

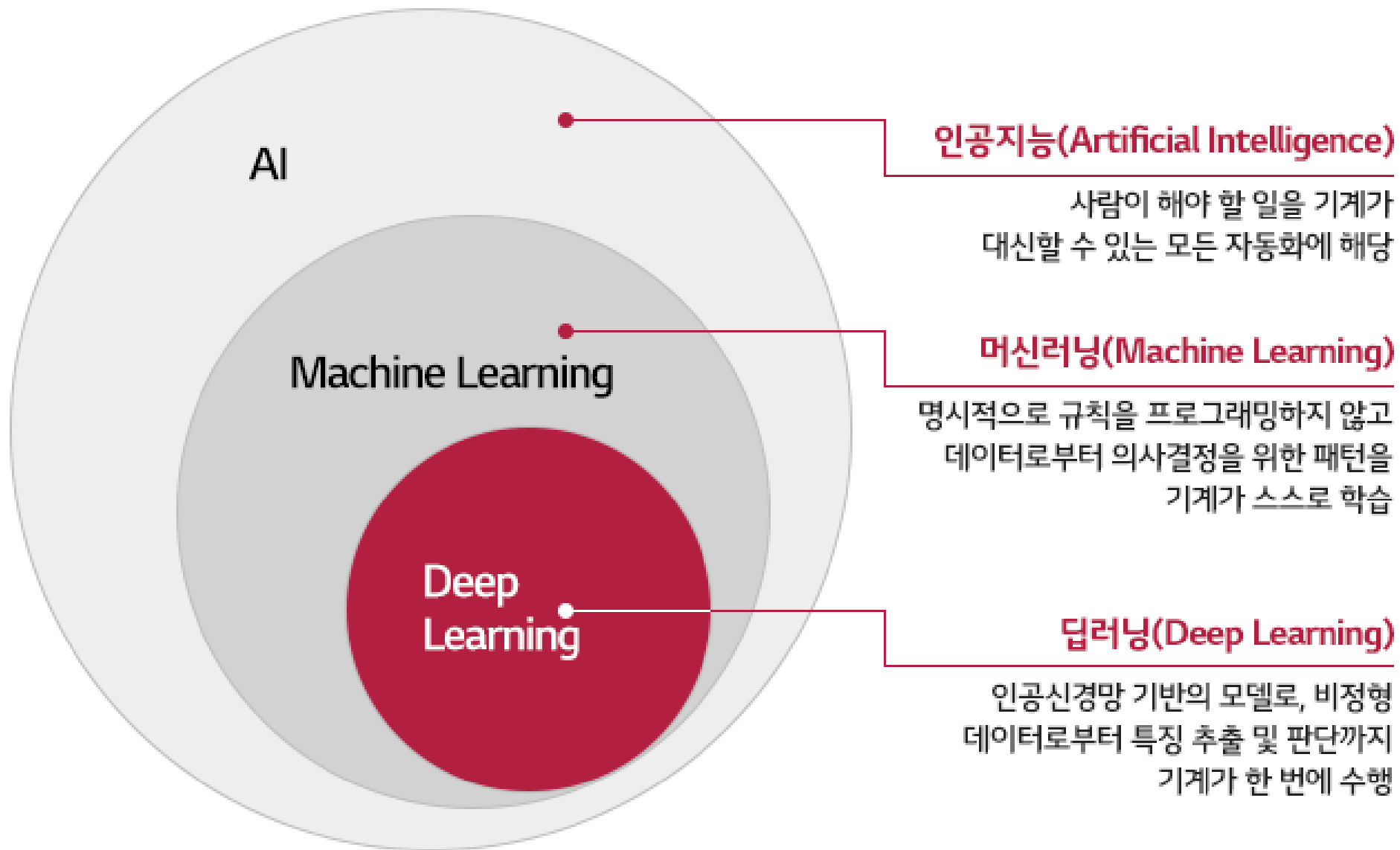
LG계약학과 제어시스템전공 2025년 하계 집중교육 특강

인공지능SW 설계 및 연구동향

안상태 교수

2025년 8월 18일 (월)

3. 인공신경망



3.1 뇌의 동작을 흉내 내자 - 연결주의자의 목표

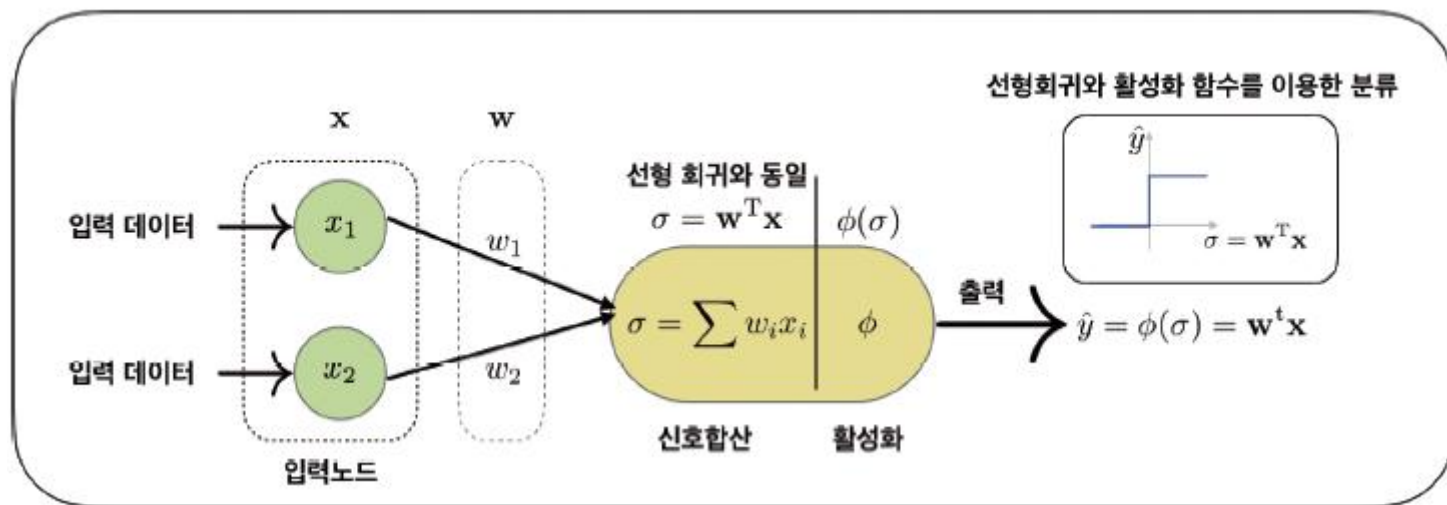
- 뇌는 뉴런neuron이라는 수없이 많은 신경세포들의 연결을 갖고 있음
- 인공 신경망은 이러한 신경세포의 동작을 흉내 내는 장치나 소프트웨어를 만들어 뇌가 수행하는 인지나 사고 능력을 갖춘 기계를 만들려는 노력
 - 이러한 방식의 연구를 통해 인공지능을 구현하려는 방식을
연결주의connectionism

3.2 학습할 수 있는 신경 모델 - 퍼셉트론

- 학습이 가능하게 만든 것이 퍼셉트론perceptron
- 1958년에 프랭크 로젠블랫Frank Rosenblatt이 만든 퍼셉트론은 미국 해군에 의해 최초로 공개되어, 당시 인공지능의 새로운 장을 연 기술로 각광

3.2 학습할 수 있는 신경 모델 - 퍼셉트론

- 가장 중요한 변화는 그림 왼쪽의 입력을 받아들이는 신경세포 x_i 들이 출력을 수행하는 노드로 연결될 때 **연결강도** w_i 에 의해 조정되어 전달된다는 것
- 출력 노드는 두 가지 일을 하는데, 우선 전달되어 오는 신호를 모두 합한다.
- 합산된 신호에 따라 출력을 결정하는 **활성화** ϕ 함수가 최종 출력을 발생

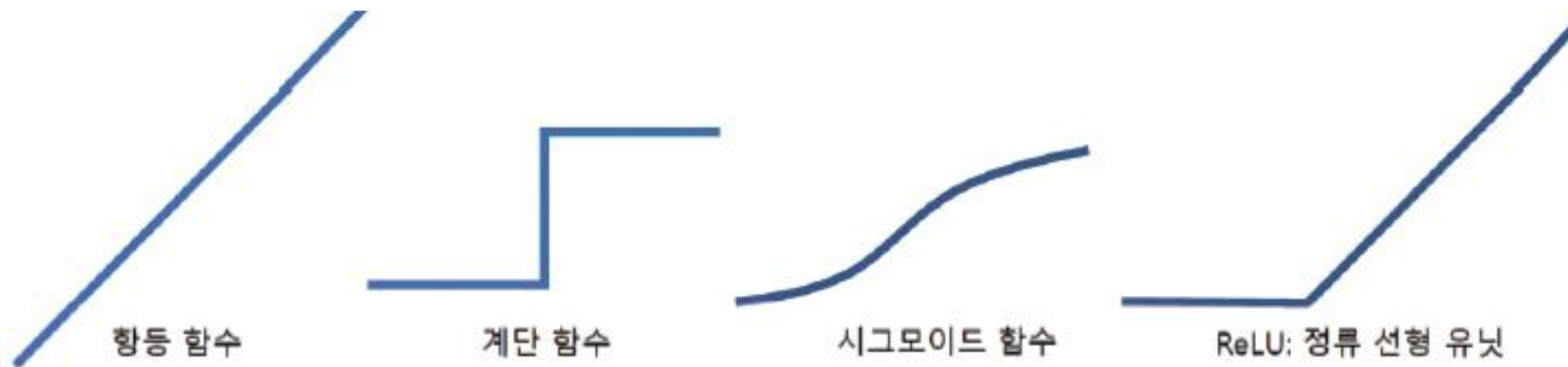


3.2 학습할 수 있는 신경 모델 - 퍼셉트론

- 학습은 경험을 통해 기능이나, 성능이 바뀌는 것
 - 이를 위해서는 학습 과정을 통해 바뀌는 부분이 모델 내에 존재해야 함
 - 퍼셉트론은 학습을 통해 연결강도를 바꾸어 나감
 - 출력의 결과와 정답을 비교하면 오차를 알 수 있으며 이를 줄이도록 **연결강도**를 바꾸는 과정이 바로 학습

3.2 학습할 수 있는 신경 모델 - 퍼셉트론

- 활성화 함수는 출력 노드에 모아진 신호를 다음으로 내어 보낼때 얼마나 강하게 보낼지를 결정하는 함수
 - 계단 함수는 최초의 퍼셉트론이 채택한 활성화 함수 미분이 되는 구간에서 미분치가 언제나 0이기 때문에 경사 하강법을 통한 최적화가 불가능
 - 이러한 이유로 신경망 분야에서 새롭게 도입되어 오랫동안 사용되던 활성화 함수가 **시그모이드** sigmoid 함수

