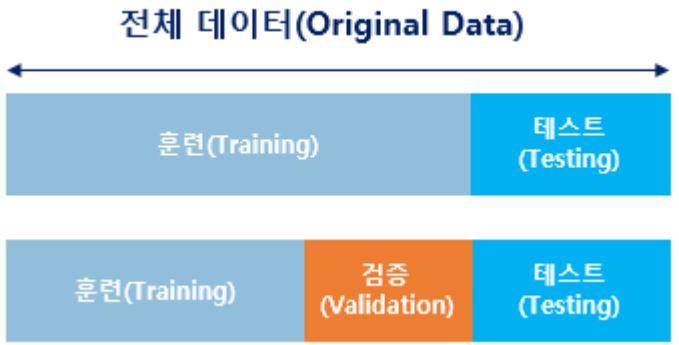
7. 인공신경망

1. 딥러닝 용어 이해
2. 퍼셉트론(Perceptron)
3. XOR 문제
4. 역전파 알고리즘
5. 그래디언트 손실 문제와 렐루(ReLU)
6. XOR 문제 - 다층 퍼셉트론 구현하기
7. 비선형 활성화 함수(Activation function)
8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류실습
9. 과적합(Overfitting)을 막는 방법들

1. 머신러닝 용어 이해

❖ 머신러닝 모델 평가



- 모델 평가 방법
  - 데이터를 훈련용, 검증용, 테스트용 세 가지로 분리하여 사용
  - **훈련용 데이터** : 모델 학습을 위해 사용
  - **테스트용 데이터** : 학습된 모델의 성능을 평가하기 위해 사용
  - **검증용 데이터** : 학습 시에 **모델 성능을 조정**하기 위한 용도, 즉, 학습 모델에 **과적합**이 되고 있는지 판단하기거나 **하이퍼파라미터 조정**을 위해 사용
  - **하이퍼파라미터** : 모델의 성능에 영향을 주는 매개변수, **사용자가 직접 정해주는 변수**(예: 경사 하강법에서 학습률(learning rate), epoch 수, 딥 러닝에서 은닉층의 수, 뉴런의 수, 드롭아웃 비율 등)
  - **매개변수** : 사용자가 결정해주는 값이 아니라 **모델이 학습하는 과정에서 얻어지는 값**(가중치(W), 편향(b))
  - **훈련용 데이터**로 **훈련**을 시킨 모델은 **검증용 데이터**를 사용하여 **정확도**를 검증하며 **하이퍼파라미터**로 **튜닝(tuning) 하고, 모델 평가**는 훈련에 참가하지 않는 새로운 **테스트 데이터** 사용

1. 머신러닝 용어 이해

❖ 분류(Classification)와 회귀(Regression)

- **이진 분류 문제(Binary Classification)**
  - 주어진 입력에 대해서 둘 중 하나의 답을 정하는 문제
  - 시험 성적에 대해서 합격, 불합격인지 판단하고 메일로부터 정상 메일, 스팸 메일인지를 판단하는 문제 등
- **다중 클래스 분류(Multi-class Classification)**
  - 주어진 입력에 대해서 세 개 이상의 정해진 선택지 중에서 답을 정하는 문제
  - 예 : 필기체 숫자 이미지 분류(입력에 대한 10개의 클래스 중 하나로 판단)
- **회귀 문제(Regression)**
  - 주어진 입력에 대한 결과가 연속된 결과 값을 가짐
  - 예 : 공부 시간에 비례한 시험 성적, 아파트 평수에 비례한 집 값 등

1. 머신러닝 용어 이해

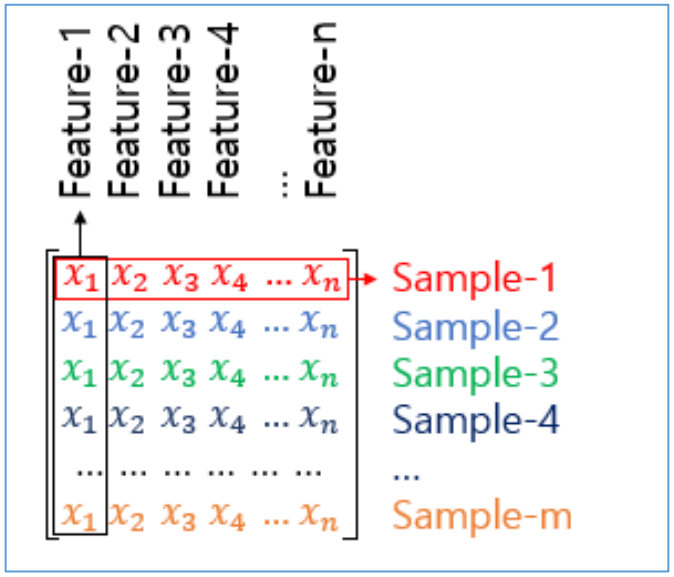
❖ 지도 학습(Supervised Learning)과 비지도 학습(Unsupervised Learning)

- **지도 학습**
  - 레이블(Label)이라는 정답과 함께 학습하는 것
  - 예측 값과 실제 값의 차이인 오차를 줄이는 방식으로 학습
- **비지도 학습**
  - 기본적으로 목적 데이터(또는 레이블)이 없는 학습 방법
  - 군집(clustering)이나 차원 축소
- **강화 학습**
  - 어떤 환경 내에서 정의된 에이전트가 현재의 상태를 인식하여, 선택 가능한 행동들 중 보상을 최대화하는 행동 혹은 행동 순서를 선택하는 방법

1. 머신러닝 용어 이해

❖ 샘플(Sample)과 특성(Feature)

- 많은 머신 러닝 문제가 1개 이상의 독립 변수 x를 가지고 종속 변수 y를 예측하는 문제
- 많은 머신 러닝 모델들, 특히 인공 신경망 모델은 독립 변수, 종속 변수, 가중치, 편향 등을 행렬 연산을 수행함
- 독립 변수 x의 행렬을 X라고 하였을 때, 독립 변수의 개수가 n개이고 데이터의 개수가 m인 행렬 X는 그림과 같다.



- 머신 러닝에서는 하나의 데이터, 즉, 하나의 행을 샘플(Sample)이라고 부름
- 종속 변수 y를 예측하기 위한 각각의 독립 변수 x를 특성(Feature)이라고 부름.

1. 머신러닝 용어 이해

❖ 혼동 행렬(Confusion Matrix)

- 머신 러닝에서는 맞춘 문제 수를 전체 문제 수로 나눈 값을 정확도(Accuracy)라고 함
- 정확도(Accuracy)는 맞춘 결과와 틀린 결과에 대한 세부적은 내용을 보여주지 않음
- 정확도를 맞춘 결과와 틀린 결과에 대한 세부적인 내용을 보여주기 위해서 사용하는 것이 혼동 행렬(Confusion Matrix)임.
- 예: 양성(Positive)과 음성(Negative)을 구분하는 이진 분류를 하였을 때 혼동 행렬은 다음과 같다.

-	참	거짓
참	TP	FN
거짓	FP	TN

TP(True Positive) : 양성을 양성으로 대답한 경우(정답을 맞춤)

TN(True Negative) : 음성을 음성이라고 대답한 경우 (정답을 맞춤)

FP(False Postivie): 음성을 양성이라고 대답한 경우(정답을 맞추지 못함)

FN(False Negative) : 양성을 음성이라고 대답한 경우(정답을 맞추지 못함)

1. 머신러닝 용어 이해

❖ 혼동 행렬(Confusion Matrix)

- **정밀도(Precision)**
  - 정밀도은 양성이라고 대답한 전체 케이스에 대한 TP의 비율

$$정밀도 = \frac{TP}{TP + FP}$$

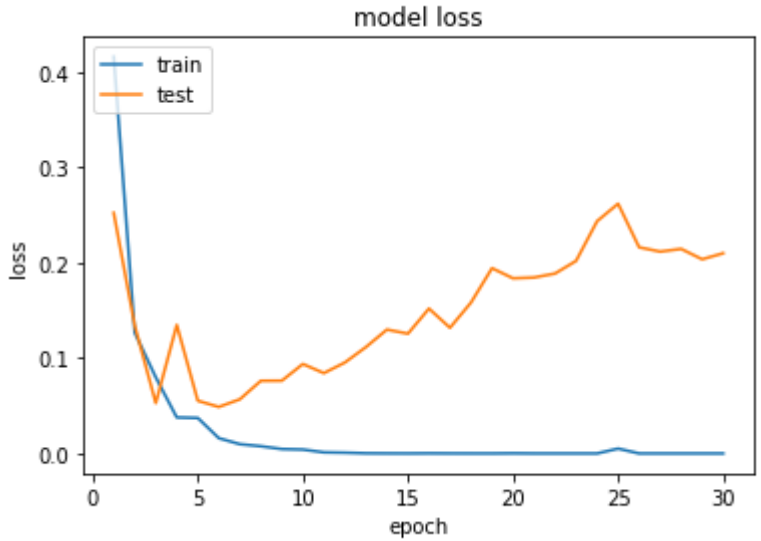
- **재현률(Recall)**
  - 실제 값이 양성인 데이터의 전체 개수에 대해서 TP의 비율
  - 즉, 양성인 데이터 중에서 얼마나 양성인지를 예측(재현)했는지를 나타냄

$$재현률 = \frac{TP}{TP + FN}$$

1. 머신러닝 용어 이해

❖ 과적합(Overfitting)과 과소 적합(Underfitting)

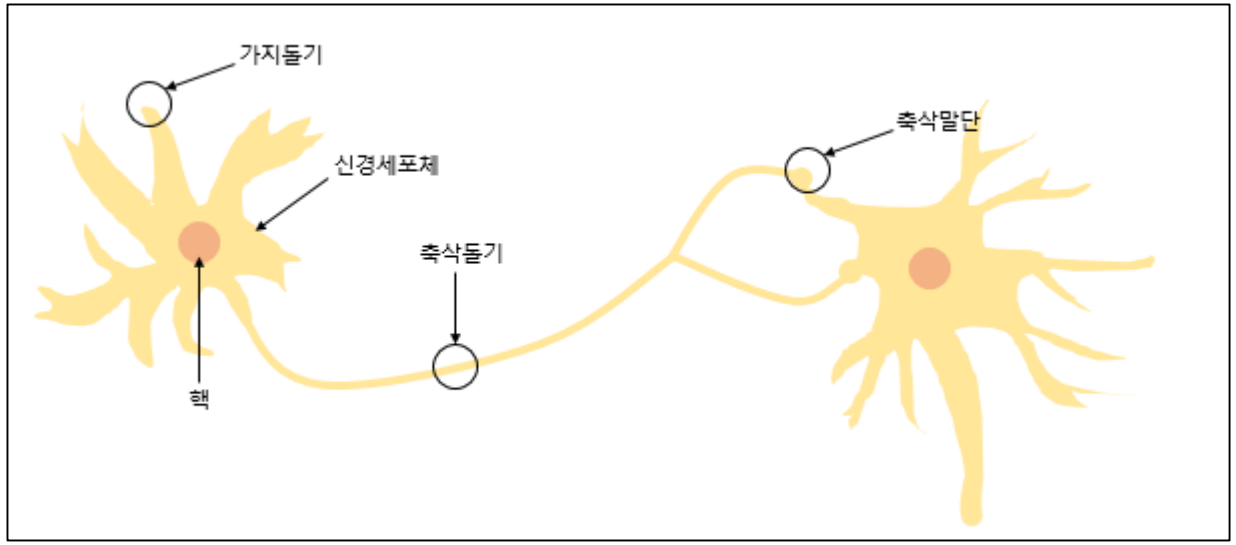
- **과적합(Overfitting)**
  - 훈련 데이터를 과하게 학습한 경우를 말함
  - 훈련 데이터는 실제로 존재하는 많은 데이터의 일부에 불과함
  - 기계가 훈련 데이터에 대해서만 과하게 학습하면 테스트 데이터나 실제 서비스에서의 데이터에 대해서는 정확도가 좋지 않은 현상이 발생함
- 과소적합(Underfitting)
  - 과적합 방지를 위해 테스트 데이터의 성능이 낮아지기 전에 훈련을 멈추는 것이 바람직함
  - 테스트 데이터의 성능이 올라갈 여지가 있음에도 훈련을 덜 한 상태
  - 과소 적합은 훈련 자체가 부족한 상태



