학습률은 1.0으로 설정해 보겠다.

```
#  $x(2,1) \Rightarrow (U(2,3)) \Rightarrow h(3,1) \Rightarrow (W(3,2)) \Rightarrow y(2,1)$ 
```

```
U = np.random.rand(2,3) # 연결강도 U
W = np.random.rand(3,2) # 연결강도 W
```

3. 퍼셉트론이 사용할 활성화 함수로 시그모이드 함수를 구현하고, 이 함수의 미분함수도 구현한다.

```
def phi(v):
    return 1/(1+np.exp(-v))

def phi_derivative(v):
    phi_value = phi(v)
    return phi_value * ( 1 - phi_value )
```

#### 4. 각 층에서 가져야 할 값을 저장할 변수들 설정

```
## 입력 x
input = np.random.rand(2,1)

## 은닉층 h #####
h = np.zeros((3,1))
h_derivative = np.zeros((3,1))
#----- 오차 역전파 값
h_e, h_delta = np.zeros((3,1)), np.zeros((3,1))
#####

## 출력 y #####
y = np.zeros((2,1))
y_derivative = np.zeros((2,1))
#----- 오차 역전파 값
y_e, y_delta = np.zeros((3,1)), np.zeros((3,1))
#####
input, h, h_derivative, y, y_derivative
```

```
(array([[0.60789658],
        [0.03280674]]),
 array([[0.],
        [0.],
        [0.]]),
 array([[0.],
        [0.],
        [0.]]),
 array([[0.],
        [0.]]),
 array([[0.],
        [0.]])
```

## 5. 순전파 과정 구현

```
def forward_xh(x):  
    global input, h, h_derivative  
    input = x  
    Ux = np.dot(U.T, input)  
    h, h_derivative = phi(Ux), phi_derivative(Ux)
```

```
def forward_hy():  
    global y, y_derivative  
    Wh = np.dot(W.T, h)  
    y, y_derivative = phi(Wh), phi_derivative(Wh)
```

## 6. 학습률(하이퍼파라미터) 설정하고 역전파 과정 구현

```
## hyperparameter  
learning_rate = 1.0
```

```
def compute_error(target):  
    return y - target
```

```
def backward_yh(error):  
    global y_e, y_delta, W  
    y_e = error  
    y_delta = y_derivative * y_e  
    W = W - learning_rate * np.dot(h, y_delta.T)
```

```
def backward_hx():  
    global h_e, h_delta, U  
    h_e = np.dot(W, y_delta)  
    h_delta = h_derivative * h_e  
    U = U - learning_rate * np.dot(input, h_delta.T)
```



## 7. 학습은 역전파를 수행하는 것으로 완성

```
loss = []

def reset_weights():
    global U, W
    U = np.random.rand(2,3) # 연결강도 U
    W = np.random.rand(3,2) # 연결강도 W

def train_all():
    for i in range(1000):
        e_accum = 0
        true = np.array([[1], [0]])
        false = np.array([[0], [1]])

        e_accum += train(np.array([[0], [0]]), false)
        e_accum += train(np.array([[0], [1]]), true)
        e_accum += train(np.array([[1], [0]]), true)
        e_accum += train(np.array([[1], [1]]), false)

    loss.append(e_accum)
```

```
reset_weights()
train_all()
```

U, W

```
(array([[ -3.60256829,  6.75574647,  5.45845467],
        [ 6.38582624, -3.75044015,  5.23438359]]),
 array([[ -6.62707397,  6.64350975],
        [-6.61790001,  6.6341572 ],
        [ 9.31082621, -9.33316195]]))
```

## 8. 테스트 실시

```
def test(X):  
    y_hat = []  
    for x in X:  
        forward_xh(x)  
        forward_hy()  
        y_hat.append(y)  
    return y_hat
```

```
test(np.array([ [0],[0]], [0],[1]], [[1],[0]], [[1],[1]] ))
```

```
[array([0.1193482 ],  
       [0.88184756]),  
 array([0.92605773],  
       [0.07312978]),  
 array([0.92550069],  
       [0.07366119]),  
 array([0.03570503],  
       [0.96494666])]
```