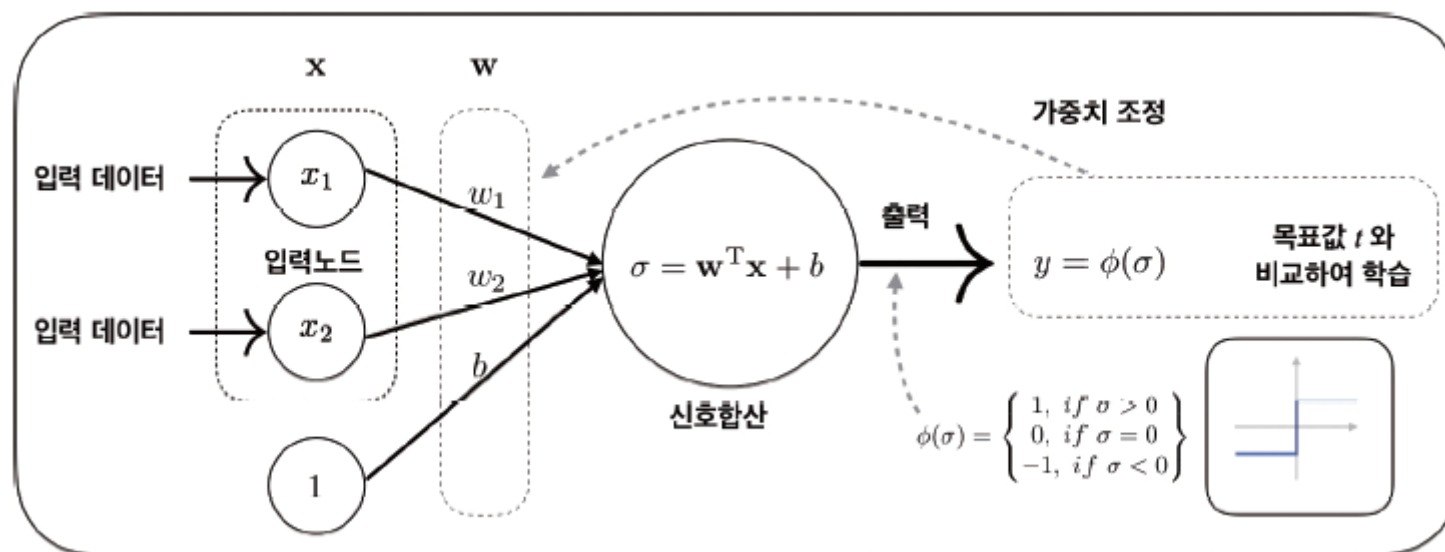


3.2 학습할 수 있는 신경 모델 - 퍼셉트론

- 신경망이 동작을 변경하는 방법은 연결강도를 조정하는 것
- 인공 신경망의 학습이라는 것은 출력을 목표치와 비교하여 오차를 계산하고 이 오차를 줄이는 방향으로 연결강도를 변경하는 일을 의미
 - 신경망 모델에서는 이 연결강도가 바로 모델의 동작을 결정하는 **파라미터**parameter
 - 오차를 계산하는 방법은 다양하며, 이 오차를 줄이는 방향으로 연결강도를 조정하는 일이 신경망의 **최적화**optimization, 즉 학습
 - 학습과정을 조절하는 **하이퍼파라미터**hyperparameter로는 오차에 따른 연결강도 조정 정도, 최적화 방법, 학습 반복 횟수 등이 있음

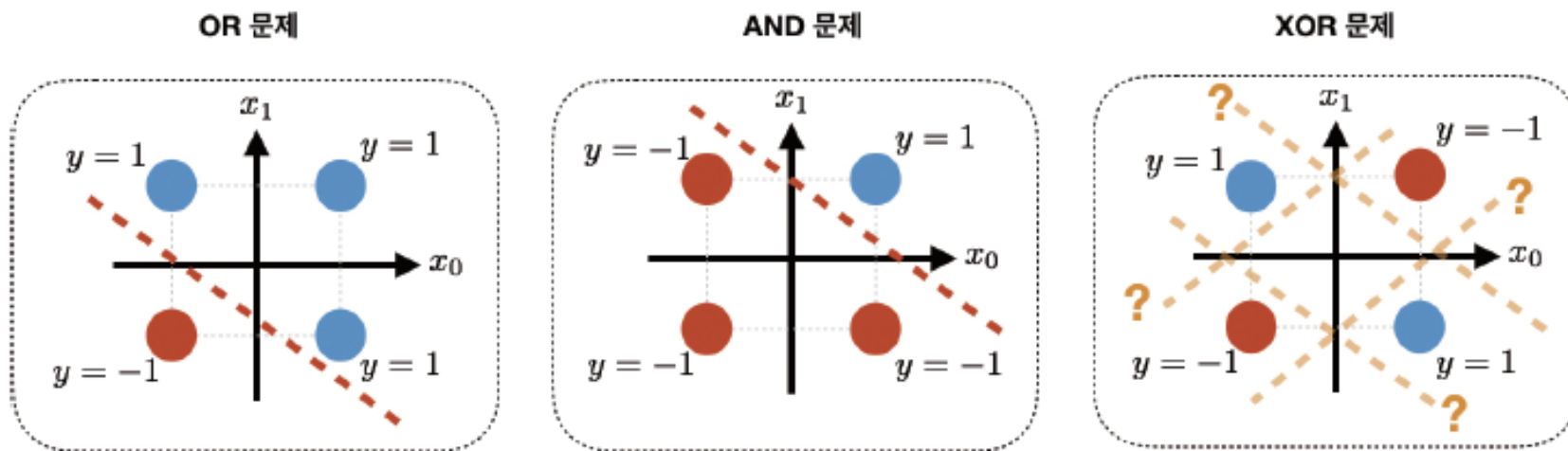
3.3 학습의 원리 - 연결강도의 변경

- 이항 불리언 연산 **binary boolean operation**이 가능하도록 하는 퍼셉트론은 아래 그림처럼 만들 수 있음
- 참 또는 거짓을 갖는 입력은 x_1 과 x_2 에 제공
- 이 신호는 가중치가 곱해져 출력 노드로 전달
- 이때 신호의 **편향** **bias**이 이루어지도록 하는 가중치 b 가 추가
- 편향은 신호값을 양의 방향이나 음의 방향으로 이동시키는 역할



3.4 퍼셉트론, 연결주의가 누린 첫 영예와 긴 좌절

- 기호주의자들은 로젠블랫이 과학적 기준을 따르지 않고, 언론을 편파적으로 활용해 "**과장된 주장**^{overclaim}"을 한다고 봄
- 마빈 민스키와 시모어 패퍼트가 쓴 "퍼셉트론"이라는 저서는 당시 엄청난 기대를 받던 신경망의 한계를 밝혀내기 위해 온 힘을 기울여 쓴 책
 - 가장 잘 알려진문제는 퍼셉트론이 XOR 연산을 구현하지 못 한다는 것

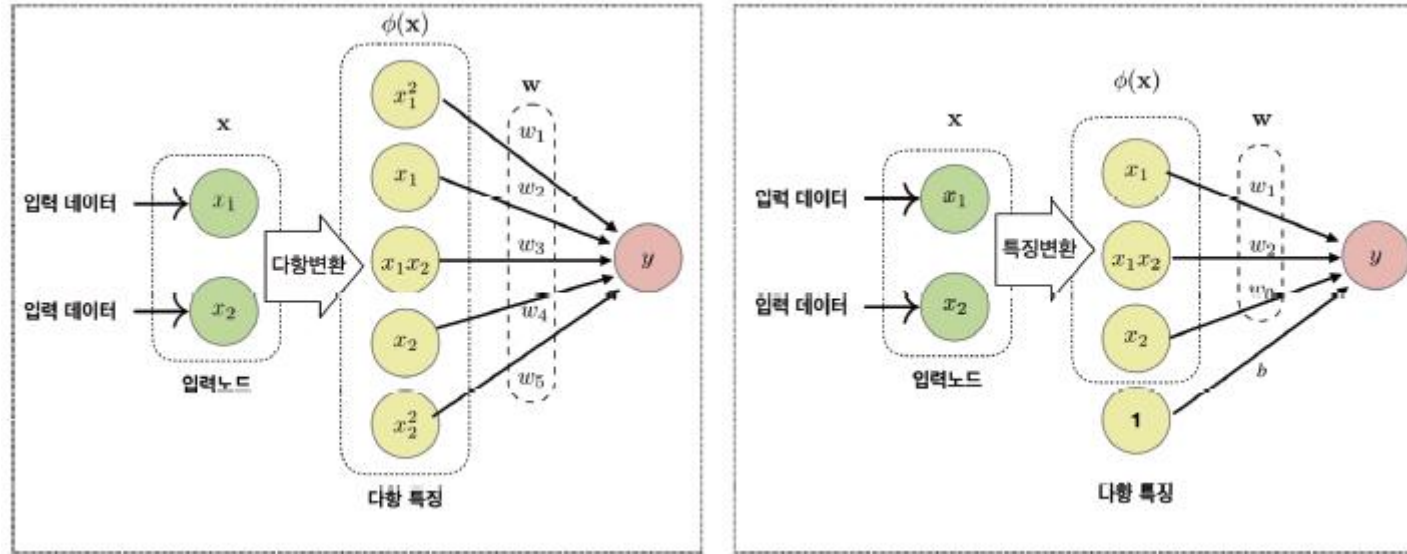


3.5 퍼셉트론이 XOR 문제를 풀 수 있게 만드는 방법

- 입력 노드가 출력으로 바로 연결되어 있는 단순한 퍼셉트론은 선형 분리만 가능하지만, 이미 맥컬록과 피츠의 신경 모델에서도 신호는 여러 층을 거쳐 전달되는 모델이 사용됨
- 퍼셉트론의 구조를 복잡하게 만들어 다양한 일을 하게 할 수 있지 않을까? 선형 함수로서 동작하는 모델을 비선형 함수로 바꾸는 방법 중에 우리가 이미 살펴본 것은 특징 벡터를 다항화하는 방법
- 단순한 퍼셉트론 모델에서는 x_1 와 x_2 의 입력을 가지므로, 이를 2차 다항화하면 다음과 같은 특징 벡터를 얻을 수 있을 것