2. 퍼셉트론(Perceptron)

❖ 퍼셉트론(Perceptron)

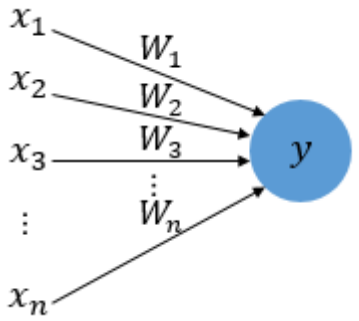
- 프랑크 로젠블라트(Frank Rosenblatt)가 1957년에 제안한 초기 형태의 인공 신경망
- 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘
- 퍼셉트론은 실제 뇌를 구성하는 신경 세포 뉴런의 동작과 유사
- 신경 세포 뉴런은 가지돌기에서 신호를 받아들이고, 이 신호가 일정치 이상의 크기를 가지면 축삭돌기를 통해서 신호를 전달



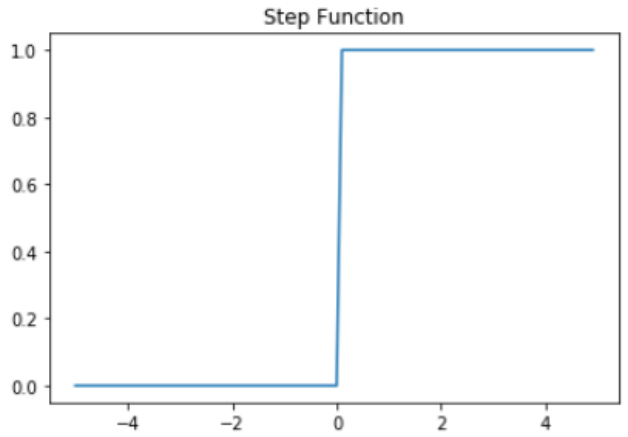
2. 퍼셉트론(Perceptron)

❖ 퍼셉트론(Perceptron)

- 신경 세포 뉴런의 입력 신호와 출력 신호가 **퍼셉트론**에서 **입력 값(x), 가중치(W), 출력 값(y)**, **뉴런(원)**에 해당 됨
- 신경 세포 뉴런에서 신호를 전하는 **축삭돌기** 역할을 퍼셉트론에서는 **가중치**가 함
- 각각의 **인공 뉴런**에서 보내진 **입력 값 x**는 각각의 **가중치 W**와 함께 종착지인 **인공 뉴런**에 전달됨.
- 입력 값에 대한 가중치의 값이 크면 클수록 해당 입력 값이 중요하다는 것을 의미
- 각 입력 값이 가중치와 곱해져서 인공 뉴런에 보내지고,
- 각 입력 값과 가중치의 곱의 **전체 합이 임계치(threshold)**를 넘으면
- 종착지 **인공 뉴런은 출력 신호 1**을 출력하고, 그렇지 않을 경우에는 **0**을 출력
- 이러한 함수를 **계단 함수(Step function)**라고 함.



다수의 입력을 받는 퍼셉트론의 그림



계단 함수 그래프

2. 퍼셉트론(Perceptron)

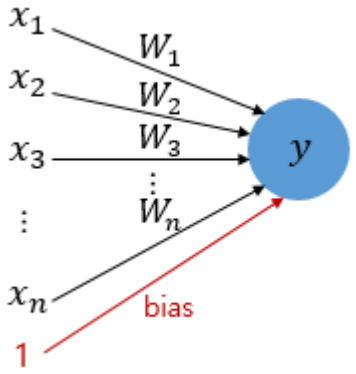
❖ 퍼셉트론(Perceptron)

- 계단 함수에 사용된 임계치 값을 수식으로 표현할 때는 보통 **세타(Θ)**로 표현
- 임계치 식에서 임계치를 좌변으로 넘기고 편향 b(bias)로 표현할 수도 있음
- 편향 b 또한 퍼셉트론의 입력으로 사용됨
- 그림으로 표현할 때는 입력값이 1로 고정되고 편향 b가 곱해지는 변수로 표현

$$if \sum_{i}^{n} W_i x_i \geq \theta \rightarrow y = 1$$

$$if \sum_{i}^{n} W_i x_i < \theta \rightarrow y = 0$$

계단 함수에 사용된 임계치 값 수식



편향이 추가된 퍼셉트론

$$if \sum_{i}^{n} W_i x_i + b \geq 0 \rightarrow y = 1$$

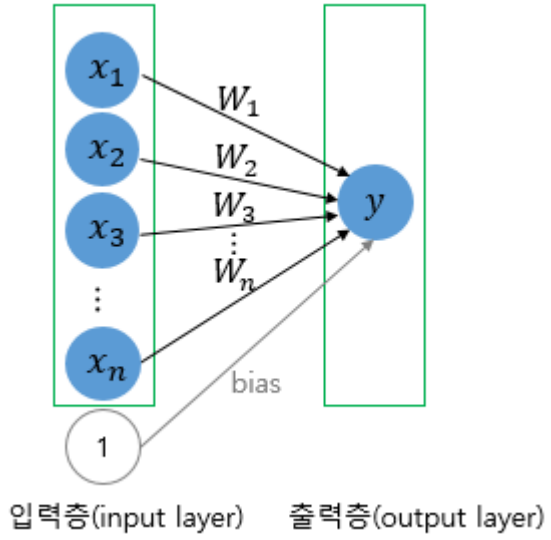
$$if \sum_{i}^{n} W_i x_i + b < 0 \rightarrow y = 0$$

편향이 추가된 임계치값 수식

2. 퍼셉트론(Perceptron)

❖ 단층 퍼셉트론(Single-Layer Perceptron)

- 단층 퍼셉트론과 다층 퍼셉트론으로 분류
- 단층 퍼셉트론 : 값을 보내는 단계와 값을 받아서 출력하는 두 단계로만 구성
- 이때 이 각 단계를 보통 층(layer)라고 부르며, 이 두 개의 층을 입력 층(input layer)과 출력 층(output layer)이라고 함

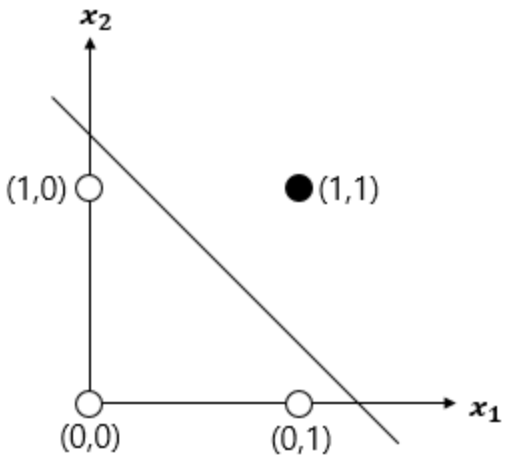


2. 퍼셉트론(Perceptron)

❖ 단층 퍼셉트론(Single-Layer Perceptron)에서 논리 게이트 (AND)

- 단층 퍼셉트론의 AND 게이트를 만족하는 가중치와 편향 값[0.5, 0.5, -0.7], [0.5, 0.5, -0.8] 또는 [1.0, 1.0, -1.0]

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1




```
import numpy as np
def AND_gate(x):
    w1=0.5
    w2=0.5
    b=-0.7
    result = x[0]*w1 + x[1]*w2 + b
    if result <= 0:
        return 0
    else:
        return 1
```

✓ 0.3s

```
input_data=np.array([[0,0],[0,1],[1,0],[1,1]])
result=[]
for x in input_data:
    result.append(AND_gate(x))
print(result)
```

✓ 0.2s

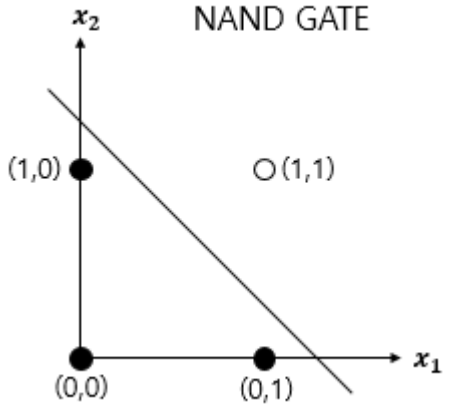
```
[0, 0, 0, 1]
```

2. 퍼셉트론(Perceptron)

❖ 단층 퍼셉트론(Single-Layer Perceptron)에서 논리 게이트 (NAND)

- AND 게이트의 가중치와 편향 값인 [0.5, 0.5, -0.7]에 -를 붙여서 [-0.5, -0.5, +0.7]을 단층 퍼셉트론의 식에 적용하면 NAND 게이트를 충족

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0




```python
def NAND_gate(x):
    w1=-0.5
    w2=-0.5
    b=0.7
    result = x[0]*w1 + x[1]*w2 + b
    if result <= 0:
        return 0
    else:
        return 1
```

✓ 0.2s

```python
input_data=np.array([[0,0],[0,1],[1,0],[1,1]])
result=[]
for x in input_data:
    result.append(NAND_gate(x))
print(result)
```

✓ 0.3s

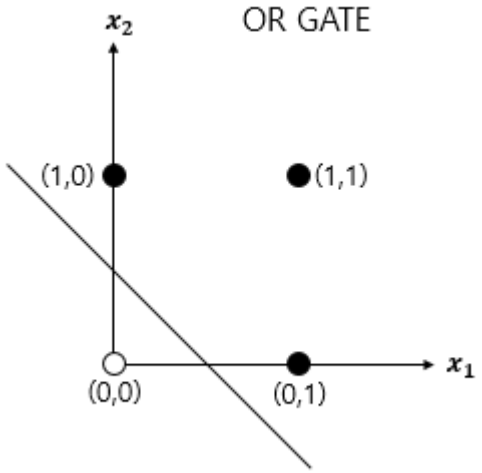
```
[1, 1, 1, 0]
```

2. 퍼셉트론(Perceptron)