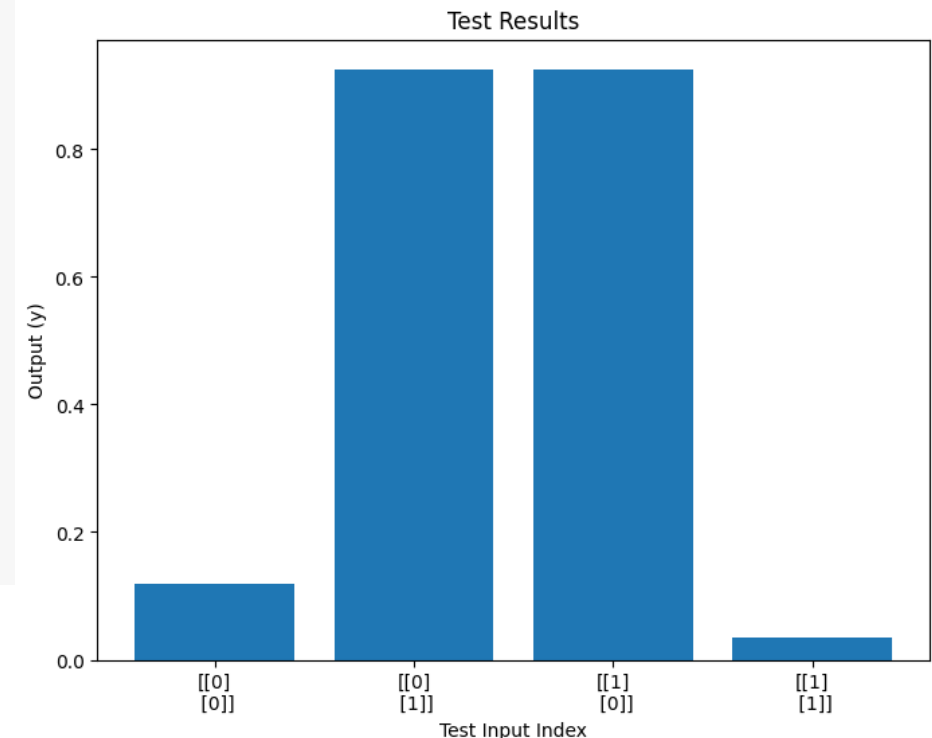
## 9. 테스트 결과 가시화

```
import numpy as np
import matplotlib.pyplot as plt

# Get test results
X_test = np.array([ [0],[0]], [0],[1]], [1],[0]], [1],[1]] )
y_hat = test(X_test)

y_values = np.array([y[0][0] for y in y_hat])

# Plotting the test result
plt.figure(figsize=(8, 6))
plt.bar(np.arange(len(y_values)), y_values)
plt.xlabel("Test Input Index")
plt.ylabel("Output (y)")
plt.title("Test Results")
plt.xticks(np.arange(len(X_test)), [str(x) for x in X_test])
plt.show()
```

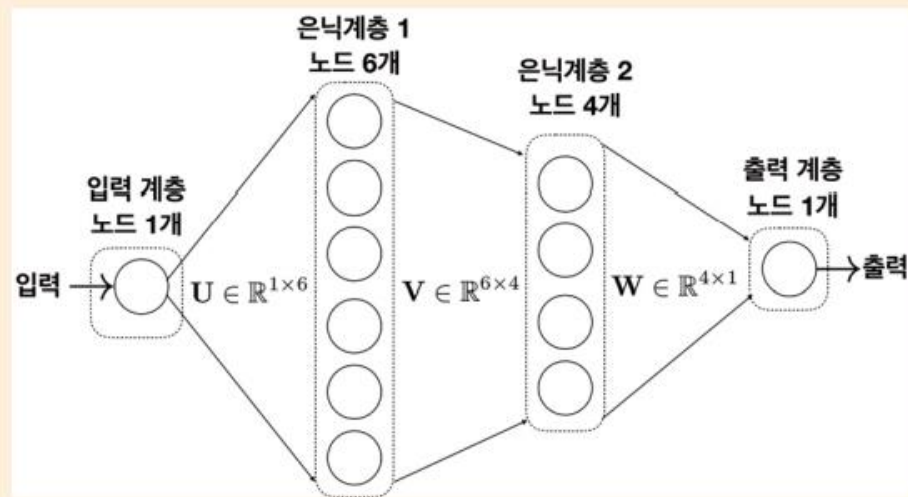


# LAB<sup>7-6</sup> 다층 퍼셉트론으로 비선형 회귀 구현하기

## 실습 목표

퍼셉트론의 층을 증가시켜 아래와 같은 모델로 비선형 회귀를 구현해 보자. 적용할 데이터는 LAB<sup>6-1</sup>에서 사용한 것과 같이 다음 URL에 있는 데이터이다.