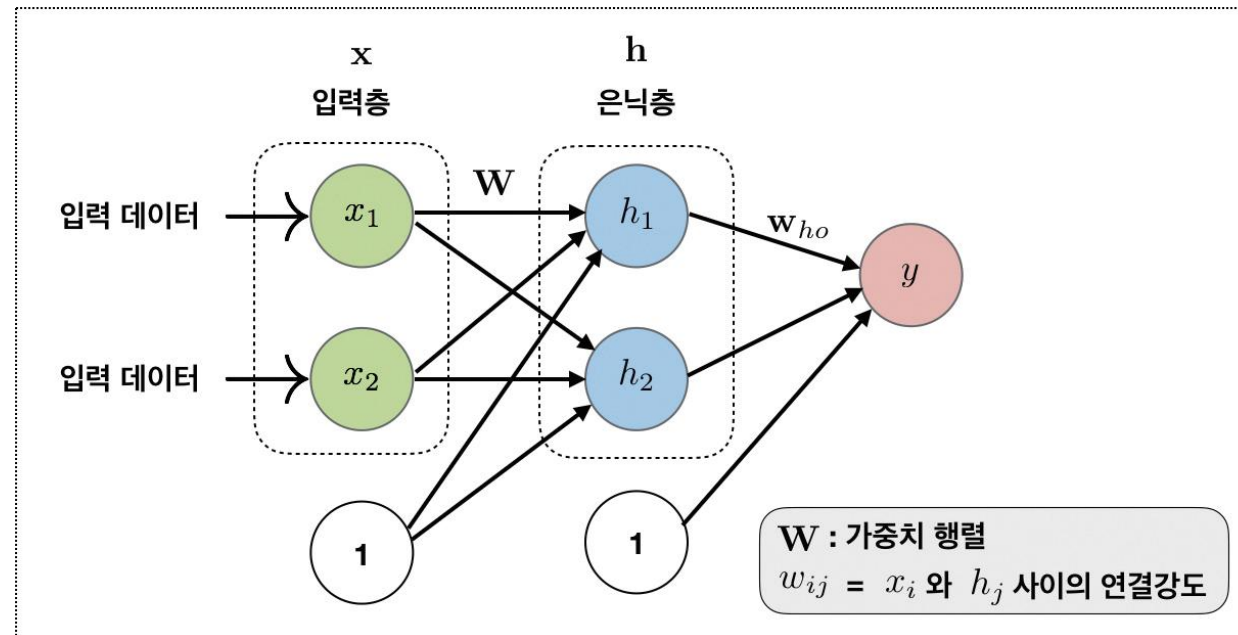
$$(x_1^2, x_1, x_1 x_2, x_2^2)$$

- 두 개의 입력을 이렇게 다항화하고 이를 퍼셉트론으로 인지하는 모델은 아래의 왼쪽과 같은 퍼셉트론으로 설명

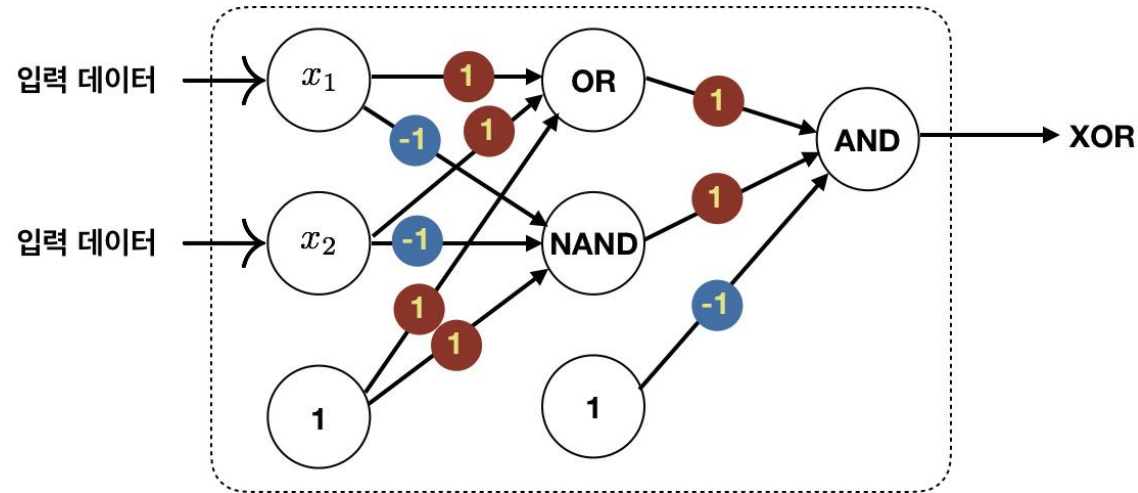


- 원래의 입력 x_1, x_2 를 다항 변환하여 새로운 입력 $\phi(x)$ 로 만드는 과정은 신호를 전달하는 것이 아니라 직접적인 계산이 이루어짐
- 입력과 출력을 바로 연결하지 않고 중간에 신호를 중계하는 신경세포들을 두는 것
 - 입력과 출력 사이에 존재하는 이런 중간 노드들의 계층을
은닉층 hidden layer

- 중간에 층을 두고 연결하면 다음과 같은 구조의 다층 퍼셉트론 multi-layer perceptron을 만들 수 있을 것
- 가중치는 입력층에 있는 n 개의 노드가 발생시키는 신호를 은닉층에 있는 m 개의 노드로 연결하는 가중치 행렬 W 에 의해 전달
- 은닉층의 노드들이 발생시키는 신호는 또 다른 연결강도 벡터 w_{ho} 에 의해 출력으로 연결



- 이런 구조의 퍼셉트론은 **더 많은 수의 연결이 더 많은 단계로 연결되어 복**잡한 동작을 할 수 있음
- 적절한 연결강도를 부여하면 단순한 모델이 할 수 없었던 XOR 문제 풀이도 가능

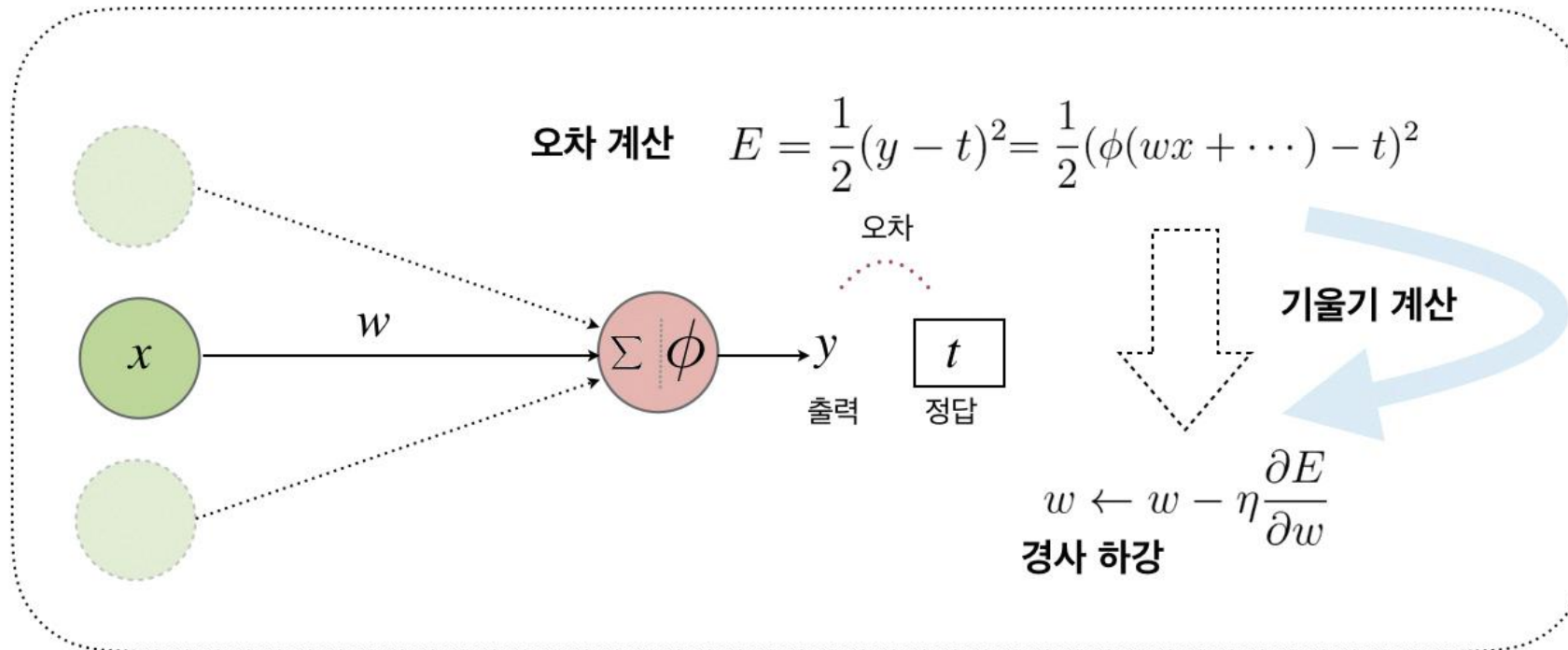


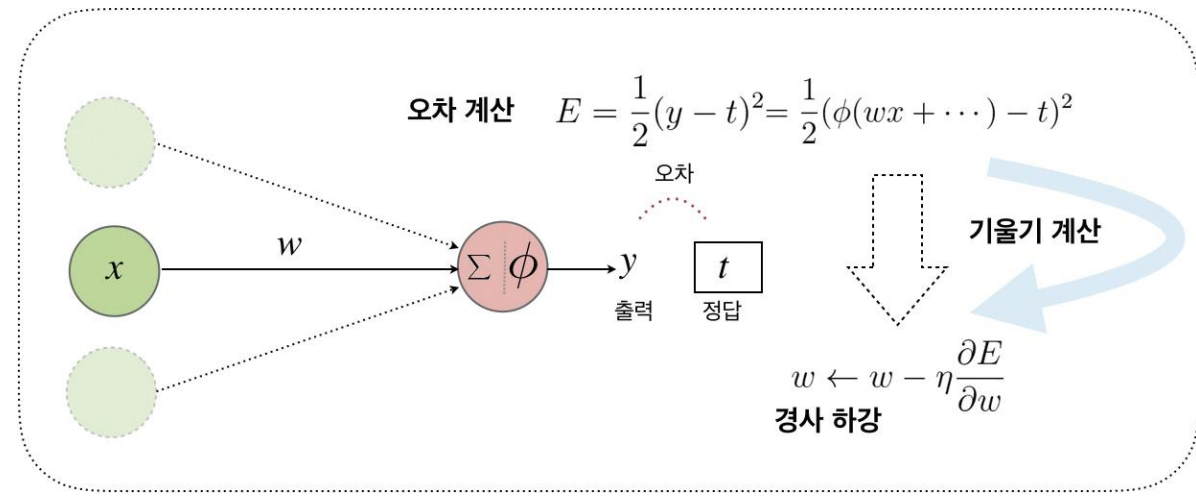
- 헵의 학습 규칙은 최종 은닉층과 출력의 연결에만 적용
- 층을 쌓아 만든 퍼셉트론이 분명 다양한 기능을 수행할 수 있지만, 오류를 순층이 고칠 수 없음

3.6 다층 퍼셉트론의 학습

- 인공 신경망에서 연결강도를 조정해 오차를 줄이는 방법
- 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 $1/2 (\phi(wx + \dots) - t)^2$



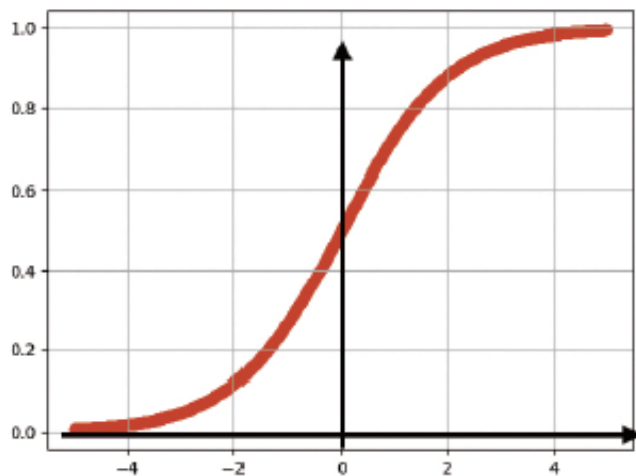


- 오차를 연결강도 w 에 대해 편미분

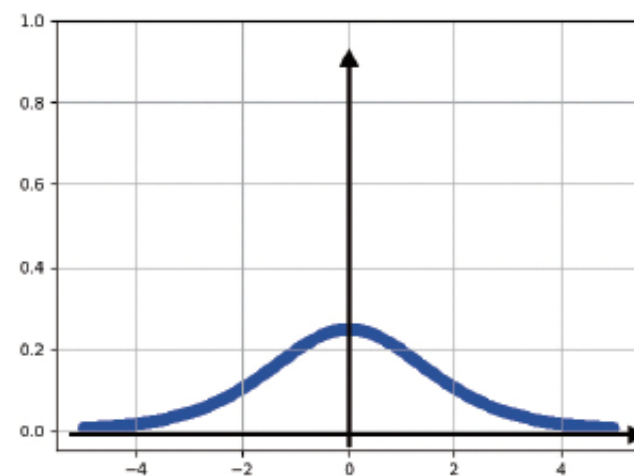
$$\begin{aligned}
 \frac{\partial E}{\partial w} &= \frac{1}{2} \frac{\partial}{\partial w} (\phi(wx + \dots) - t)^2 \\
 &= (\phi(wx + \dots) - t) \cdot \frac{\partial}{\partial w} \phi(wx + \dots) \\
 &= (\phi(wx + \dots) - t) \cdot \phi'(wx + \dots) \cdot \frac{\partial}{\partial w} (wx + \dots) \\
 &= (y - t) \cdot \phi'(wx + \dots) \cdot x
 \end{aligned}$$

- 기울기를 이용하여 연결강도를 수정
- 활성화 함수 $\phi()$ 의 미분 $\phi'()$ 만 안다면 입력 x , 출력 y , 목표값 t 를 이용하여 간단히 계산할 수 있음
- 계단함수는 미분이 불가능한 지점이 있고, 미분이 되는 곳에서도 미분치가 언제나 0
 - 이 방법을 위해서는 미분이 가능한 새로운 활성화 함수가 필요
 - 신경망에서는 다음과 같은 **시그모이드**sigmoid 혹은 **로지스틱**logistic 함수를 활성화 함수로 도입

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



$$\phi'(x) = \frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}}\right)$$



- 이 함수는 미분이 가능하다는 장점과 함께, 이 함수의 미분은 다음과 같이 원래의 함수를 이용하여 표현할 수 있는 장점이 있음

$$\phi'(x) = \phi(x)(1 - \phi(x))$$

- 조금 전 구했던 오차의 기울기에서 활성화 함수의 미분으로 표시된 부분을 출력값 y 로 표현할 수 있음

$$\begin{aligned} \frac{\partial E}{\partial w} &= (y - t) \cdot \phi'(wx + \dots) \cdot x \\ &= (y - t) \cdot \phi(wx + \dots) \cdot (1 - \phi(wx + \dots)) \cdot x \\ &= (y - t) \cdot y \cdot (1 - y) \cdot x \end{aligned}$$