❖ 단층 퍼셉트론(Single-Layer Perceptron)에서 논리 게이트 (OR)

- OR 게이트를 충족하는 가중치와 편향 : [0.6, 0.6, -0.5]

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1




```
def OR_gate(x):
    w1=0.6
    w2=0.6
    b=-0.5
    result = x[0]*w1 + x[1]*w2 + b
    if result <= 0:
        return 0
    else:
        return 1
```

✓ 0.3s

```
input_data=np.array([[0,0],[0,1],[1,0],[1,1]])
result=[]
for x in input_data:
    result.append(OR_gate(x))
print(result)
```

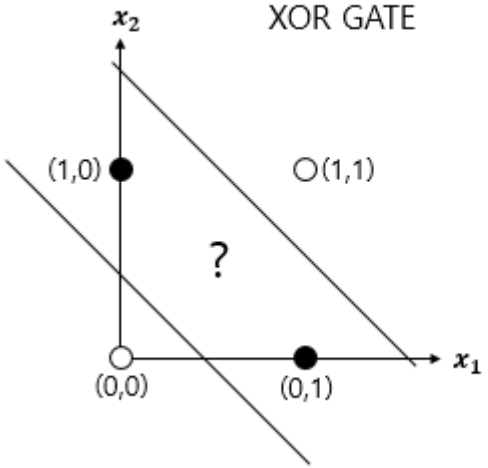
✓ 0.3s

```
[0, 1, 1, 1]
```

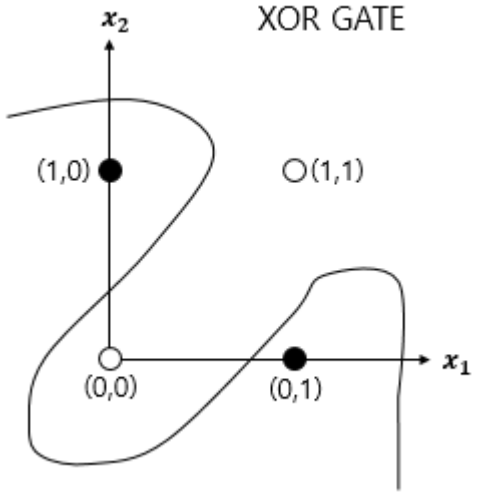
2. 퍼셉트론(Perceptron)

❖ 단층 퍼셉트론(Single-Layer Perceptron)에서 논리 게이트 (XOR)

- XOR 게이트는 입력값 두 개가 서로 다른 값을 갖고 있을 때, 출력값이 1이 되고, 입력값 두 개가 서로 같은 값을 가지면 출력값이 0이 되는 게이트
- 단층 퍼셉트론은 직선 하나로 두 영역을 나눌 수 있는 문제이므로 XOR 게이트 구현할 수 없다.
- 다층 퍼셉트론으로 구현



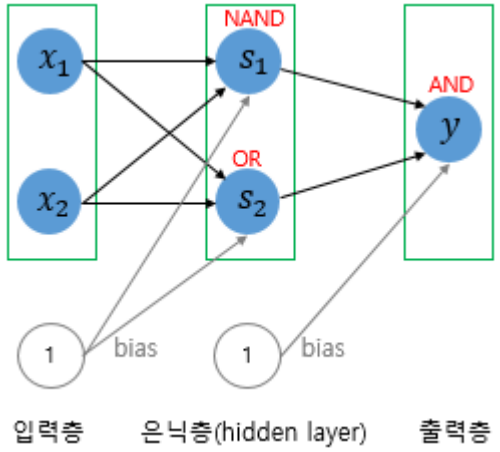
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



2. 퍼셉트론(Perceptron)

❖ 다층 퍼셉트론(MultiLayer Perceptron, MLP)

- XOR 게이트는 기존의 AND, NAND, OR 게이트를 조합하면 만들 수 있음
- 다층 퍼셉트론 : 입력층, 은닉층, 출력층으로 구성



x_1	x_2	s_1 (NAND)	s_2 (OR)	y (XOR)
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

```
input_data=np.array([[0,0],[0,1],[1,0],[1,1]])

s1=[]
s2=[]
new_input_data=[] # AND 입력
final_output=[] #AND 출력 리스트

for i in range(len(input_data)):
    s1=NAND_gate(input_data[i]) # NAND 출력
    s2=OR_gate(input_data[i]) # OR 출력

    new_input_data.append(s1) #AND 입력
    new_input_data.append(s2)

    result=AND_gate(new_input_data) # AND 1개 수행결과 출력
    final_output.append(result)
    new_input_data=[]
print(final_output)
```

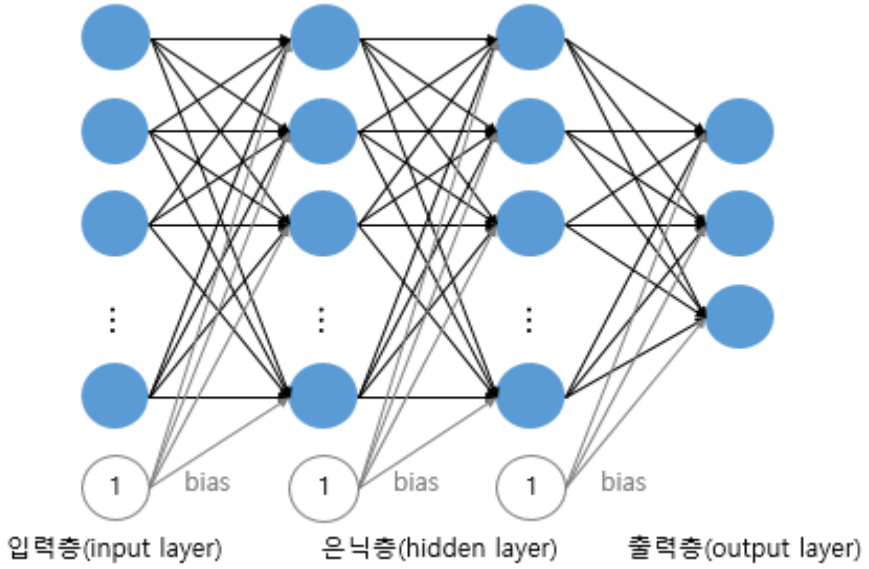
✓ 0.4s

```
[0, 1, 1, 0]
```

2. 퍼셉트론(Perceptron)

❖ 다층 퍼셉트론(MultiLayer Perceptron, MLP)

- 다층 퍼셉트론은 은닉층이 1개 이상인 퍼셉트론
- 즉, XOR 문제보다 더욱 복잡한 문제를 해결하기 위해서 다층 퍼셉트론은 중간에 수많은 은닉층을 더 추가할 수 있음
- **심층 신경망(Deep Neural Network, DNN) :** 은닉층이 2개 이상인 퍼셉트론
- 이 경우에는 은닉층이 2개, 뉴런의 개수를 늘린 다층 퍼셉트론



3. XOR 게이트 - 단층 퍼셉트론 구현하기

- 앞의 OR, AND, XOR 게이트, 가중치 (W), 편향(b) 찾기 문제
  - 퍼셉트론이 정답을 출력할 때까지 가중치를 바꿔보면서 맞는 가중치를 수동으로 찾음
  - 머신러닝 학습을 통해 기계가 가중치를 스스로 찾아내도록 자동화 필요
    - 선형 회귀와 로지스틱 회귀의 **손실 함수(Loss function)**와 **옵티마이저(Optimizer)**를 사용
    - 학습을 시키는 인공 신경망이 심층 신경망일 경우에는 심층 신경망을 학습시킨다고 하여, **딥 러닝(Deep Learning)**이라고 함.

3. XOR 게이트 - 단층 퍼셉트론 구현하기

❖ 단층 퍼셉트론

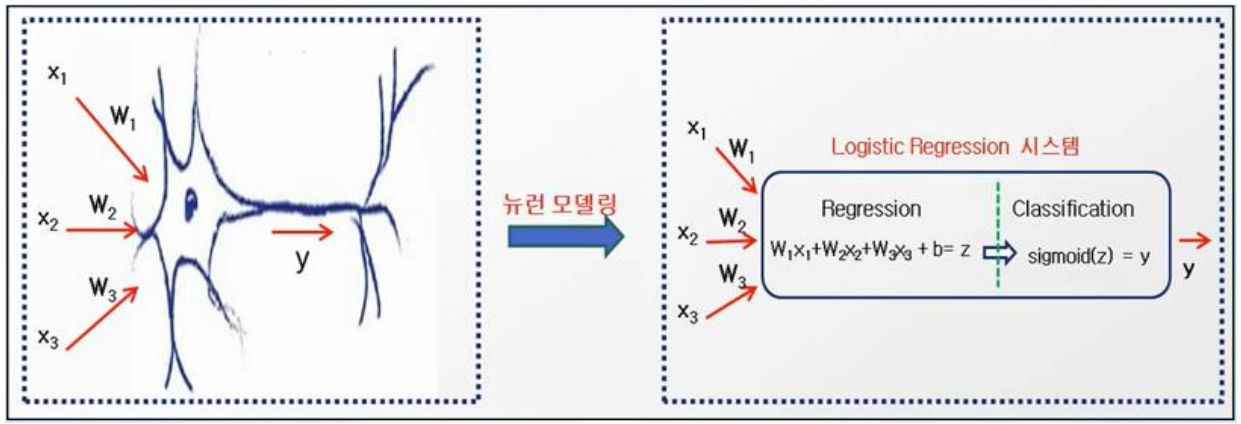
```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = torch.FloatTensor([[0], [1], [1], [0]])

linear = nn.Linear(2, 1, bias=True)
sigmoid = nn.Sigmoid()
model = nn.Sequential(linear, sigmoid)

criterion = torch.nn.BCELoss() # 이진 분류에 사용하는 비용함수인 크로스엔트로피 함수
optimizer = torch.optim.SGD(model.parameters(), lr=1)
```