<https://github.com/dknife/ML/raw/main/data/nonlinear.csv>



힌트

층을 늘이는 것은 순전파와 역전파 단계가 하나 늘어나는 것뿐이고, 계산은 동일한 방식으로 이루어진다.

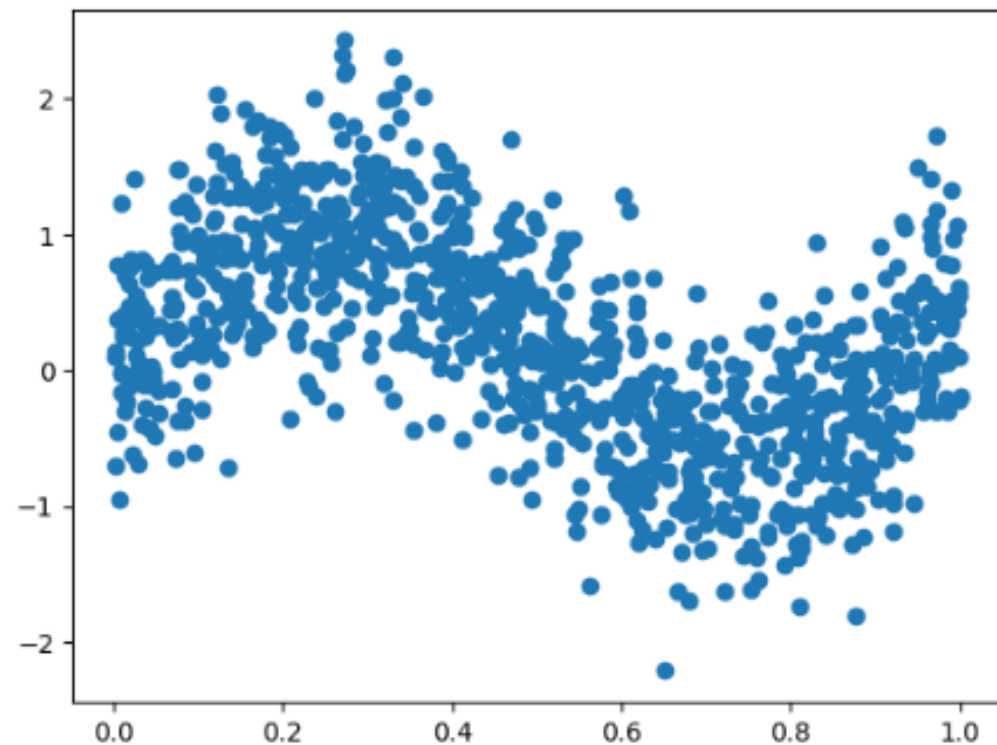
교재의 해답 코드를 하나하나 단계별로 수행해 봅니다

## 1. 데이터 확인

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

data_loc = 'https://github.com/dknife/ML/raw/main/data/'
df = pd.read_csv(data_loc + 'nonlinear.csv')
plt.scatter(df['x'], df['y'])
```

<matplotlib.collections.PathCollection at 0x7fe7ca6e1c50>



## 2. 각 층에 저장할 값 준비