- 출력값과 목표값만 알면 출력의 오차를 줄이는 연결강도 변경을 아래의 수식을 사용하여 수행할 수 있음

$$w \leftarrow w - \eta \cdot \partial E / \partial w$$

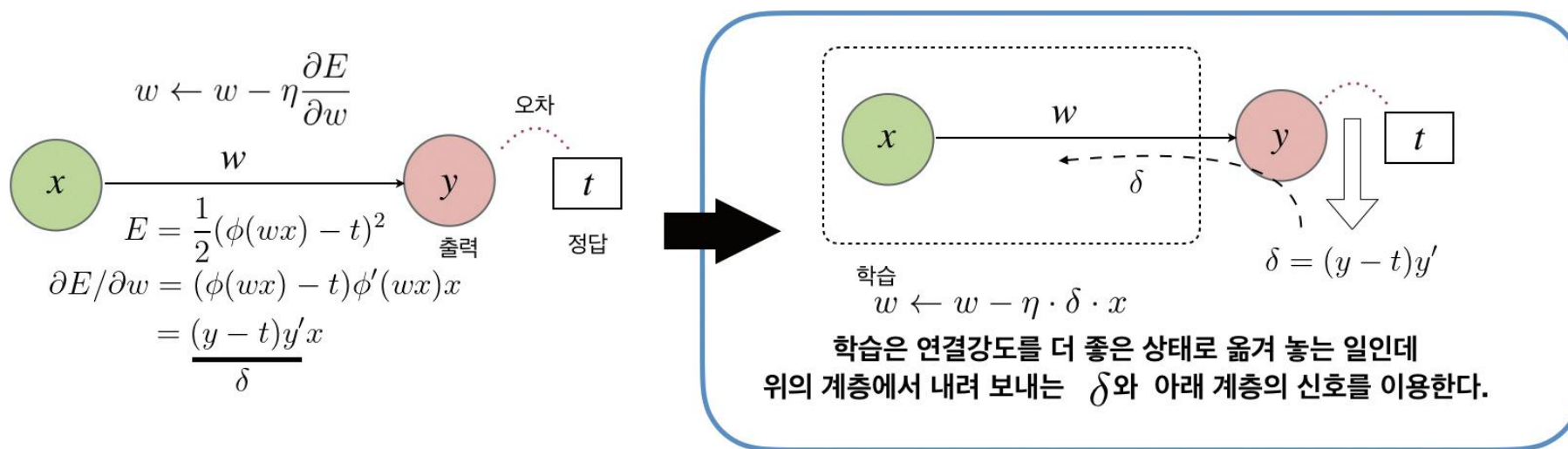
- 여기서 출력값 y 와 목표값 t 만 알면 연결강도 수정에 필요한 $\delta = (y - t)\phi'(wx + \dots)$ 가 $\delta = (y - t) \cdot y \cdot (1 - y)$ 를 통해 쉽게 계산되며 δ 를 이용하여 연결강도를 더 좋은 상태로 바꿀 수 있음

$$w \leftarrow w - \eta \cdot \delta \cdot x$$

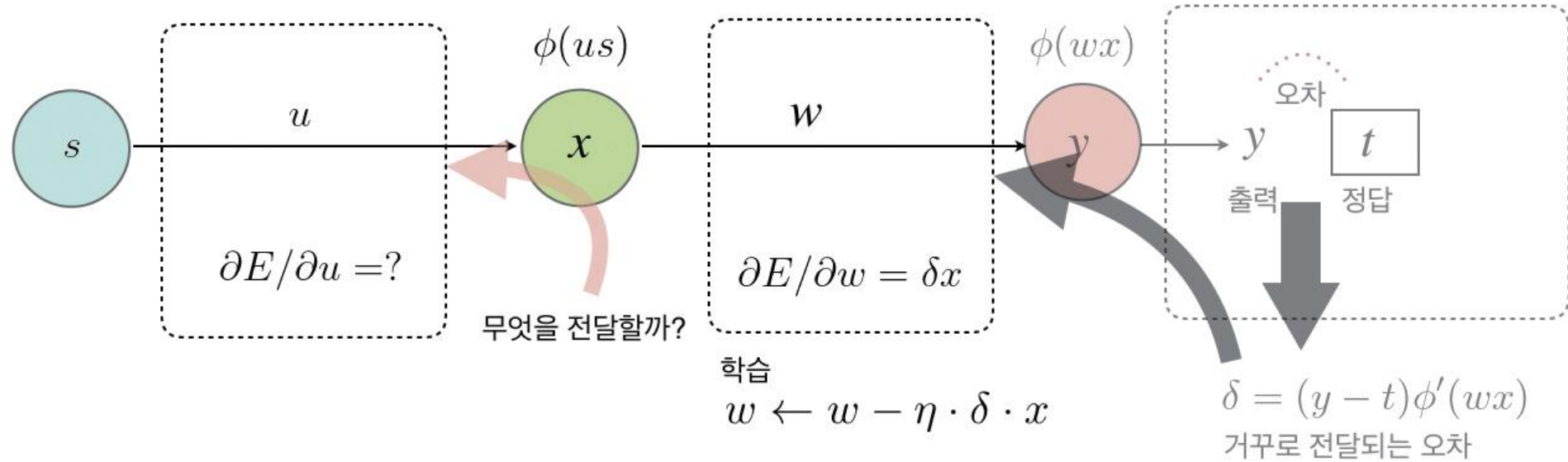
- 신경망의 각 연결강도에 적용하여 오차를 줄일 수 있다면, 복잡한 신경망도 학습이 가능

3.7 다층 퍼셉트론의 학습 - 오차 전파

- 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**(error backpropagation)의 기본 개념



- 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

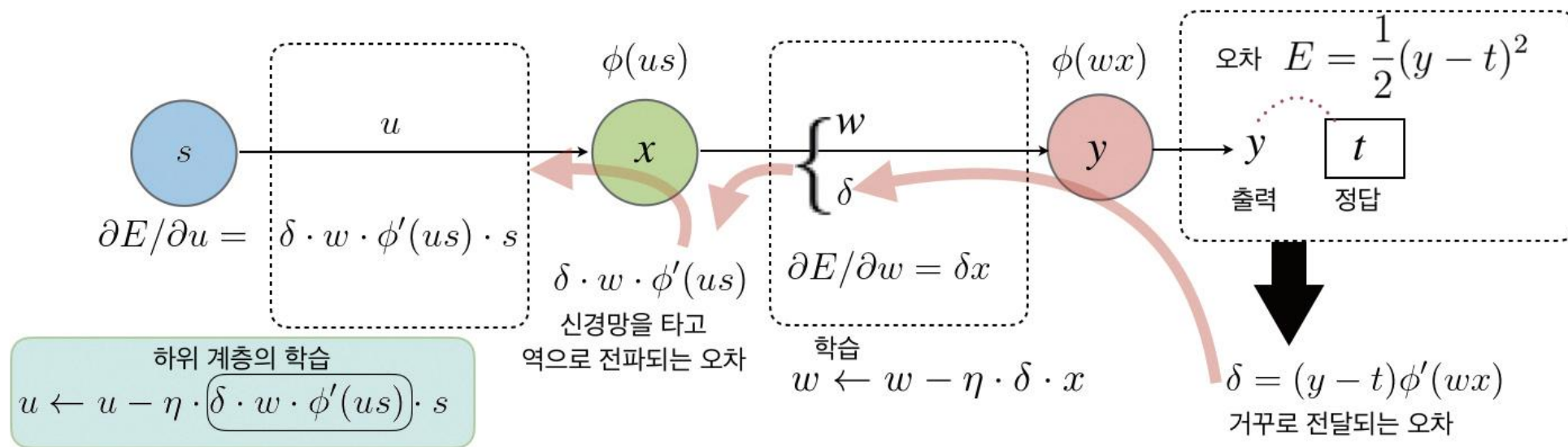
$$\partial E / \partial w = (y - t)y'x = \delta x$$

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

- 출력에 더 떨어진 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

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

- 순전파forward 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 함수를 거친 뒤에 연결강도에 의해 증폭되어 입력단까지 전달되는 과정으로, 이러한 신경망 학습 방법을 오차 역전파error backpropagation 알고리즘



3.8 TensorFlow

- 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** TensorFlow 라는 도구를 사용



TensorFlow

3.8 TensorFlow

- TensorFlow는 케라스^{keras}라는 심층신경망 구현을 위한 API가 포함되어 있는데, 이를 이용하면 간단히 신경망을 만들고 학습 가능

> **pip install tensorflow**



```
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
```

3.8 TensorFlow

- 모델은 신경 계층이 차례로 이어져 순차적으로 신호를 전달하는 모델
- 이런 모델은 Keras에서 제공하는 시퀀셜 Sequential 클래스로 구현
- 모든 층들은 이전 단계의 층이 가지는 신경세포 노드와 빠짐없이 다 연결
 - 이러한 연결을 밀집 dense 계층
 - 각 층이 가진 활성화 함수는 sigmoid



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

3.8 TensorFlow

- 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')
```

3.8 TensorFlow

- 모델을 훈련시킬 데이터



```
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, Epoch은 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
```

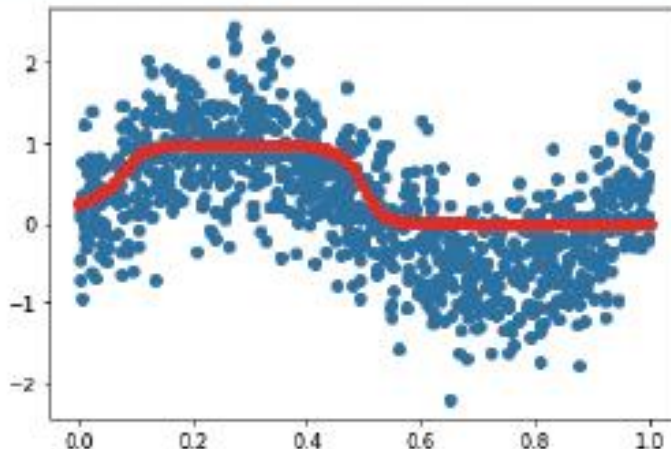
```
32/32 [=====] - 0s 1ms/step - loss: 0.3652
```

3.8 TensorFlow

- 테스트용 입력을 모델에 넣어서 예측
- Keras 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')
```



4. 심층신경망

4.1 활성화 함수의 다양화

- 서로 다른 초기화 방법이 제안된 이유는 활성화 함수에 따라 신호 전파가 다르기 때문
- 깊은 층을 가진 네트워크에서는 활성화 함수가 포화되지 않는 ReLU와 그 변형들이 오차 기울기를 잘 전달할 수 있는 것으로 알려짐

$$ReLU(x) = \max(0, x)$$

- ReLU는 쉽게 죽는다는 단점
 - 사용하는 노드가 0을 출력하게 되면, 해당 노드의 미분은 0 오차를 아래로 전달하지 못해 계속해서 0을 출력하는 노드로 남게 되는 것
 - 이 ReLU를 개선하기 위해 만들어진 활성화 함수들은 ReLU 변종 variants