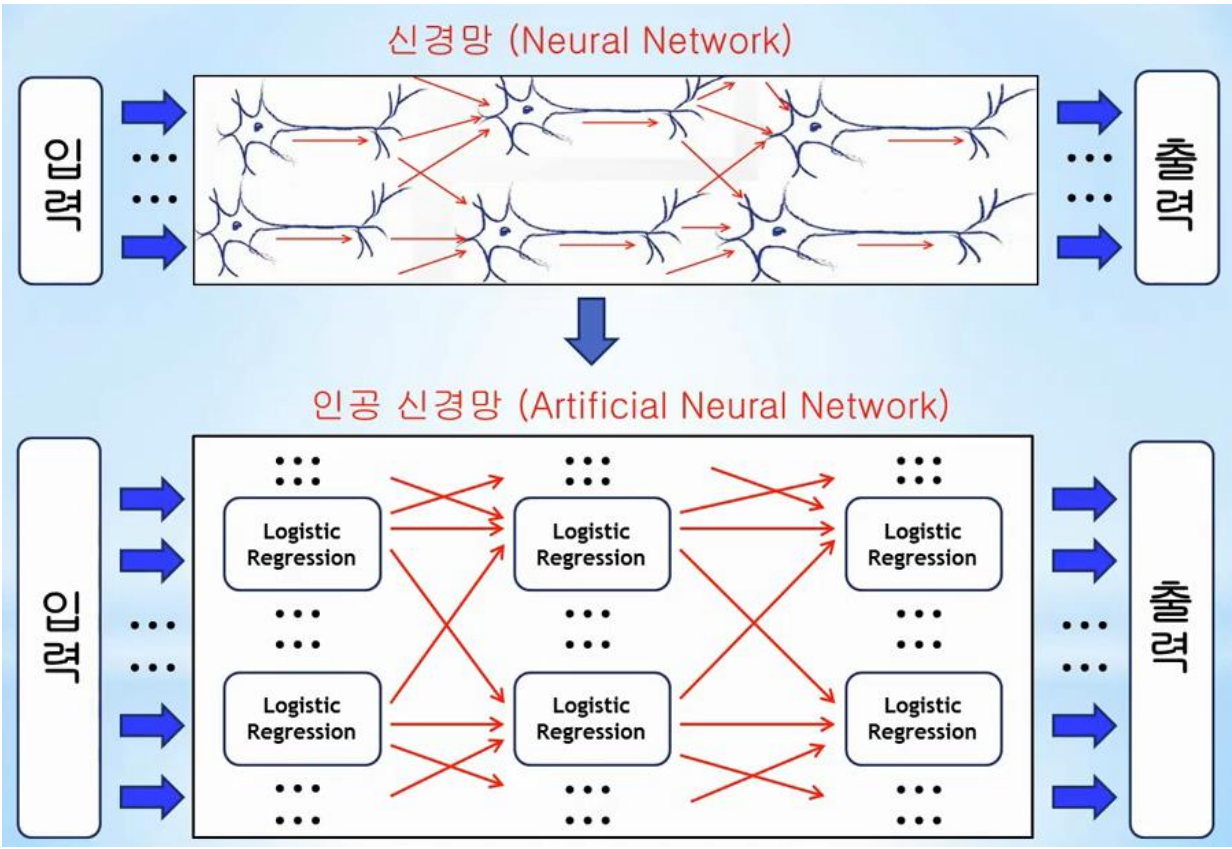
```
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)

    # 비용 함수
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100 == 0: # 100번째 에포크마다 비용 출력
        print(step, cost.item())
```

```
prediction=model(X)  # 학습모델에 대한 예측
print((prediction>0.5).float()) # 0.5보다 큰 값은 1, 그렇지 않은 값은 0으로 표시
```

4. 인공 신경망

❖ 인공 신경망의 이해(Neural Network Overview)

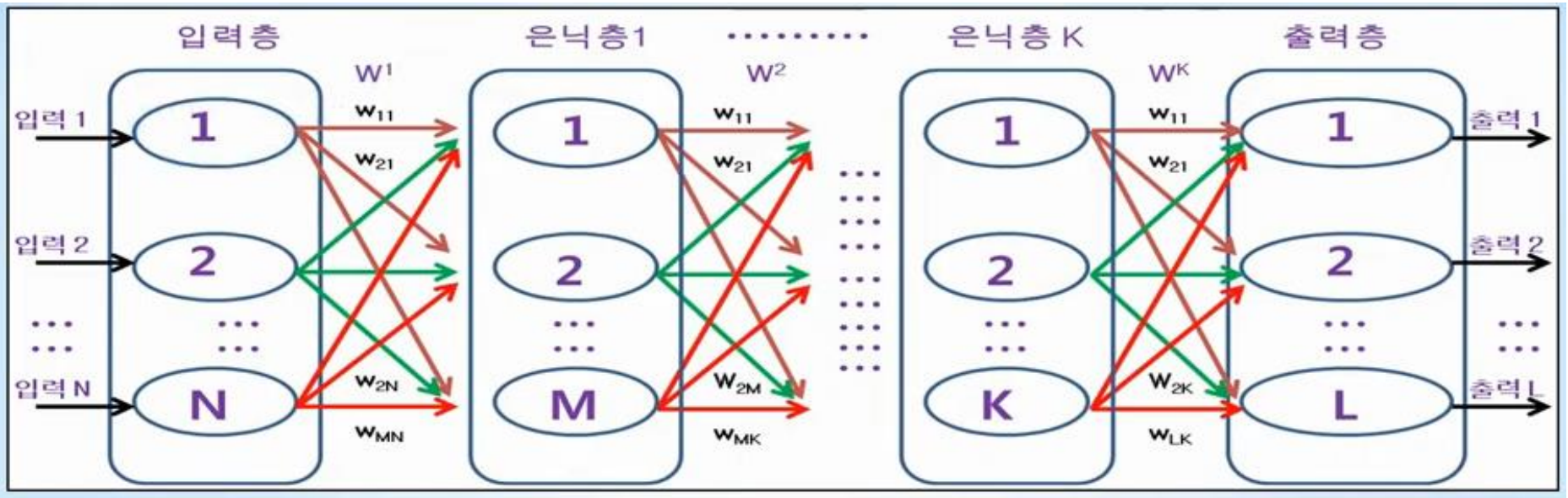




- 신경 세포인 뉴런 동작 원리를 머신러닝에 적용하기 위해서는,
  - 입력 신호와 가중치를 곱하고 적당한 바이어스를 더한 후 Linear Regression) 수행
  - 그 값을 활성화 함수(simoid) 입력으로 전달(Classification)해서 sigmod 함수 임계점 0.5를 넘어면 1, 그렇지 않으면 0을 다음 뉴런에 전달

Multi-variable Logsitic Regression 시스템 구축

4. 인공 신경망



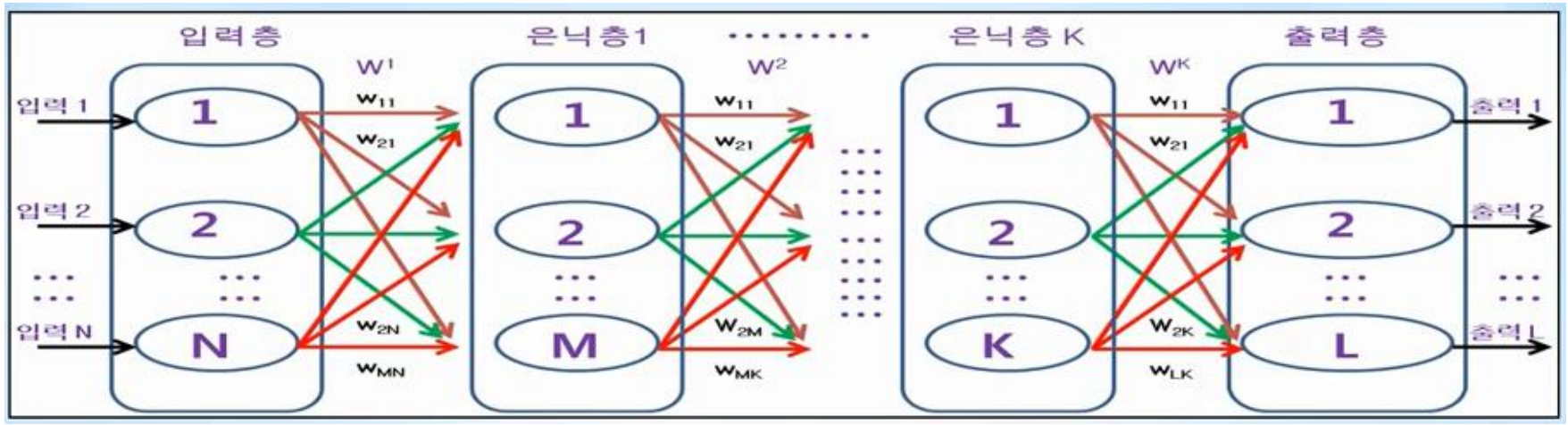
➢ 딥러닝 (Deep Learning)

노드(node) : 1개의 logistic regression을 나타냄

- 노드가 서로 연결되어 있는 신경망 구조를 바탕으로 입력층(Input Layer), 1개 이상의 은닉층(Hidden Layer), 출력층(Output Layer)을 구축하고, 출력층(Output Layer)에서의 오차를 기반으로 각 노드(뉴런)의 가중치(Weight)를 학습하는 머신러닝 한 분야

- [참고] 딥러닝 구조에서 1개 이상의 은닉층(hidden layer)을 이용하여 학습시키면 정확도가 높은 결과를 얻을 수 있다고 알려져 있음. 즉 은닉층을 깊게(Deep) 할수록 정확도가 높아진다고 해서 딥(Deep)러닝이라는 용어가 사용 되고 있음