```
nx, nh1, nh2, ny = 1, 6, 4, 1

## layer : input
input = np.zeros((nx, 1))

## layer : h1
h1, h1_deriv, h1_delta = np.zeros((nh1, 1)), np.zeros((nh1, 1)), np.zeros((nh1, 1))

## layer : h2
h2, h2_deriv, h2_delta = np.zeros((nh2, 1)), np.zeros((nh2, 1)), np.zeros((nh2, 1))

## layer : output y
y, y_deriv, y_delta = np.zeros((ny, 1)), np.zeros((ny, 1)), np.zeros((ny, 1))

U = np.random.randn(nx, nh1)
V = np.random.randn(nh1, nh2)
W = np.random.randn(nh2, ny)

learning_rate = 0.1

input, h1, h2, y, U, V, W
```

### 3. 활성화 함수와 순전파 구현

```
# 활성화 함수
def sigmoid(v) :
    return 1 / ( 1 + np.exp(-v))
```

```
def forward(x):
    global input, h1, h1_deriv, h2, h2_deriv, y, y_deriv

    input = x

    aggregate = U.T.dot(input)
    h1 = sigmoid(aggregate)
    h1_deriv = h1 * (1 - h1)

    aggregate = V.T.dot(h1)
    h2 = sigmoid(aggregate)
    h2_deriv = h2 * (1 - h2)

    aggregate = W.T.dot(h2)
    y = sigmoid(aggregate)
    y_deriv = y * (1 - y)
```

#### 4. 오차 계산과 역전파 정의

```
def compute_error(target):  
    return y - target
```

```
def backward(error):  
    global U, V, W, h1_delta, h2_delta, y_delta  
  
    y_delta = error * y_deriv  
    h2_delta = W.dot(y_delta) * h2_deriv  
    h1_delta = V.dot(h2_delta) * h1_deriv  
  
    W -= learning_rate * h2.dot(y_delta.T)  
    V -= learning_rate * h1.dot(h2_delta.T)  
    U -= learning_rate * input.dot(h1_delta.T)
```



## 5. 훈련 함수와 훈련에 사용할 데이터 준비

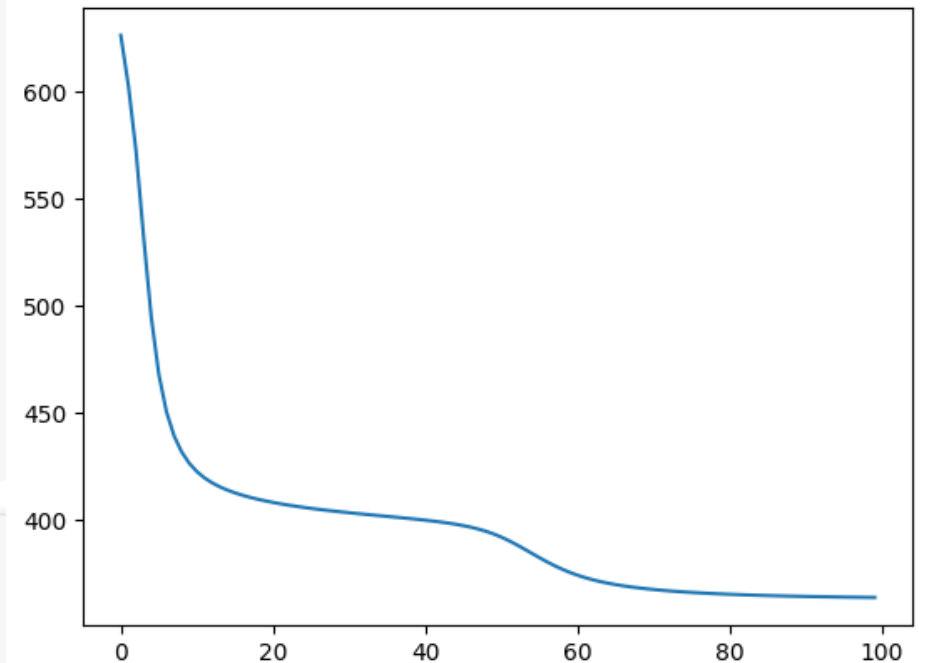
```
def train(x ,target) :  
    forward(x)  
    e = compute_error(target)  
    backward(e)  
    return e**2
```

```
X = df['x'].to_numpy()  
y_label = df['y'].to_numpy()  
  