4.1 활성화 함수의 다양화

- 첫 번째는 Leaky ReLU로 다음과 같은 식

$$\text{LeakyReLU}_a(x) = \max(ax, x), \quad 0 < a < 1$$

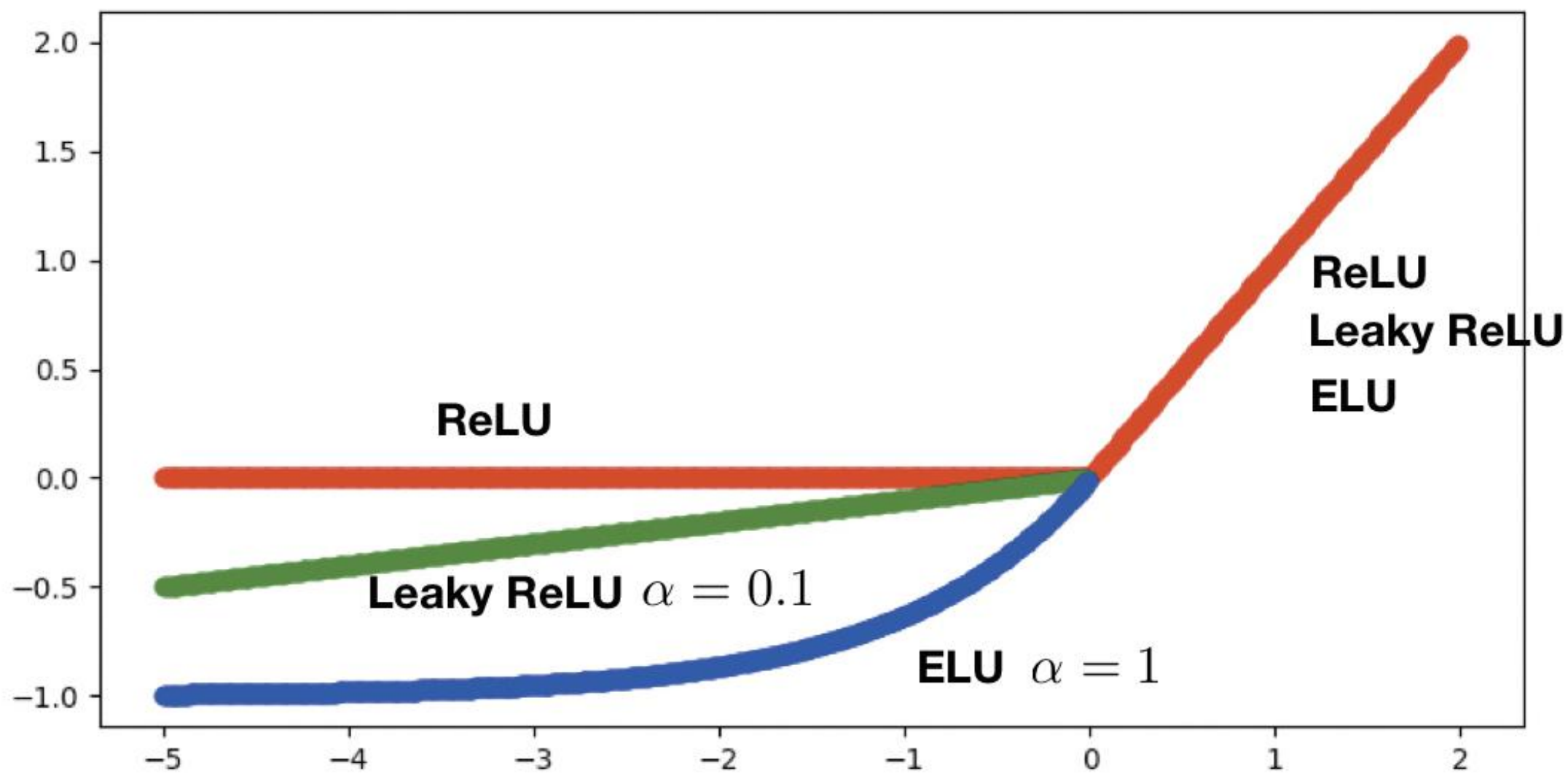
- 입력이 음수인 곳에서도 0을 출력하지 않고 1보다 작은 기울기로 약간의 신호를 내어 보냄
- 이 구간의 미분은 a 가 되고, 이 값은 0이 아니기 때문에 오차를 내려 보내어 연결강도 조정하는 일이 가능

4.1 활성화 함수의 다양화

- 허 초기화를 제안한 논문에서 함께 제안한 활성화 함수가 **PReLU**인데, 이것은 α 를 하이퍼 파라미터가 아니라 파라미터로 만들어 학습 과정에 이것도 함께 학습되게 하는 것
- 음수 구간에서 선형 모델을 사용하지 않고 지수 함수를 사용하는 다음과 같은 **ELU 모델**도 제안

$$ELU(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

- Leaky ReLU와 마찬가지로 음수 구간에서 음수 출력을 하여 사라지는 기울기 문제를 완화시키면서 α 가 1이면 함수가 어디서나 미분가능하다는 장점
- 경사 하강법의 동작이 튀지 않게 만들지만 계산 비용이 ReLU보다 크다는 점은 단점

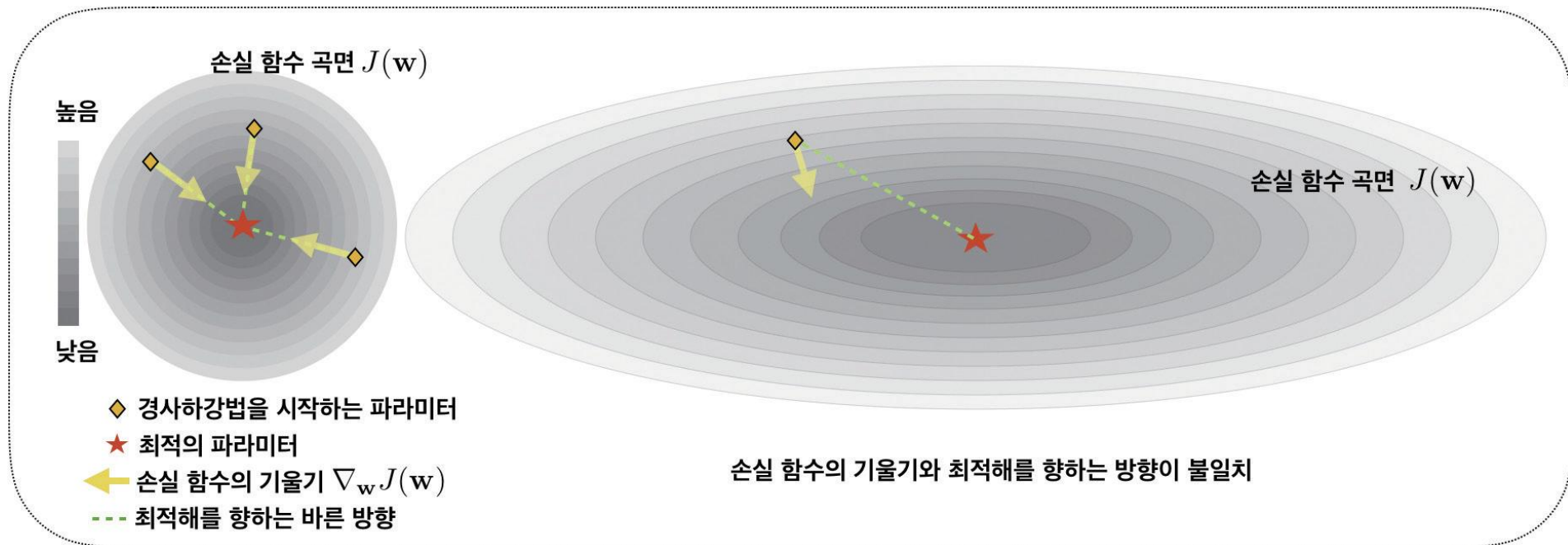


4.2 최적화 기법

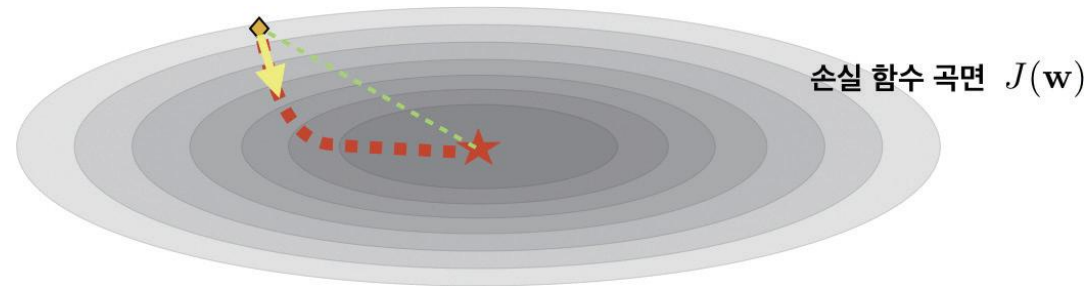
- 인공 신경망 모델의 학습에 사용되는 경사 하강법은 파라미터가 w 일 때, 오차를 측정하는 손실함수 $J(w)$ 를 정의하고, 파라미터에 대한 손실함수의 기울기를 따라 파라미터를 수정하는 방법
- 수정하는 정도가 학습률 η 이며 경사 하강법의 기본적인 수식 ($\nabla J(w)$ 는 파라미터 w 에 대한 손실함수의 기울기)

$$w \leftarrow w - \eta \nabla J(w)$$

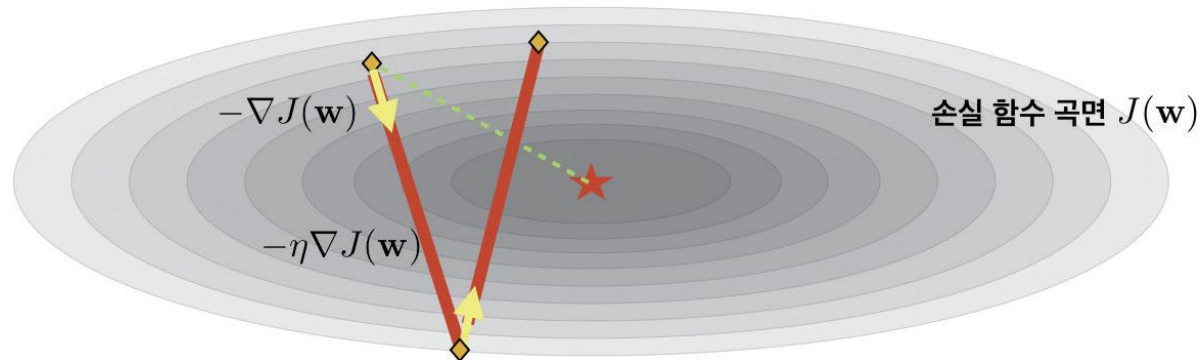
- 그림의 왼쪽처럼 오차 곡면이 최적해를 중심으로 하는 동심원 (3차원에서는 **구**^{sphere}, 4차원 이상에서는 **초구**^{hypersphere}가 될 것이다) 모양일 경우, 오차를 표현하는 손실 함수 곡면의 기울기의 반대방향은 언제나 최적해를 정확히 가리킴
- 곡면이 오른쪽과 같이 특정한 방향으로 늘어난 형태라고 생각하면 **기울기를 따라 가는 방법은 최적해를 향하는 가장 빠른 길을 따르지 못 함**



- 학습률 η 는 기울기를 따라 얼마나 이동할 것인지를 결정하는 하이퍼파라미터이며 작게 유지하면 기울기를 따라 조금씩 움직일 것이고, 그 결과는 아래 그림과 같이 급한 경사를 먼저 내려간 뒤에 골을 따라 최적해로 이동하게 될 것