4. 인공 신경망

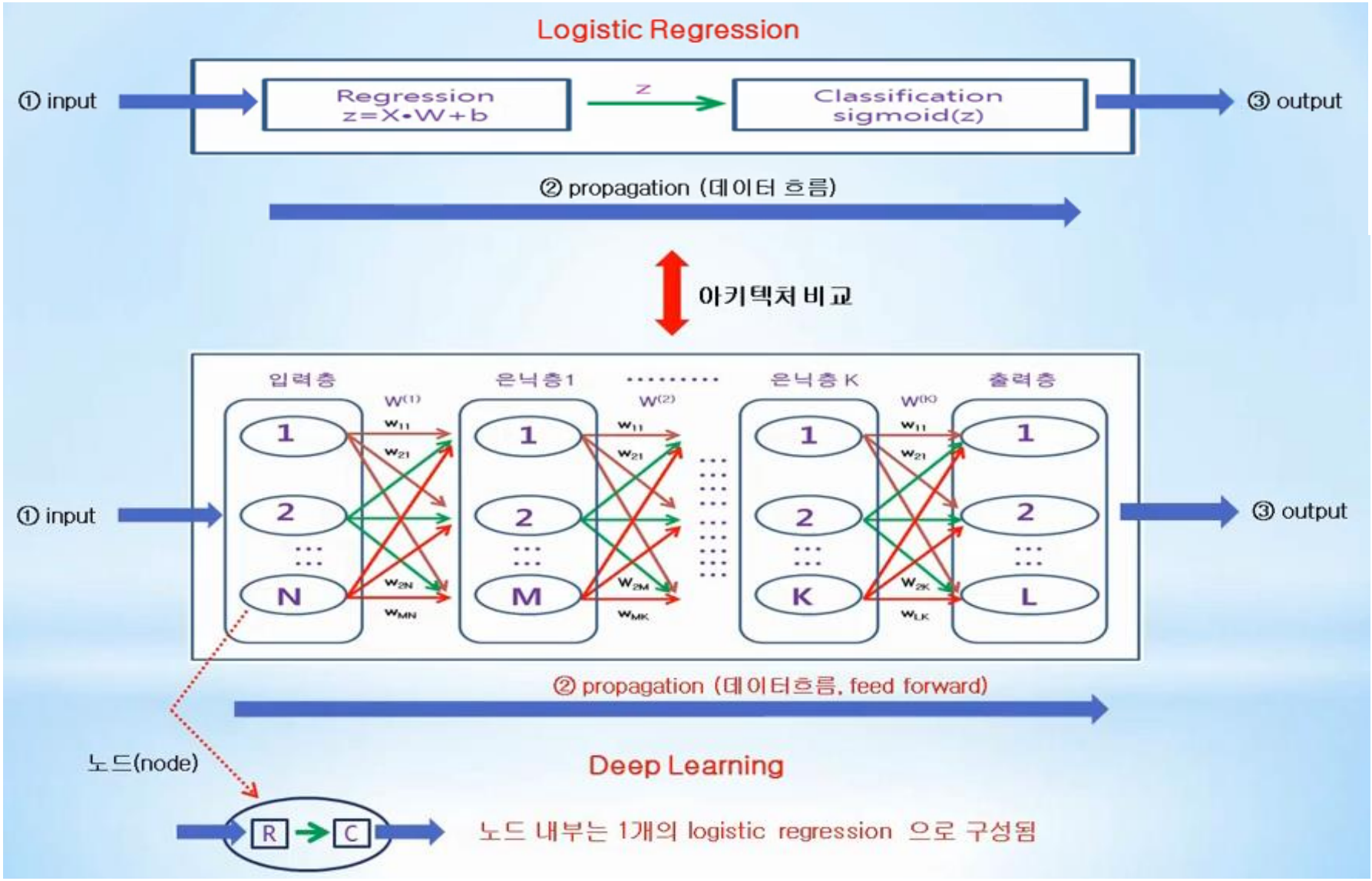


가중치 w_{21} ⇒ 특정 계층의 노드 1에서 다음 계층의 노드 2로 전달되는 신호를 강화 또는 약화 시키는 가중치 (즉, 다음계층의 노드 번호가 먼저 나옴)

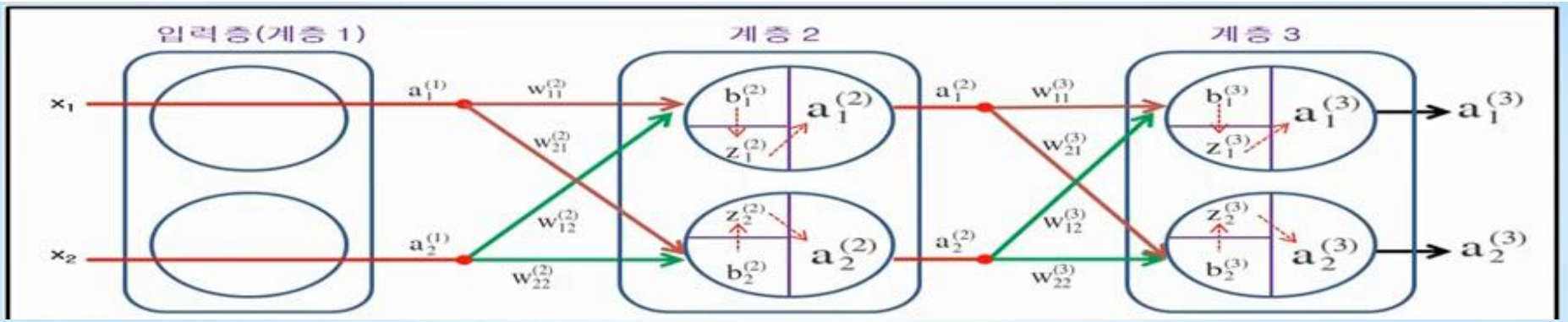
가중치 w_{2N} ⇒ 특정 계층의 노드 N에서 다음 계층의 노드 2로 전달되는 신호를 강화 또는 약화 시키는 가중치 (즉, 다음계층의 노드 번호가 먼저 나옴)

이러한 가중치 값들은, 층과 층사이의 모든 노드에 초기화 되어 있으며, 데이터가 입력층에서 출력층으로 전파(propagation) 될 때, 각 층에 있는 노드의 모든 가중치(w_{11}, w_{12}, … w_{MK}, w_{LK} 등)는 신호를 약화시키거나(낮은 가중치) 또는 신호를 강화(높은 가중치) 시키며, 최종적으로는 오차가 최소 값이 될 때 최적의 값을 가지게 됨

4. 인공 신경망


Logistic Regression
① input
Regression
z=X•W+b
z
Classification
sigmoid(z)
③ output
② propagation (데이터 흐름)
아키텍처 비교
입력층
은닉층1
은닉층 K
출력층
① input
③ output
② propagation (데이터흐름, feed forward)
노드(node)
Deep Learning
노드 내부는 1개의 logistic regression 으로 구성됨

4. 인공 신경망



➢ 계층간 가중치 표기법 (weight notation)

- 가중치 $w_{21}^{(2)}$ ⇒ 계층 2의 노드에 적용되는 가중치로서, 1 계층의 노드 1에서 2계층의 노드 2로 전달되는 신호를 강화 또는 약화시키는 가중치 (즉, 가중치에서의 아래숫자는 다음계층의 노드 번호가 먼저 나옴)

➢ 노드의 바이어스 표기법 (bias notation)

- 바이어스 $b_1^{(2)}$ ⇒ 계층 2에 있는 첫번째 노드 (node 1) 에 적용되는 바이어스

➢ 노드의 선형회귀 계산 값 표기법 (linear regression notation)

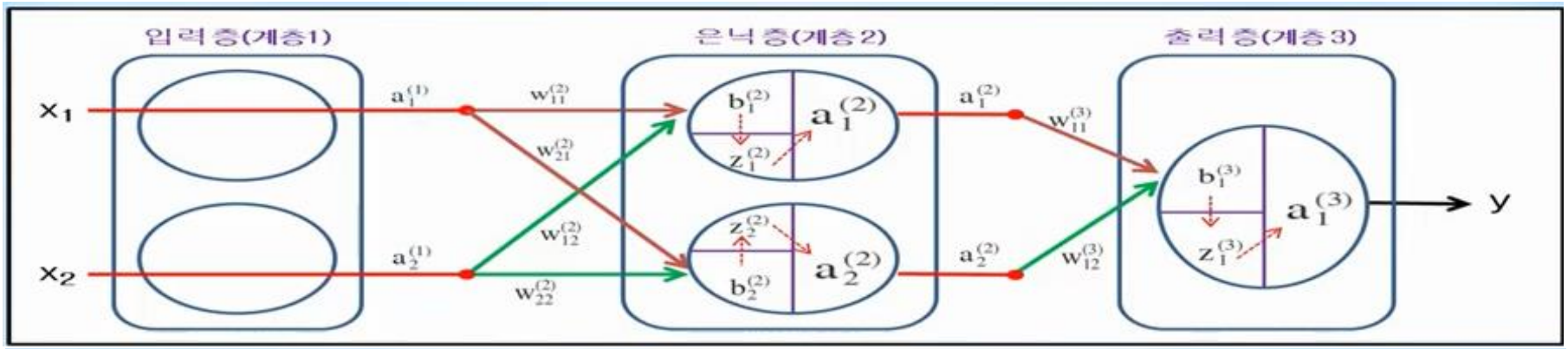
- 선형회귀 계산 값 $z_2^{(2)}$ ⇒ 계층 2의 두번째 노드 (node 2) 선형회귀 계산 값 ($z_2^{(2)} = x_1 w_{21}^{(2)} + x_2 w_{22}^{(2)} + b_2^{(2)}$)

➢ 노드의 출력 표기법 (node output notation)

- 노드의 출력 값 $a_2^{(2)}$ ⇒ 계층 2의 두번째 노드 (node 2) 출력값으로서, logistic regression 계산 값.

  활성화함수(activation function)로서 sigmoid를 사용한다면 $a_2^{(2)} = \mathrm{sigmoid}(z_2^{(2)})$

4. 인공 신경망



➢ 피드포워드 (feed forward)

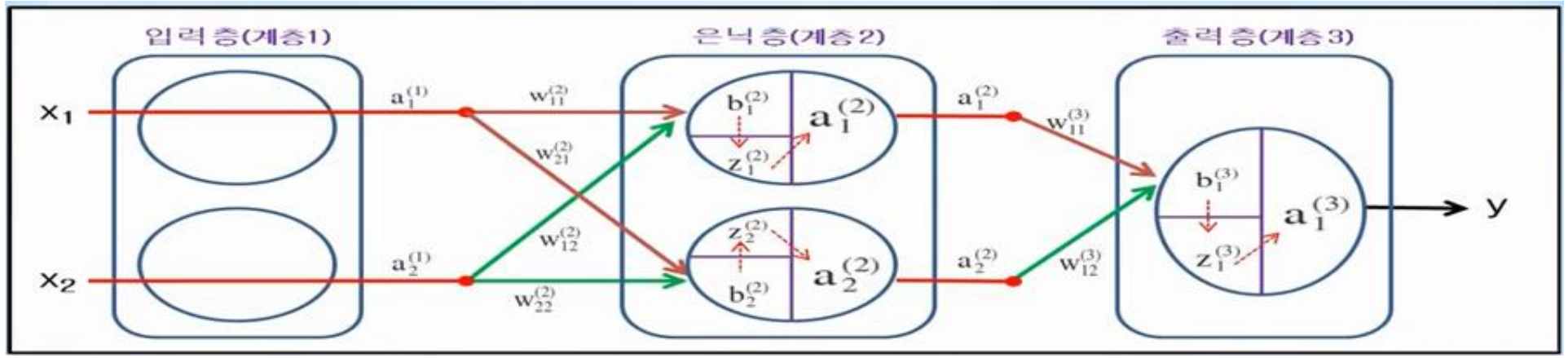
- 입력 층(input layer)으로 데이터가 입력되고, 1개 이상으로 구성되는 은닉 층(hidden layer)을 거쳐서 마지막에 있는 출력 층(output layer)으로 출력 값을 내보내는 과정
- 딥러닝 에서는 이전 층(previous layer)에서 나온 출력 값 ⇒ 층과 층 사이에 적용되는 가중치(weight) 영향을 받은 다음 ⇒ 다음 층(next layer)의 입력 값 으로 들어 가는 것을 의미함

➢ 입력 층(input layer) 출력

- 딥러닝 입력 층에서는 활성화 함수인 sigmoid 를 적용하지 않고, 입력 값 그대로 출력으로 내보내는 것이 관례화 되어 있음.