X.shape, y_label.shape  
  
((1000,), (1000,))
```

## 6. 훈련실시 및 오차 확인

```
loss = []  
for i in range(100) :  
    e_accum = 0  
    for x, y in zip(X, y_label):  
        x_in = np.array([x]).reshape(nx,1)  
        y_target = np.array([y]).reshape(ny,1)  
        e_accum += train(x_in, y_target)  
    loss.append(e_accum)
```

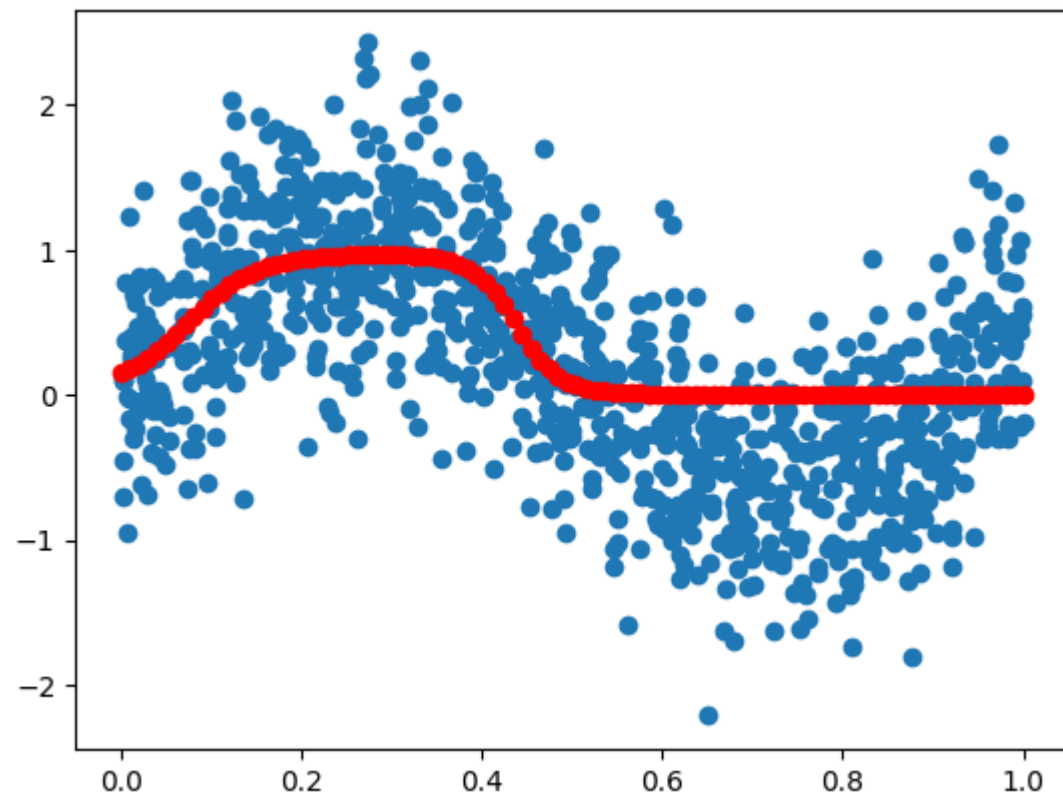
```
err_log = np.array(loss).flatten()  
plt.plot(err_log)
```



## 7. 예측 실시

```
def predict(X):  
    y_hat = []  
    for x in X:  
        forward(np.array([x]).reshape(nx,1))  
        y_hat.append(y)  
    return y_hat
```