- 학습률 η 가 너무 큰 값이 되면 다음과 같이 골을 여러번 건너며 빠르게 수렴하지 못 하거나, 심지어 발산해 버릴 수도 있음



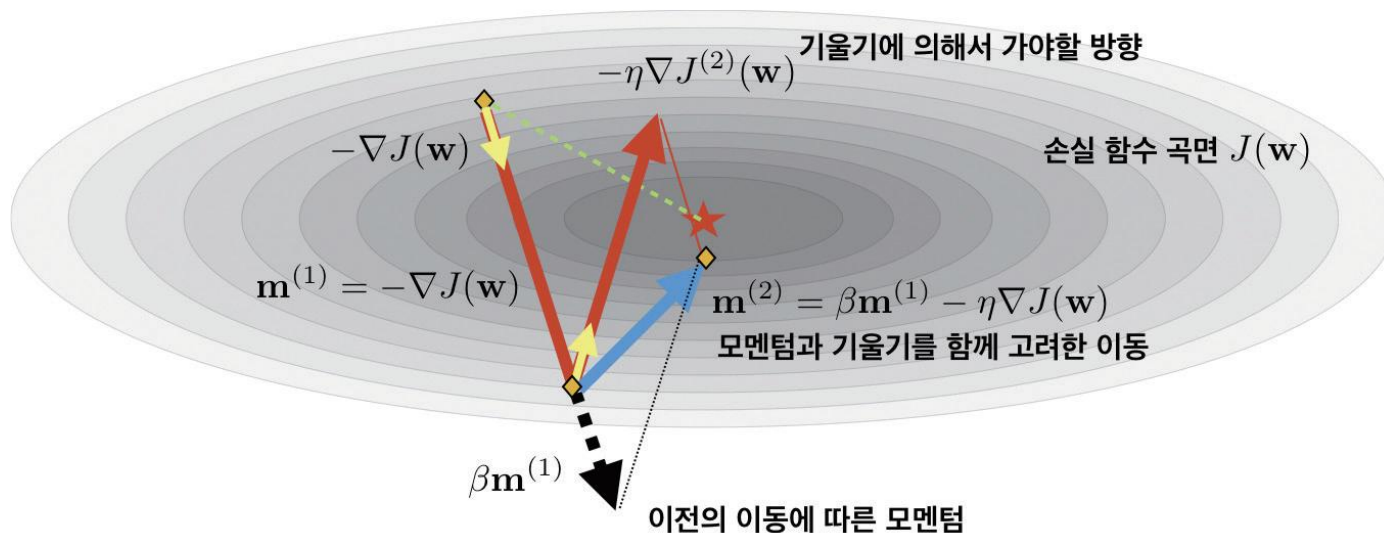
- 이러한 문제를 막는 방법은 η 를 안전하게 작은 값으로 두는 것
- 학습률을 작은 값으로 둔다는 것은 한 번 계산할 때 움직이는 거리가 작아지는 것
 - 이것은 최적해에 도달하기 위해 더 많은 기울기 계산을 해야 한다는 것을 의미
- 문제를 개선하고, 학습률을 크게 하면서 큰 걸음으로 최적해에 도달할 수 있도록 하는 다양한 변형 기법들이 등장

모멘텀 momentum 기법

- 이것은 현재 위치에서 계산된 기울기만으로 이동할 방향을 정하는 것이 아니라, 지금까지 **기울기를 따라 이동한 경로를 누적한 모멘텀 m** 을 정의하고, 이를 이용해 파라미터를 갱신
- 모멘텀은 매번 β 에 의해 감쇠하고 오차곡면의 새로운 기울기를 따라 이동하는 정도가 추가로 누적되

$$m \leftarrow \beta m - \eta \nabla J(m)$$

$$w \leftarrow w + m$$



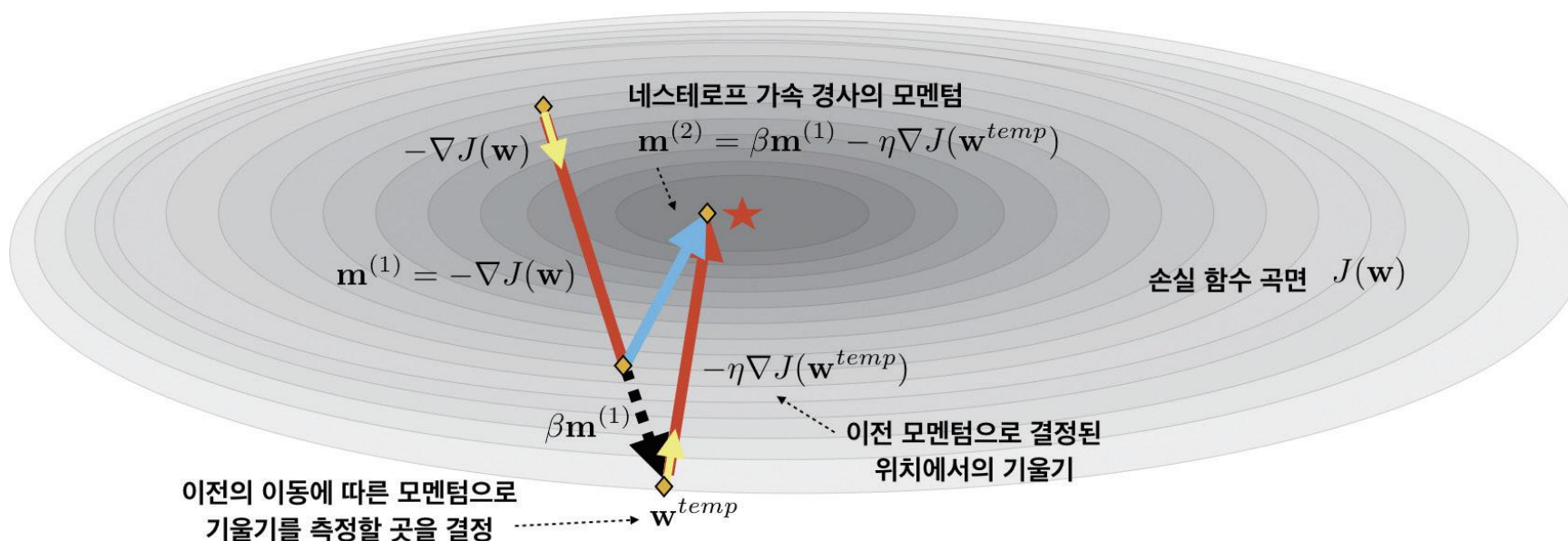
네스테로프 가속 경사 Nesterov accelerated gradient 기법

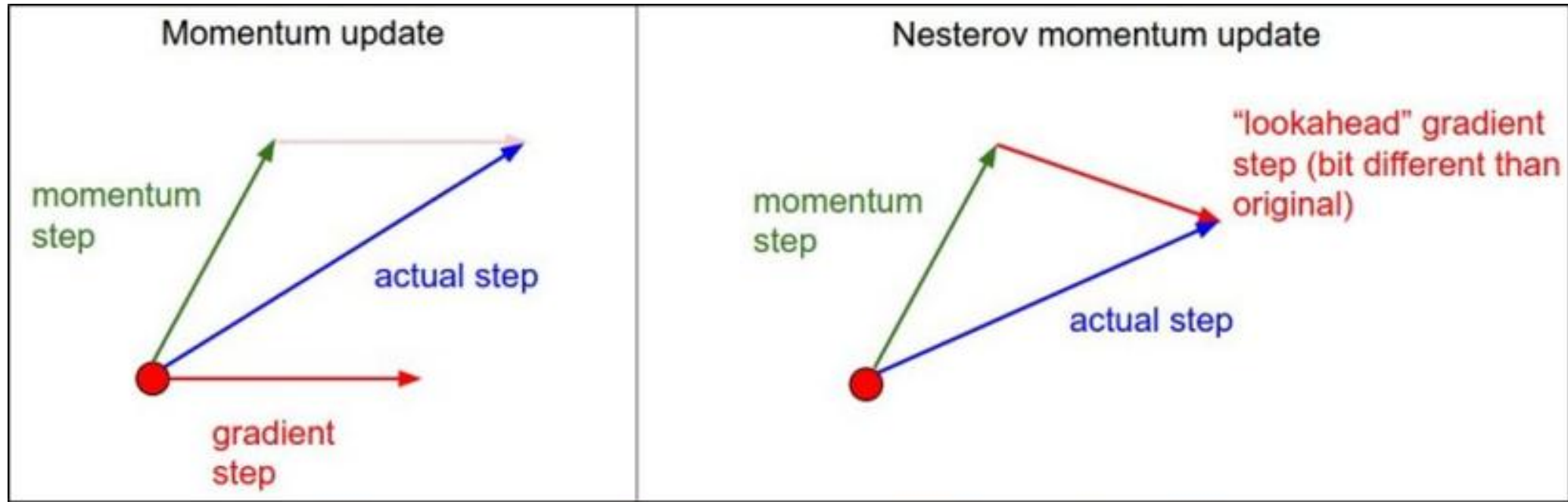
- 손실 함수 곡면의 기울기를 계산할 때 이전 모멘텀만큼 이동한 곳에서 기울기를 측정하여 이 기울기와 모멘텀을 함께 고려하여 최종적으로 이동할 곳을 찾는 것

$$\mathbf{w}^{temp} \leftarrow \mathbf{w} + \beta \mathbf{m}$$

$$\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla J(\mathbf{w}^{temp})$$

$$\mathbf{w} \leftarrow \mathbf{w}^{temp} + \mathbf{m}$$





Difference between Momentum and NAG. Picture from CS231.

- 변형된 경사 하강법이 필요한 이유는 이것은 손실 함수 곡면이 동그란 공 모양이 아니고 특정한 축 방향으로 늘어나 있기 때문이라고 생각할 수 있다. 길게 늘어난 있는 축 방향으로 손실 함수 곡면의 크기를 줄여주면 더 좋은 수렴 성능을 보일 수 있을 것
 - 적응적 경사adaptive gradient 기법인 AdaGrad
- 기울기를 구할 때마다 기울기 벡터의 i 번째 요소의 제곱값을 구해서 s_i 누적한다. 이 누적값은 해당 차원의 경사가 급할수록 큰 값이 되므로 기울기를 따라 파라미터를 이동할 때 이 s_i 값으로 학습률을 조정하여 경사가 급한 곳은 천천히 이동하게 하면 손실 함수 곡면의 왜곡을 보정하는 역할
- s_i 의 제곱근으로 학습률을 나누는 방식으로 간단히 구현할 수 있다. 수식은 아래와 같다. ϵ 은 0으로 나누는 일을 막기 위한 작은 상수

$$s_i \leftarrow s_i + \nabla J(\mathbf{w})_i^2$$

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \frac{\eta}{\sqrt{s_i + \epsilon}} \nabla J(\mathbf{w})_i$$

- 이 방법은 손실함수 곡면의 왜곡을 수정하지만, s 는 지속적으로 증가하게 된다.
 - 학습의 후반부에 학습률은 큰 수로 나누어지기 때문에 점점 해를 찾는 속도가 느려짐
- 이런 문제를 해결한 방법인 RMSProp 알고리즘을 소개
- 이 기법은 간단히 이전 s_i 가 일정한 비율로 쇠퇴하게 만들고 새로운 값이 쇠퇴한 만큼 추가되도록 하는 것으로 쇠퇴 비율 ρ 는 보통 0.9 정도를 사용

$$s_i \leftarrow \rho \cdot s_i + (1 - \rho) \cdot \nabla J(\mathbf{w})_i^2$$



- 현재 가장 많이 사용되는 최적화 방법 중에 하나는 모멘텀 최적화와 RMSProp을 섞어서 사용하는 적응적 모멘텀 추정 기법^{adaptive momentum estimation}, 일반적으로 Adam이라 부르는 기법

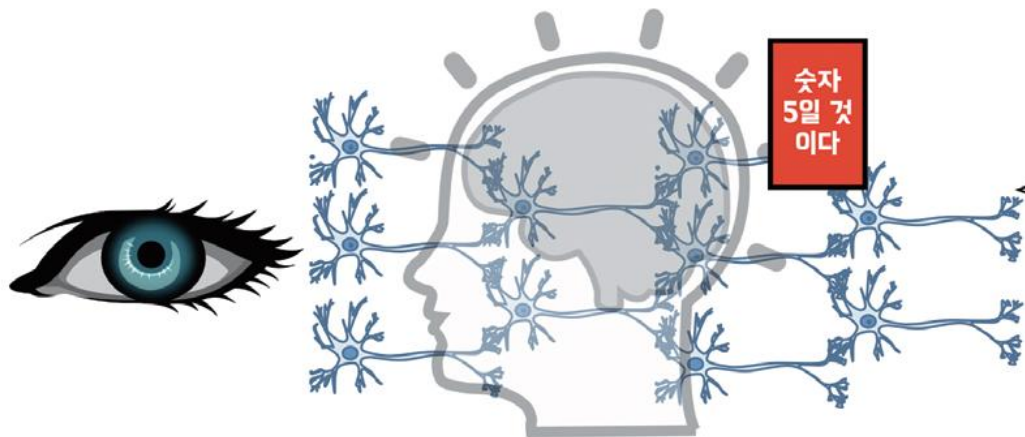
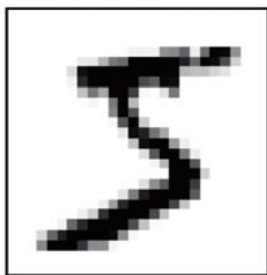
4.3 텐서플로우 (TensorFlow)