일반 식	행렬 식
$a_1^{(1)} = x_1 \quad a_2^{(1)} = x_2$	$\begin{pmatrix} a_1^{(1)} & a_2^{(1)} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$

4. 인공 신경망



➤ 은닉 층(hidden layer) 선형회귀 값

입력층과 은닉층 사이의 가중치

일반 식	행렬 식
$z_1^{(2)} = a_1^{(1)}w_{11}^{(2)} + a_2^{(1)}w_{12}^{(2)} + b_1^{(2)}$ $z_2^{(2)} = a_1^{(1)}w_{21}^{(2)} + a_2^{(1)}w_{22}^{(2)} + b_2^{(2)}$	$\begin{pmatrix} z_1^{(2)} & z_2^{(2)} \end{pmatrix} = \begin{pmatrix} a_1^{(1)} & a_2^{(1)} \end{pmatrix} \bullet \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{pmatrix} + \begin{pmatrix} b_1^{(2)} & b_2^{(2)} \end{pmatrix}$

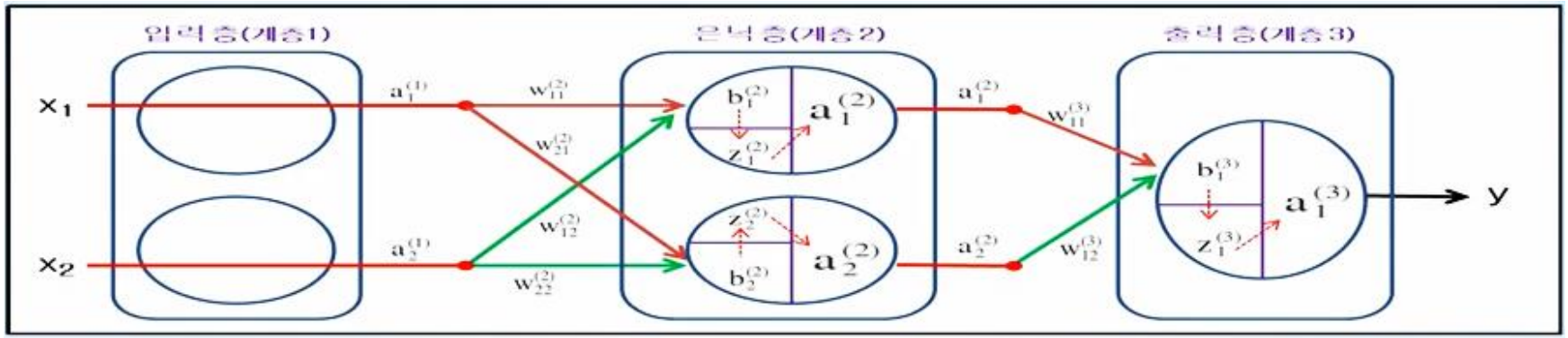
은닉 층 선형회귀 값 / 입력 층 출력 / 은닉 층 바이어스

➤ 은닉 층(hidden layer) 출력

$$a_1^{(2)} = \text{sigmoid}(z_1^{(2)}) \qquad a_2^{(2)} = \text{sigmoid}(z_2^{(2)})$$

- 활성화함수(activation function)는 sigmoid 이므로 출력 값의 범위는 0~1 사이의 값만을 가짐

4. 인공 신경망



➢ 출력 층(output layer) 선형회귀 값

은닉층과 출력층 사이의 가중치

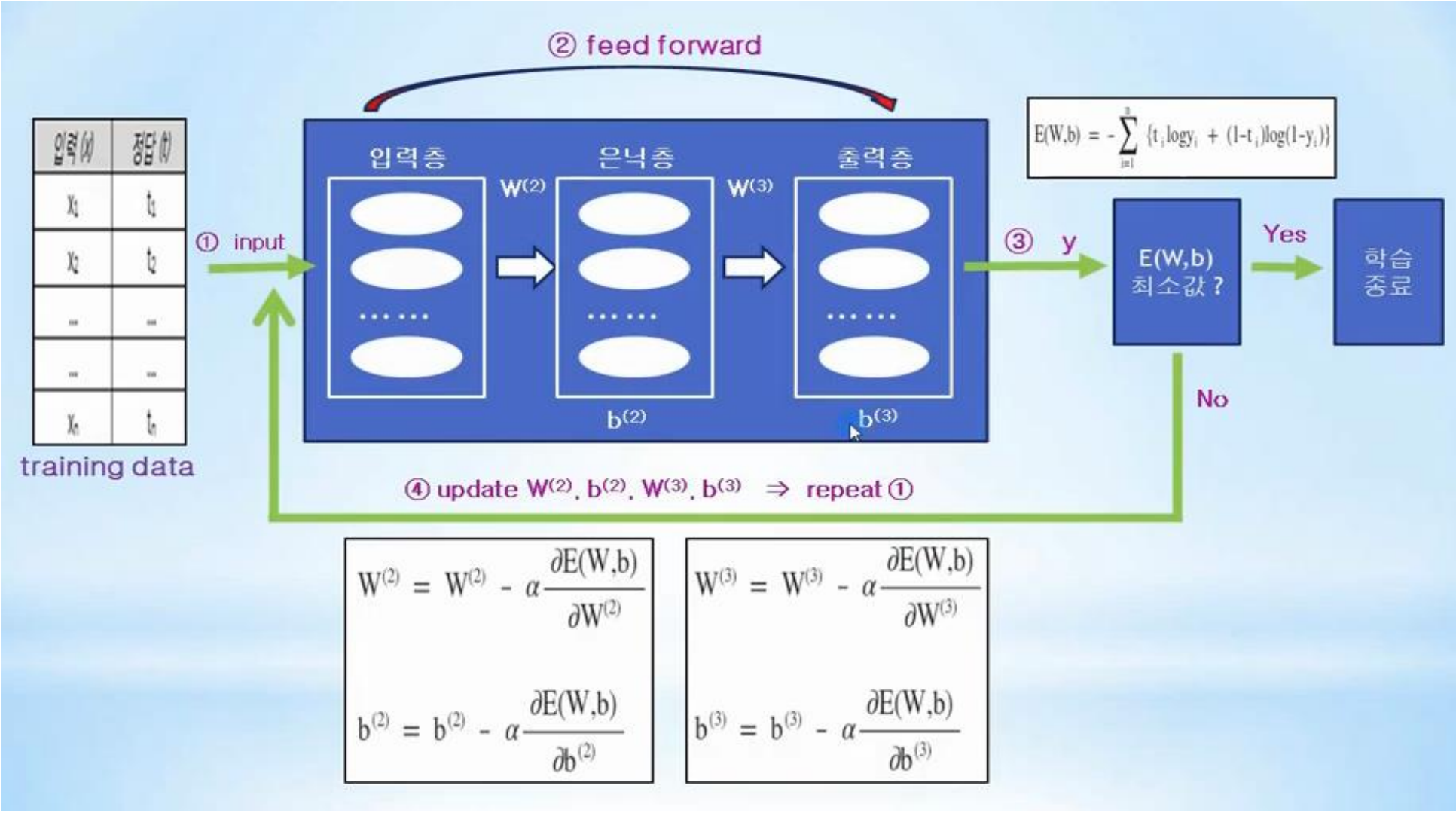
일반 식	행렬 식
$z_1^{(3)} = a_1^{(2)}w_{11}^{(3)} + a_2^{(2)}w_{12}^{(3)} + b_1^{(3)}$	출력 층 선형회귀 값 / 은닉 층 출력 / 출력 층 바이어스: $(z_1^{(3)}) = (a_1^{(2)} \; a_2^{(2)}) \cdot \begin{pmatrix} w_{11}^{(3)} \\ w_{12}^{(3)} \end{pmatrix} + (b_1^{(3)})$

➢ 출력 층(output layer) 출력

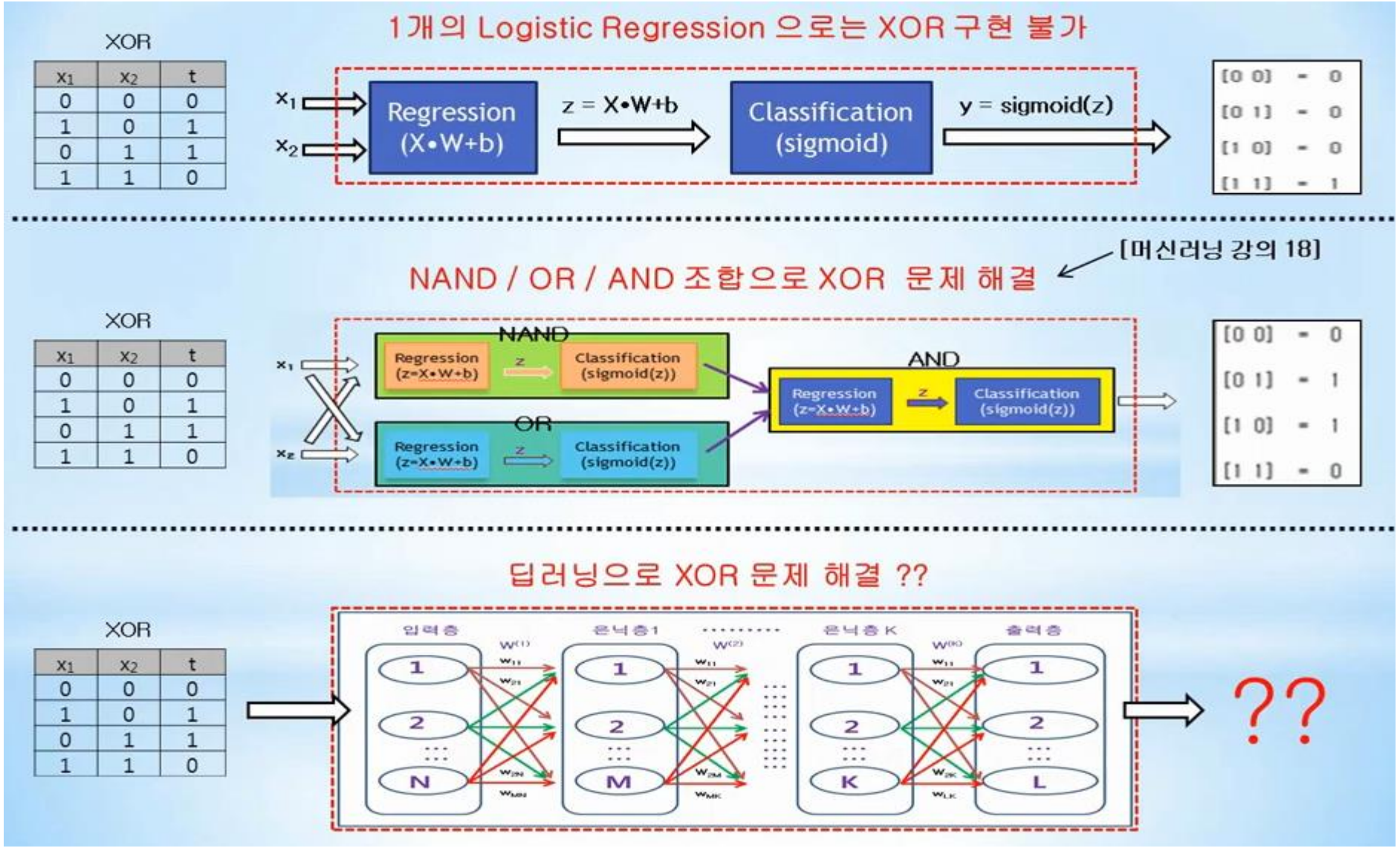
$$y = a_1^{(3)} = \text{sigmoid}(z_1^{(3)})$$