```
domain = np.linspace(0, 1, 100).reshape(-1, 1)  
y_hat = predict(domain)  
plt.scatter(df['x'], df['y'])  
plt.scatter(domain, y_hat, color='r')
```



# 민스키의 재등판



## 잠깐 - 역전파 알고리즘은 신경망의 부활을 가져올까?

1980년대에 신경망 분야는 역전파 알고리즘으로 활기를 되찾게 된다. 그런데 민스키는 역전파 알고리즘에 대해서도 퍼셉트론의 2판을 써서 다음과 같은 비판적인 의견을 내었다.

"많은 연결주의자들은 역전파가 단순히 기울기를 계산하는 특별한 방법에 불과하다는 것을 이해하지 못하고, 이것이 **언덕 오르기(hill-climbing)** 알고리즘의 기본적 한계를 극복하는 새로운 학습 기법이라 여기는 것 같다." <sup>10)</sup>

민스키의 지적대로 역전파가 마법은 아니지만, 신경망으로 해결할 수 있는 문제는 크게 확장되었다. 그런데, 신경망 분야는 금세 다시 침체에 빠지게 된다. 역전파가 무력해지는 경우가 많기 때문이다. 이 문제의 이유와 극복 과정이 앞으로 살펴볼 내용이다.

---

10) Minsky, M. and Papert, S. (1988) Perceptrons: An Introduction to Computational Geometry 2nd Ed., MIT Press, 260-61.

## 신경망을 설계하고 훈련을 실시할 수 있는 도구 : 텐서플로우

- 오차 역전파를 통한 신경망 학습의 방법을 살펴보고 직접 구현하면서 여러 층으로 이루어진 신경세포 층들 사이의 신호 전달이 동일한 방식으로 차원만 바꾸어 진행되는 행렬-벡터 연산임을 확인
- 신경망 구현은 **텐서플로우**TensorFlow라는 도구를 사용할 것이며 텐서플로우는 구글에서 만든 것으로 지금까지 실습을 진행하기 위해 사용한 구글 코래버러토리에 설치가 되어있어 준비없이 바로 사용할 수 있음
- 텐서플로우는 **케라스**keras라는 심층신경망 구현을 위한 API가 포함되어 있는데, 이를 이용하면 간단히 신경망을 만들고 학습 가능



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# 텐서플로우와 케라스를 사용할 수 있도록 준비
import tensorflow as tf      # 텐서플로우는 주로 tf라는 별명을 사용한다
from tensorflow import keras
```

- 모델은 신경 계층이 차례로 이어져 순차적으로 신호를 전달하는 모델
- 이런 모델은 케라스에서 제공하는 시퀀셜`Sequential` 클래스로 구현
- $LAB^{7-6}$ 의 회귀 모델은 입력 계층을 제외하면 6개, 4개, 1개의 노드를 가진 층`layer`들이 쌓여 있으며 이 모든 층들은 이전 단계의 층이 가지는 신경세포 노드와 빠짐없이 다 연결되어 있음
  - 이러한 연결을 밀집`dense` 계층
  - 각 층이 가진 활성화 함수는 시그모이드



```
model = keras.models.Sequential( [  
    keras.layers.Dense(6, activation= 'sigmoid'),  
    keras.layers.Dense(4, activation= 'sigmoid'),  
    keras.layers.Dense(1, activation= 'sigmoid'),  
])
```

- 전체 데이터를 다 사용하지 않고 임의로 선택된 데이터 인스턴스만을 가지고 경사 하강법을 적용하는 **확률적 경사 하강법** stochastic gradient descent:SGD을 사용
- 확률적 경사 하강법은 전체 데이터를 이용하여 학습하는 경사 하강법과는 다르게 임의적으로 추출한 일부 데이터를 사용해서 가중치를 조절 하기에 가중치 조절의 속도가 개선되는 효과
- 모델을 사용하기 위해서는 **컴파일** compile 과정
  - 컴파일 과정에서는 최적화 방법을 지정하고, 오차 측정을 어떤 기준으로 할 것인지 지정
  - 평균 제곱 오차를 손실함수로 사용한다면 'mse'를 손실 매개변수 loss의 인자로 지정



```
optimizer = keras.optimizers.SGD(learning_rate=5.0)
model.compile(optimizer=optimizer, loss='mse')
```

- 모델을 훈련시킬 데이터는  $LAB^{7-6}$ 에서 사용한 것과 동일



```
data_loc = 'https://github.com/dknife/ML/raw/main/data/'  
df = pd.read_csv(data_loc+'nonlinear.csv')  
X = df['x'].to_numpy()  
y_label = df['y'].to_numpy()
```

- 모델의 훈련은 입력과 레이블을 fit() 메서드
- **에폭**epoch을 지정하는데, 에폭은 하나의 데이터셋에 대해 몇 번 훈련을 반복할 것인지를 의미



```
model.fit(X, y_label, epochs=100)
```

```
Epoch 1/100
```

```
32/32 [=====] - 1s 1ms/step - loss: 0.6938
```

```
...
```

```
Epoch 100/100
```