- 머신러닝과 딥러닝을 위한 프레임워크로는 **텐서플로우**, **테아노**^{theano}, **파이토치**^{pytorch}, **mxnet**, **케라스**^{keras}, **사이킷런**^{scikit-learn}, **NLTK** 등
- 텐서플로우**^{TensorFlow}는 머신러닝과 딥러닝을 위한 오픈소스 프레임워크로 구글의 인공지능 개발부서에서 개발하여 내부적으로 사용하다가 2015년에 오픈소스로 공개되어 현재 텐서플로우2 버전으로 발전



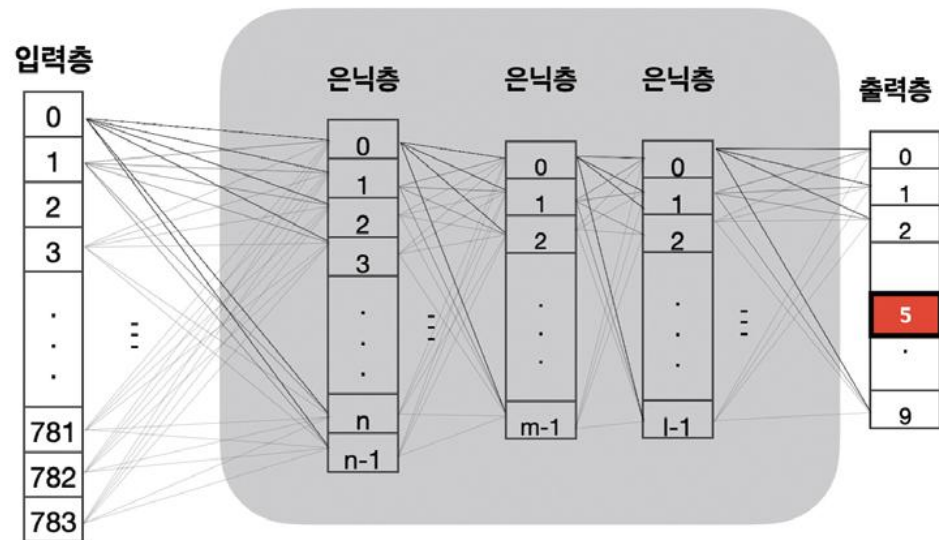
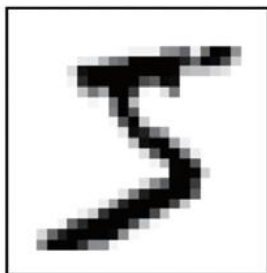
4.3 텐서플로우 (TensorFlow)

- 텐서플로우 역시 기계학습을 위한 많은 학습 데이터를 제공하고 있는데, 이 중 하나인 MNIST 데이터를 사용
- 미국 국립표준 연구소 National Institute of Standards and Technology에서는 손으로 쓴 숫자들로 이루어진 글씨 데이터를 스캐닝하여 다양한 화상처리 시스템에서 사용하기 위하여 제공하고 있는데 이 데이터가 바로 **MNIST** Modified National Institute of Standards and Technology 데이터베이스



무수히 많은 신경세포들의 자극의 조합이 숫자 5를 인식하는 과정입니다.

인간의 신경세포와 인식의 과정



많은 노드들과 이를 연결하는 가중치를 곱한 결과를 출력층에 내보냅니다. 출력 노드에는 예측 값이 전달되며, 숫자 5에 대한 예측값이 가장 높게 나타나도록 학습을 진행합니다.

입력의 처리를 위한 다수의 은닉층을 추가할 수 있음

다층 신경망과 인식의 과정

4.4 MNIST 예제



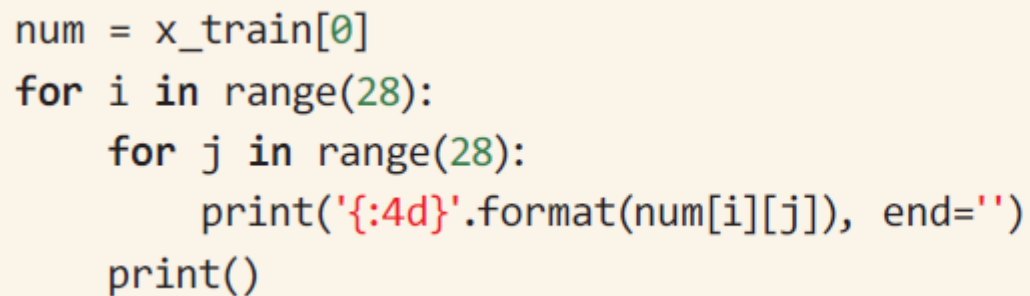
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

print('x_train 데이터의 형태:', x_train.shape)
print('x_train[0] 데이터의 형태:', x_train[0].shape)
print('y_train 데이터의 형태:', y_train.shape)
```

```
x_train 데이터의 형태: (60000, 28, 28)
x_train[0] 데이터의 형태: (28, 28)
y_train 데이터의 형태: (60000,)
```

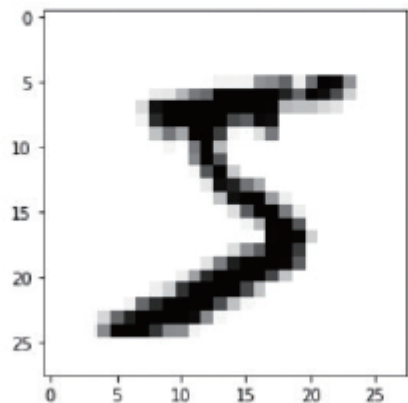


0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0
0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	225	172	253	242	195	64	0	
0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	
0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	35	241	225	160	108	1	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0		
0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0		
0	0	0	0	0	0	0	0	23	66	213	253	253	253	198	81	2	0	0	0	0	0	0		
0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0		
0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0		
0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0		
0	0	0	0	0</																				

- 출력 결과는 대부분의 수가 0의 값을 가지며 최소값이 0이고, 최대값이 255인 정수 데이터임을 알 수 있음
 - 이 데이터의 형태가 기울어진 5와 비슷
 - 이제 다음과 같은 방법으로 부드럽게 보간된 회색조 이미지로 만들어 이 배열의 의미를 이해하자



```
plt.imshow(num, cmap='Greys', interpolation='nearest')
```



- 이 배열 데이터는 숫자 5의 손글씨와 비슷한 이미지 임을 알 수 있는데, 이 이미지의 의미는 다음과 같이 `y_train[0]` 배열에 5라고 레이블링 되어 있다.



```
print('y_train[0] =', y_train[0])
```

```
y_train[0] = 5
```

4.5 심층 신경망 구축

- 이제 MNIST 데이터를 사용하여 우리가 할 일
 1. 6만 개의 이미지로 이루어진 `x_train` 데이터를 심층 신경망 모델에 넣어서 `y_train` 데이터의 숫자로 인식하도록 학습을 시키는 일
 2. 이 때 노드의 활성화 함수, 학습을 위한 최적화 함수, 손실 함수, 측정 방법을 정의하는 일
 3. 1만 개의 이미지로 이루어진 `x_test` 데이터를 학습을 마친 모델의 입력으로 넣어서 이 모델의 정확도를 알아보는 일
- 심층 신경망을 Numpy를 통해서 구현하는 것은 많은 노력과 시간이 필요하며, 코드의 이식성도 떨어지기 때문에 TensorFlow의 Keras 사용

- 가장 상단에 $x_train / 255$, $x_test / 255$ 는 원래 입력값의 범위 0~255를 0에서 1사이의 값으로 조정하는 정규화(normalization) 단계



```
x_train, x_test = x_train / 255, x_test / 255 # 입력값 정규화
```

```
model = keras.models.Sequential( [  
    keras.layers.Flatten(input_shape = (28, 28)),  
    keras.layers.Dense(256, activation = 'relu'),  
    keras.layers.Dense(10, activation = 'softmax'),  
])
```