- 출력 값 a1 은, 입력 데이터에 대해 최종적으로 계산해야 하는 y 값 이며, 이러한 y 값과 정답 t 와의 차이인 오차(loss)를 통해서 가중치와 바이어스를 학습해야 하는 것을 의미함
- 즉, 딥러닝 에서는 출력 층(output layer)에서의 출력 값(y)과 정답(t)과의 차이를 이용하여, 오차가 최소가 되도록 각 층에 있는 가중치와 바이어스를 최적화 해야 함

4. 인공 신경망


② feed forward
입력층
은닉층
출력층
W(2)
W(3)
b(2)
b(3)
① input
training data
③ y
E(W,b) 최소값 ?
Yes
학습 중료
No
④ update W(2), b(2), W(3), b(3) ⇒ repeat ①
E(W,b) = - Σ {t_i log y_i + (1-t_i) log(1-y_i)}
W(2) = W(2) - α ∂E(W,b)/∂W(2)
b(2) = b(2) - α ∂E(W,b)/∂b(2)
W(3) = W(3) - α ∂E(W,b)/∂W(3)
b(3) = b(3) - α ∂E(W,b)/∂b(3)

4. 인공 신경망


1개의 Logistic Regression 으로는 XOR 구현 불가
XOR
x1 x2 t
0 0 0
1 0 1
0 1 1
1 1 0
x1
x2
Regression (X•W+b)
z = X•W+b
Classification (sigmoid)
y = sigmoid(z)
[0 0] - 0
[0 1] - 0
[1 0] - 0
[1 1] - 1
[머신러닝 강의 18]
NAND / OR / AND 조합으로 XOR 문제 해결
XOR
x1 x2 t
0 0 0
1 0 1
0 1 1
1 1 0
NAND
Regression (z=X•W+b)
z
Classification (sigmoid(z))
OR
Regression (z=X•W+b)
z
Classification (sigmoid(z))
AND
Regression (z=X•W+b)
z
Classification (sigmoid(z))
[0 0] - 0
[0 1] - 1
[1 0] - 1
[1 1] - 0
딥러닝으로 XOR 문제 해결 ??
XOR
x1 x2 t
0 0 0
1 0 1
0 1 1
1 1 0
입력층
은닉층1
은닉층 K
출력층
1
2
N
M
K
L
??

4. 인공신경망

❖ 다층 퍼셉트론

```
# 훈련데이터
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = torch.FloatTensor([[0], [1], [1], [0]])
```

```
# 모델 생성
model=nn.Sequential(
    nn.Linear(2,10, bias=True),
    nn.Sigmoid(),
    nn.Linear(10,10, bias=True),
    nn.Sigmoid(),
    nn.Linear(10,10, bias=True),
    nn.Sigmoid(),
    nn.Linear(10,1, bias=True),
    nn.Sigmoid()
)
```

```
# 비용함수와 옵티마이즈
criterion=torch.nn.BCELoss()
optimizer=torch.optim.SGD(model.parameters(),lr=1)
```

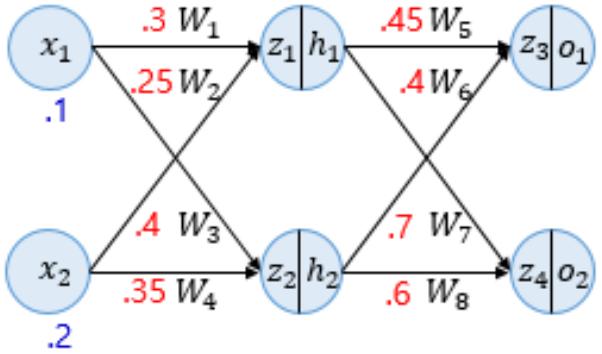
```
# 모델 학습
for epoch in range(10001):
    optimizer.zero_grad()
    y_hat=model(X)
    cost=criterion(y_hat, Y)
    cost.backward()
    optimizer.step()

    if epoch%100==0:
        print(epoch, cost.item())
```

```
# 모델을 사용하여 값 예측
prediction=model(X)
print((prediction>0.5).float())
```

5. 역전파(BackPropagation)

❖ 순전파(Forward Propagation)

- 입력층 -> 은닉층 -> 출력층 진행



$$z_1 = W_1x_1 + W_2x_2 = 0.3\times0.1 + 0.25\times0.2 = 0.08 \qquad h_1 = sigmoid(z_1) = 0.51998934$$

$$z_2 = W_3x_1 + W_4x_2 = 0.4\times0.1 + 0.35\times0.2 = 0.11 \qquad h_2 = sigmoid(z_2) = 0.52747230$$

$$z_3 = W_5h_1 + W_6h_2 = 0.45\times h_1 + 0.4\times h_2 = 0.44498412 \qquad o_1 = sigmoid(z_3) = 0.60944600$$

$$z_4 = W_7h_1 + W_8h_2 = 0.7\times h_1 + 0.6\times h_2 = 0.68047592 \qquad o_2 = sigmoid(z_4) = 0.66384491$$

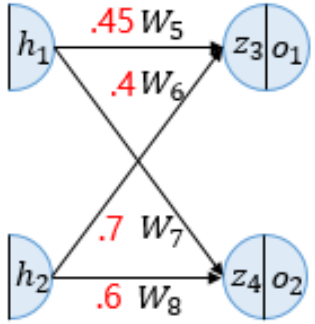
$$E_{o1} = \frac{1}{2}(target_{o1} - output_{o1})^2 = 0.02193381$$

$$E_{o2} = \frac{1}{2}(target_{o2} - output_{o2})^2 = 0.00203809$$

$$E_{total} = E_{o1} + E_{o2} = 0.02397190$$

5. 역전파(BackPropagation)

❖ 역전파 1단계(BackPropagation Step 1)



실제값 : 0.4

실제값 : 0.6

- 순전파 : 입력층에서 출력층으로 향하면서 계산이 이루어짐
- 역전파 : 출력층에서 입력층 방향으로 계산하면서 가중치를 업데이트
- 출력층 바로 이전의 은닉층을 N층이라고 하였을 때, 출력층과 N층 사이의 가중치를 업데이트하는 단계를 역전파 1단계하고 함.
- 역전파 1단계에서 업데이트 해야 할 가중치는 W_5,W_6,W_7,W_8 총 4개
  - W5 업데이트를 다음 식(미분 연쇄법칙)식으로 구함 : $\frac{\partial E_{total}}{\partial W_5} = \frac{\partial E_{total}}{\partial o_1} \times \frac{\partial o_1}{\partial z_3} \times \frac{\partial z_3}{\partial W_5}$
  - 첫 번째 항 미분 :

$$E_{total} = \frac{1}{2}(target_{o1} - output_{o1})^2 + \frac{1}{2}(target_{o2} - output_{o2})^2$$

Etotal : 순전차 진행에서 계산된 전체 오차 값

$$\frac{\partial E_{total}}{\partial o_1} = 2 \times \frac{1}{2}(target_{o1} - output_{o1})^{2-1} \times (-1) + 0$$

$$\frac{\partial E_{total}}{\partial o_1} = -(target_{o1} - output_{o1}) = -(0.4 - 0.60944600) = 0.20944600$$

5. 역전파(BackPropagation)

❖ 역전파 1단계(BackPropagation Step 1)

$$\frac{\partial E_{total}}{\partial W_5} = \frac{\partial E_{total}}{\partial o_1} \times \frac{\partial o_1}{\partial z_3} \times \frac{\partial z_3}{\partial W_5}$$

- 두번째 항 미분
  - o1이라는 값은 시그모이드 함수의 출력 값
  - 시그모이드 함수의 미분은 $f(x) \times (1 - f(x)$

$$\frac{\partial o_1}{\partial z_3} = o_1 \times (1 - o_1) = 0.60944600(1 - 0.60944600) = 0.23802157$$

- 세번째 항 h1과 동일 $\frac{\partial z_3}{\partial W_5} = h_1 = 0.51998934$
- 우변을 결합한 결과 : $\frac{\partial E_{total}}{\partial W_5} = 0.20944600 \times 0.23802157 \times 0.51998934 = 0.02592286$
- 학습률(learning rate) α는 0.5라고 가정 : $W_5^+ = W_5 - \alpha \frac{\partial E_{total}}{\partial W_5} = 0.45 - 0.5 \times 0.02592286 = 0.43703857$
- 같은 방법으로 W_6^+, W_7^+, W_8^+ 를 계산하면 :

$$\frac{\partial E_{total}}{\partial W_6} = \frac{\partial E_{total}}{\partial o_1} \times \frac{\partial o_1}{\partial z_3} \times \frac{\partial z_3}{\partial W_6} \rightarrow W_6^+ = 0.38685205$$

$$\frac{\partial E_{total}}{\partial W_7} = \frac{\partial E_{total}}{\partial o_2} \times \frac{\partial o_2}{\partial z_4} \times \frac{\partial z_4}{\partial W_7} \rightarrow W_7^+ = 0.69629578$$

$$\frac{\partial E_{total}}{\partial W_8} = \frac{\partial E_{total}}{\partial o_2} \times \frac{\partial o_2}{\partial z_4} \times \frac{\partial z_4}{\partial W_8} \rightarrow W_8^+ = 0.59624247$$

5. 역전파(BackPropagation)

❖ 역전파 2단계(BackPropagation Step 2)