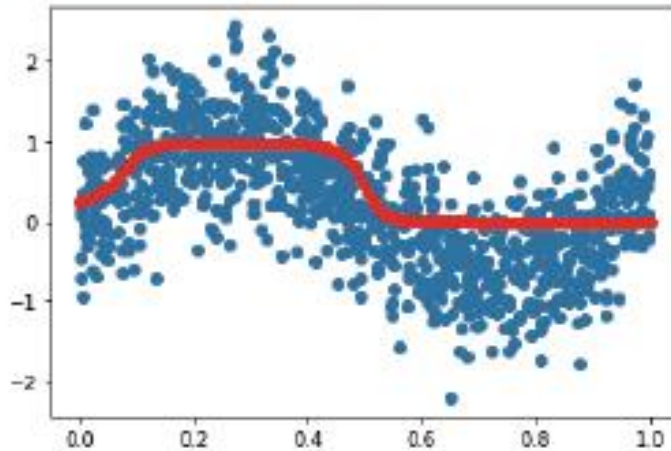
```
32/32 [=====] - 0s 1ms/step - loss: 0.3652
```



- $LAB^{7-6}$  에서 사용한 테스트용 입력을 모델에 넣어서 예측
- 케라스 모델에서 예측을 수행하는 방법은 predict() 메서드에 입력 데이터를 넘김



```
domain = np.linspace(0, 1, 100).reshape(-1,1) # 입력은 2차원 벡터로 변형  
y_hat = model.predict(domain)  
plt.scatter(df['x'], df['y'])  
plt.scatter(domain, y_hat, color='r')
```



# 핵심 정리

- 맥컬록과 피츠는 1940년대 초 선구적인 연구를 통해 신경세포의 신호전달 모델을 이용하여 계산을 할 수 있음을 보였다.
- 맥컬록과 피츠의 신경망 모델은 학습이 불가능하다는 단점이 존재한다.
- 로젠블랫은 퍼셉트론 모델을 통해 신경세포 모델을 이용하여 스스로 학습하여 문제를 해결할 수 있는 모델을 1950년대에 제안하였다. 이 때의 학습 방법은 헵의 학습 규칙에 근거한 것이었다.
- 헵의 학습 규칙은 "같이 활성화 되는 신경세포는 함께 연결된다"로 요약할 수 있다.
- 로젠블랫의 퍼셉트론 모델은 당시 큰 관심을 모았고 많은 연구자들이 퍼셉트론과 같은 방식의 인공 신경 망 연구에 참여하였다.
- 기호주의 인공지능 연구자들은 퍼셉트론에 대한 과열된 관심이 언론을 편파적으로 활용한 비과학적인 주장 때문이라고 보았다.
- 기호주의 인공지능 분야의 대가 민스키는 패퍼트와 함께 퍼셉트론이라는 제목의 책을 저술하여 퍼셉트론이 가진 한계를 보였다.
- 민스키의 비판 가운데 가장 잘 알려진 것은 퍼셉트론이 XOR와 같은 단순한 문제도 풀 수 없는 한계를 가진다는 것이었다.

# 핵심 정리

- XOR 문제는 당시 퍼셉트론이 가진 **구조적 단순성**으로는 극복할 수 없었다.
- 복잡한 구조의 퍼셉트론으로 민스키가 제기한 문제를 해결할 수 있지만, 당시에는 **복잡한 구조의 신경망을 학습시킬 방법**이 없었다.
- 1980년대에 **럼멜하트, 힌튼, 윌리엄스**가 **오차 역전파 알고리즘**을 개발함으로써 여러 층으로 이루어진 퍼셉트론의 학습이 가능하게 되었다.
- 역전파 알고리즘은 1974년 **워보스**의 박사학위 논문에서 같은 개념이 제안되었다.
- 역전파 알고리즘은 **인공 신경망 분야의 새로운 활기**를 가져왔다.
- 역전파 알고리즘은 순전파와 마찬가지로 계층들 사이의 신호 전달을 하는 것이며, 이 과정은 **행렬과 벡터의 연산**으로 표현할 수 있다.
- **텐서플로우**와 같은 도구를 이용하면 다층 구조의 신경망을 쉽게 정의하고 사용할 수 있다.
- 경사 하강법의 속도를 개선하는 방법으로 학습시 전체 데이터를 사용하지 않고 랜덤하게 추출한 데이터를 사용하는 **확률적 경사 하강법**이 있다.