```
# 학습을 위한 최적화 함수, 손실 함수등을 가진 모델을 컴파일
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',  
metrics=['accuracy'])  
model.fit(x_train, y_train, epochs = 5)
```

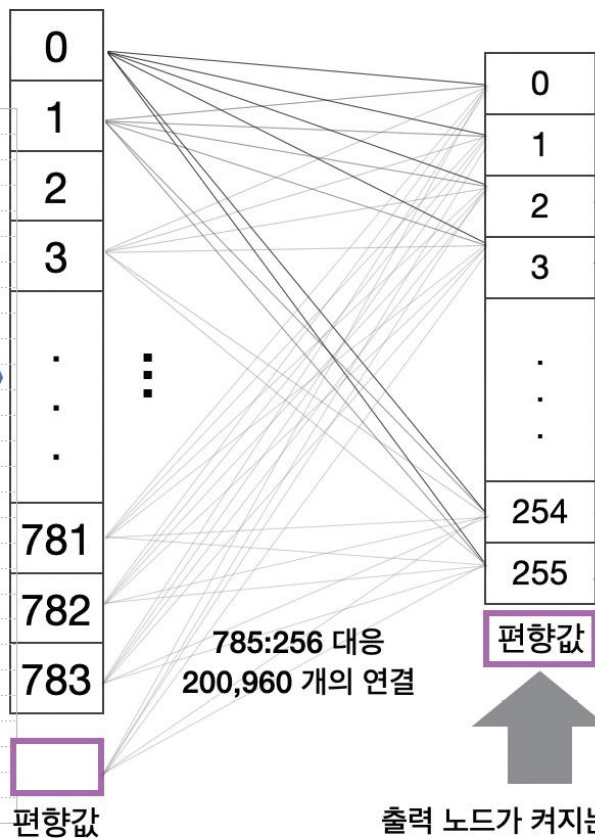
- Sequential() 모델을 사용
 - 여러 개의 층으로 이루어진 순차적 모델로 각 입력층은 하나의 입력 텐서와 출력 텐서를 가짐
- 28x28 크기의 이미지는 1차원 배열로 변환한 경우 784개의 값이 되는데, 이 값이 신경 회로망을 통과하여 10개 중 하나의 범주category로 분류
- 학습층을 만드는 명령이 keras.layers.Dense()이며 Dense는 학습을 위한 연결을 밀집된dense 구조의 네트워크 층 혹은 완전 연결fully connected층으로 한다는 의미
- 최종적으로 10개의 카테고리에 입력이 연결되도록 하는 Keras 모델을 keras.Sequential()로 생성
- TensorFlow와 Keras를 이용하면 복잡한 노드와 그 구조를 손쉽게 생성

Flatten
2차원 입력 28x28 배열을
784개의 1차원 배열로 변환

입력: 28x28 픽셀

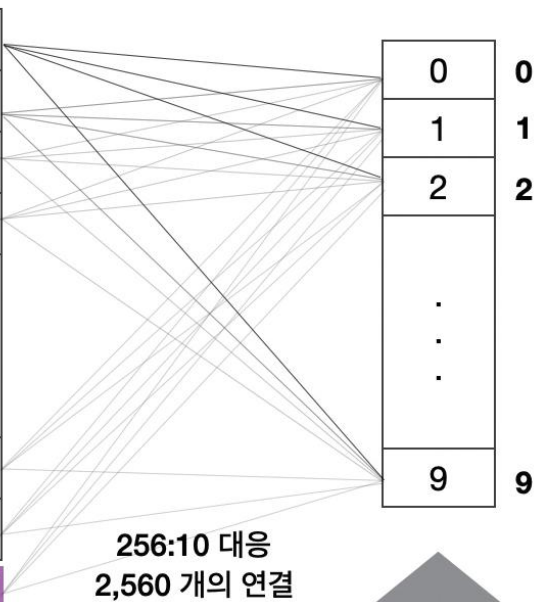


Dense
785개 입력이 256개 출력에
촘촘히 연결됨



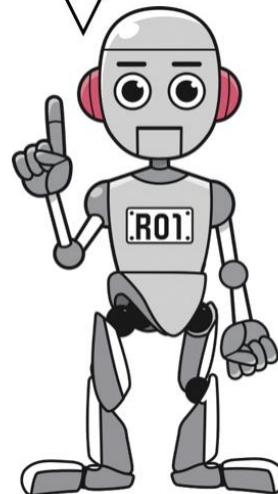
출력 노드가 켜지는 방법은
입력을 모아
ReLU를 적용

Dense
256개 입력이 10개 출력에
촘촘히 연결됨



출력 노드가 켜지는 방법은
입력을 모아
SoftMax를 적용

10개의 범주로 분류하기 위해
입력에서 출력까지 연결되어 있죠.
이 연결의 강도를 변화시킬 수 있는
순차적 구조를 만들어 보아요.



4.6 신경망 최적화

- 순차 모델은 compile() 함수를 통해서 학습을 위한 하이퍼파라미터 등의 환경을 설정하는 단계를 거침
- compile() 함수에서 모델의 하이퍼파라미터로
 - 최적화 함수를 무엇으로 할 것인지를 설정하는 optimizer
 - 손실 함수는 어떤 것을 사용할지를 설정하는 loss
 - 훈련기간동안 어떠한 측정 도구를 사용하여 정답 레이블과의 차이를 측정할 지를 결정하는 metrics 등이 있음
- 우리는 최적화 함수로 Adam 알고리즘, 손실 함수로 Categorical Cross Entropy 함수, 측정 기법으로 **정확도** accuracy 측정기를 사용

- 인공 신경망 모델을 만들고 `model.fit()` 함수를 통해서 최적화를 수행
 - 인자로 `x_train`, `y_train`을 주고 `epoch`이라는 추가적인 인자를 부여
 - **에폭**epoch은 전체 훈련 데이터를 모두 신경망에 넣어서 학습을 수행시키는 것을 말하는데, 여기서는 6만 개의 훈련 데이터를 순전파 전달 과정과 역전파 알고리즘에 적용
- 학습에 사용되는 데이터 6만 개를 한꺼번에 학습시켜 가중치를 갱신하는 방법은 가중치 계산에 시간이 많이 걸리고 가중치 갱신도 느려 비효율적
 - 일부 데이터를 골라서 학습을 시킬 수 있는데 이것이 미니 배치mini-batch 학습 방식이며 미니 배치의 크기는 디폴트로 32를 사용
 - 1,875번의 가중치 갱신을 해야 6만 개 데이터 전체에 대한 학습이 완료된다. 이것이 바로 1 에폭



- 만든 신경망 모델이 어떤 형태를 가지고 있으며, 몇 개의 파라미터를 훈련 시켜서 만든 모델인가를 `model.summary()` 함수를 통해서 확인



`model.summary()`

Model: "sequential"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

flatten (Flatten)	(None, 784)	0
-------------------	-------------	---

dense (Dense)	(None, 256)	200960
---------------	-------------	--------

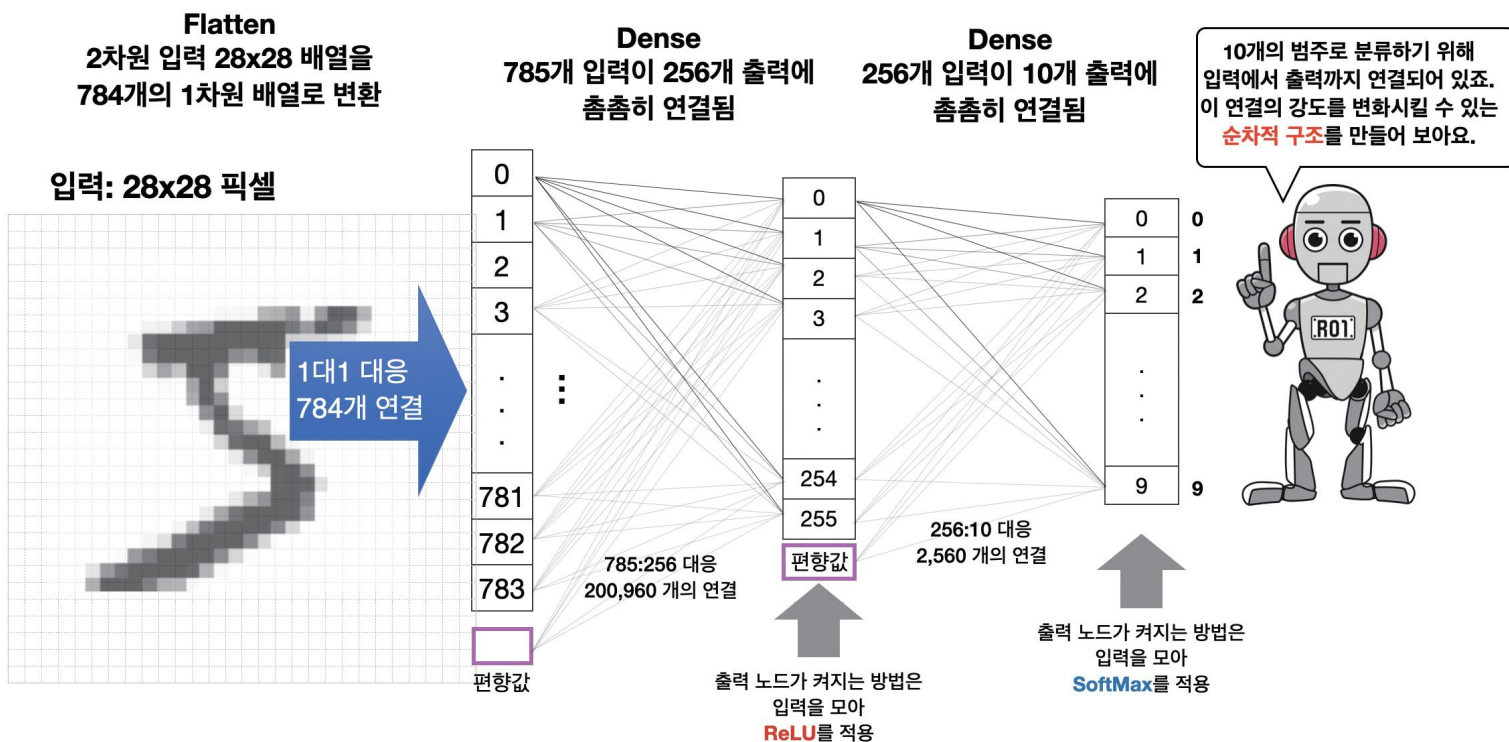
dense_1 (Dense)	(None, 10)	2570
-----------------	------------	------

Total params: 203,530

Trainable params: 203,530

Non-trainable params: 0

- 이때 입력 데이터가 첫번째 은닉층을 통과할때 모두 784개의 입력과 1개의 편향값(784 + 1)에 대하여 256개의 은닉층 노드간의 연결이 필요하므로 $785 * 256 = 200,960$ 개의 파라미터를 훈련시켜야함
- 256개의 은닉층과 10개의 출력층을 연결하는 파라미터에도 편향값이 하나 더 추가되어 2,570개가 훈련 파라미터가 필요
 - 우리가 학습시켜야 할 전체 파라미터는 $200,960 + 2,570 = 253,530$ 개의 매우 많은 파라미터



- 이제까지 학습에 사용하지 않은 1만 개의 숫자 이미지가 있는 x_test를 이 모델에 넣어서 y_test와 비교



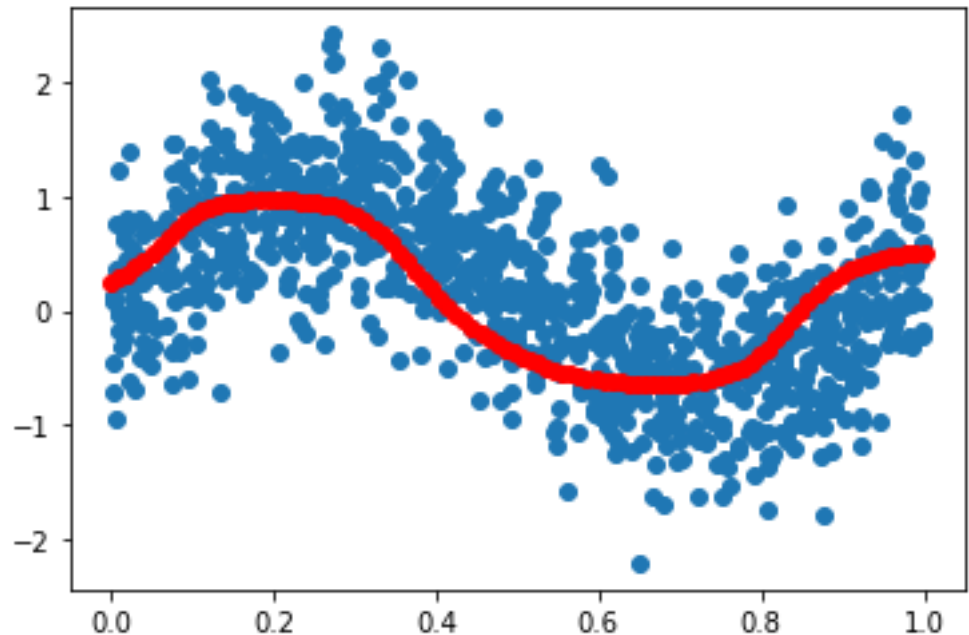
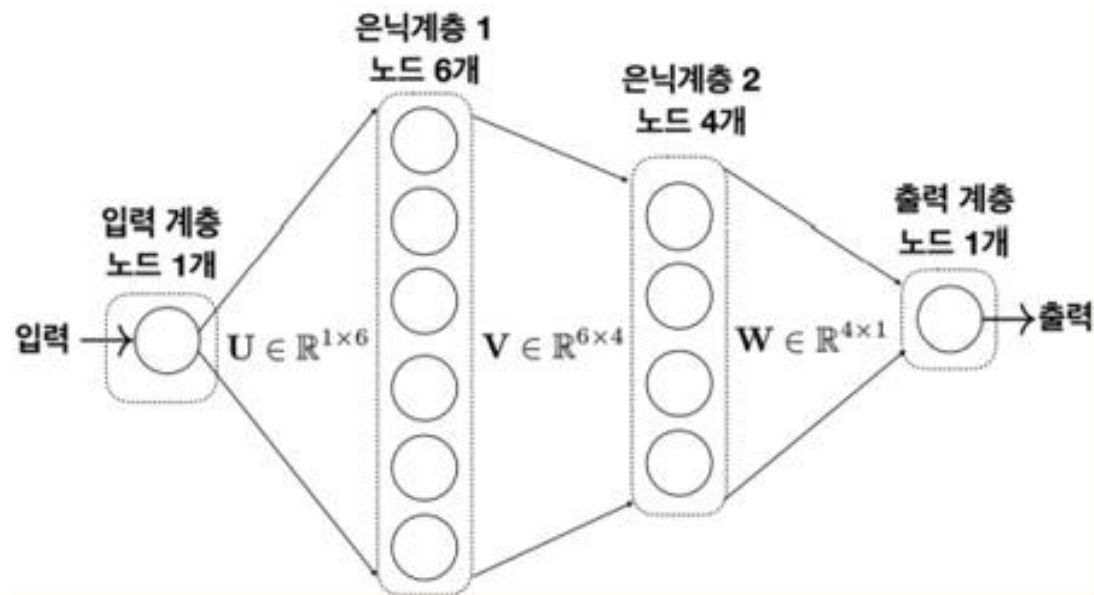
```
print('신경망 모델의 학습 결과 :')  
eval_loss, eval_acc = model.evaluate(x_test, y_test)  
print('test 데이터의 손실값', eval_loss, 'test 데이터의 정확도', eval_acc)
```

신경망 모델의 학습 결과 :

```
313/313 [=====] - 1s 2ms/step - loss: 0.2734 -  
accuracy: 0.9213  
test 데이터의 손실값 0.27340787649154663 test 데이터의 정확도 0.9212999939918518
```


실습문제

- 아래 그림과 같은 다층 퍼셉트론을 주어진 데이터와 Keras를 이용해 구현하고 아래 그래프를 그려라.
 - "dataset2.csv" 데이터 사용
 - 곡선의 범위는 0과 1 사이에 100개
 - learning rate = 0.1, epoch = 100
 - 활성화함수: tanh (hyperbolic tangent)



감사합니다