- 가중치 W_1, W_2, W_3, W_4 업데이트
- W_1 업데이트 식 : $\frac{\partial E_{total}}{\partial W_1} = \frac{\partial E_{total}}{\partial h_1} \times \frac{\partial h_1}{\partial z_1} \times \frac{\partial z_1}{\partial W_1}$
  - 첫번째 항 식 :

$$\frac{\partial E_{total}}{\partial h_1} = \frac{\partial E_{o1}}{\partial h_1} + \frac{\partial E_{o2}}{\partial h_1}$$

$$\frac{\partial E_{o1}}{\partial h_1} = \frac{\partial E_{o1}}{\partial z_3} \times \frac{\partial z_3}{\partial h_1} = \frac{\partial E_{o1}}{\partial o_1} \times \frac{\partial o_1}{\partial z_3} \times \frac{\partial z_3}{\partial h_1}$$

$$= -(target_{o1} - output_{o1}) \times o_1 \times (1 - o_1) \times W_5$$

$$= 0.20944600 \times 0.23802157 \times 0.45 = 0.02243370$$

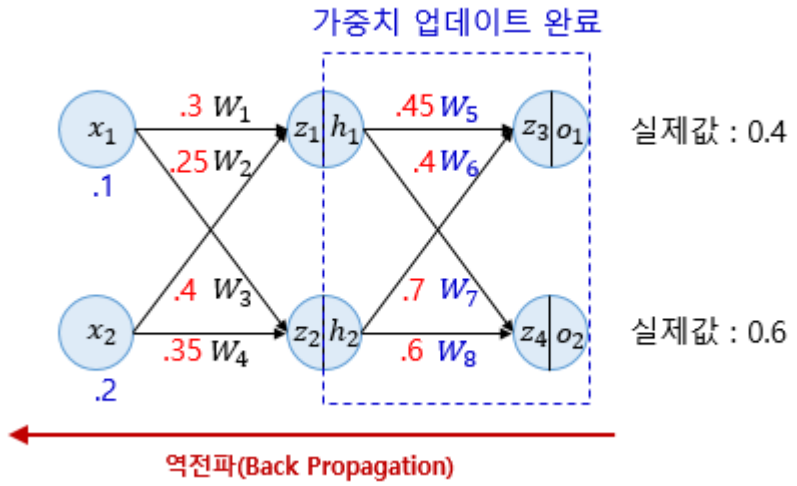
+

$$\frac{\partial E_{o2}}{\partial h_1} = \frac{\partial E_{o2}}{\partial z_4} \times \frac{\partial z_4}{\partial h_1} = \frac{\partial E_{o2}}{\partial o_2} \times \frac{\partial o_2}{\partial z_4} \times \frac{\partial z_4}{\partial h_1} = 0.00997311$$

$$\frac{\partial E_{total}}{\partial h_1} = 0.02243370 + 0.00997311 = 0.03240681$$

  - 두번째 항과 세번째 항 식 : $\frac{\partial h_1}{\partial z_1} = h_1 \times (1 - h_1) = 0.51998934(1 - 0.51998934) = 0.24960043$

$$\frac{\partial z_1}{\partial W_1} = x_1 = 0.1$$



5. 역전파(BackPropagation)

❖ 역전파 2단계(BackPropagation Step 2)

- 첫번째 최종 미분 결과 : $\frac{\partial E_{total}}{\partial W_1} = 0.03240681 \times 0.24960043 \times 0.1 = 0.00080888$
- 학습률이 0.5일때 업데이터 식: $W_1^+ = W_1 - \alpha \frac{\partial E_{total}}{\partial W_1} = 0.1 - 0.5 \times 0.00080888 = 0.29959556$
- 같은 방법으로 W_2^+, W_3^+, W_4^+를 구하면 :

$$\frac{\partial E_{total}}{\partial W_2} = \frac{\partial E_{total}}{\partial h_1} \times \frac{\partial h_1}{\partial z_1} \times \frac{\partial z_1}{\partial W_2} \rightarrow W_2^+ = 0.24919112$$

$$\frac{\partial E_{total}}{\partial W_3} = \frac{\partial E_{total}}{\partial h_2} \times \frac{\partial h_2}{\partial z_2} \times \frac{\partial z_2}{\partial W_3} \rightarrow W_3^+ = 0.39964496$$

$$\frac{\partial E_{total}}{\partial W_4} = \frac{\partial E_{total}}{\partial h_2} \times \frac{\partial h_2}{\partial z_2} \times \frac{\partial z_2}{\partial W_4} \rightarrow W_4^+ = 0.34928991$$

5. 역전파(BackPropagation)

❖ 결과 확인

- 업데이트 된 가중치에 대해서 다시 한 번 순전파를 진행하여 오차가 감소하였는지 확인

$$z_1 = W_1 x_1 + W_2 x_2 = 0.29959556 \times 0.1 + 0.24919112 \times 0.2 = 0.07979778$$

$$z_2 = W_3 x_1 + W_4 x_2 = 0.39964496 \times 0.1 + 0.34928991 \times 0.2 = 0.10982248$$

$$h_1 = sigmoid(z_1) = 0.51993887$$

$$h_2 = sigmoid(z_2) = 0.52742806$$

$$z_3 = W_5 h_1 + W_6 h_2 = 0.43703857 \times h_1 + 0.38685205 \times h_2 = 0.43126996$$

$$z_4 = W_7 h_1 + W_8 h_2 = 0.69629578 \times h_1 + 0.59624247 \times h_2 = 0.67650625$$

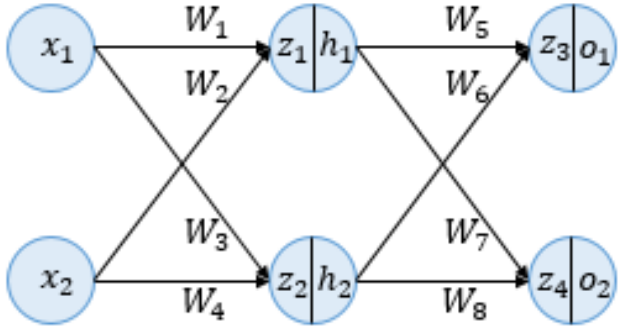
$$o_1 = sigmoid(z_3) = 0.60617688$$

$$o_2 = sigmoid(z_4) = 0.66295848$$

$$E_{o1} = \frac{1}{2}(target_{o1} - output_{o1})^2 = 0.02125445$$

$$E_{o2} = \frac{1}{2}(target_{o2} - output_{o2})^2 = 0.00198189$$

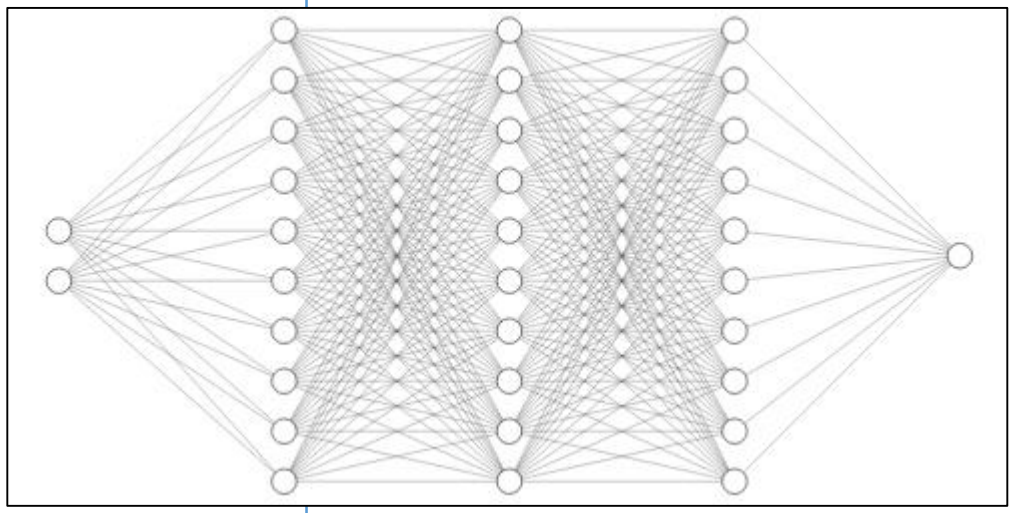
$$E_{total} = E_{o1} + E_{o2} = 0.02323634$$



- 기존의 전체 오차 E_{total}가 0.02397190
- 1번의 역전파로 오차가 감소한 것을 확인 가능
- 인공 신경망의 학습은 오차를 최소화하는 가중치를 찾는 목적으로 순전파와 역전파를 반복하는 것

6. XOR 문제 - 다층 퍼셉트론 구현하기

```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)

#입력층, 은닉층1, 은닉층2, 은닉층3, 출력층을 가지는 은닉층이 3개인 인공 신경망
model = nn.Sequential(
        nn.Linear(2, 10, bias=True), # input_layer = 2, hidden_layer1 = 10
        nn.Sigmoid(),
        nn.Linear(10, 10, bias=True), # hidden_layer1 = 10, hidden_layer2 = 10
        nn.Sigmoid(),
        nn.Linear(10, 10, bias=True), # hidden_layer2 = 10, hidden_layer3 = 10
        nn.Sigmoid(),
        nn.Linear(10, 1, bias=True), # hidden_layer3 = 10, output_layer = 1
        nn.Sigmoid()
        ).to(device)

# 비용함수 : nn.BCELoss()는 이진 분류에서 사용하는 크로스엔트로피 함수
riterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)  # modified learning rate from 0.1 to 1
```



. 비선형 활성화 함수(Activation function)

❖ 비선형 활성화 함수(Activation function)

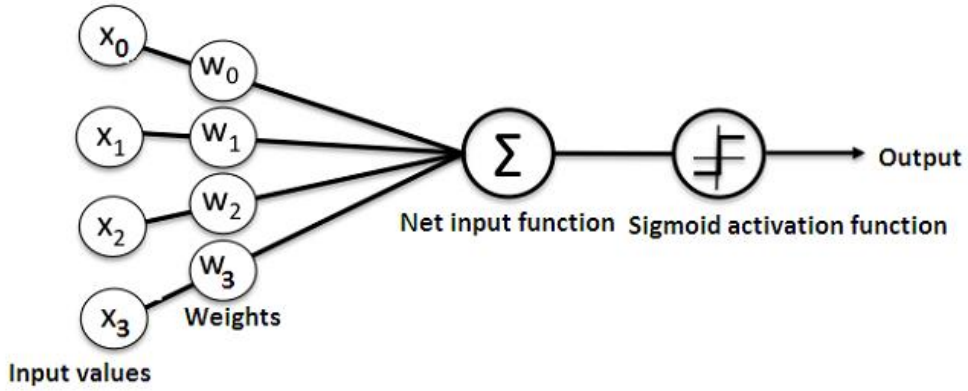
- 입력을 받아 수학적 변환을 수행하고 출력을 생성하는 함수
- 시그모이드, 소프트맥스, 렐루(ReLU) 함수는 대표적인 활성화 함수 중 하나
- 인공 신경망의 은닉층에서 왜 활성화 함수로 시그모이드(sigmoid) 함수를 사용하는 것을 지양

❖ **활성화 함수의 특징 - 비선형 함수(Nonlinear function)**

- 활성화 함수의 특징은 비선형 함수를 사용
- 선형 함수 : 출력이 입력의 상수배만큼 변하는 함수를 선형함수(예 : f(x)=Wx+b)
- 인공 신경망의 능력을 높이기 위해서는 은닉층을 계속해서 추가, 활성화 함수로 선형 함수를 사용하게 되면 은닉층을 쌓을 수가 없음
  - 예 : f(x) = Wx+b 은닉층 2개 추가하면 출력층 포함해서 y(x)=f(f(f(x))) 가 됨
  - $W \times W \times W \times x,\ W^3 = k$라고 하면 , $y(x) = kx$와 같이 표현 가능, 선형함수는 은닉층을 여러 번 추하하여도 1회 추가한 것과 차이를 줄 수 없음

7. 비선형 활성화 함수(Activation function)

❖ 시그모이드 함수(Sigmoid function)와 기울기 소실



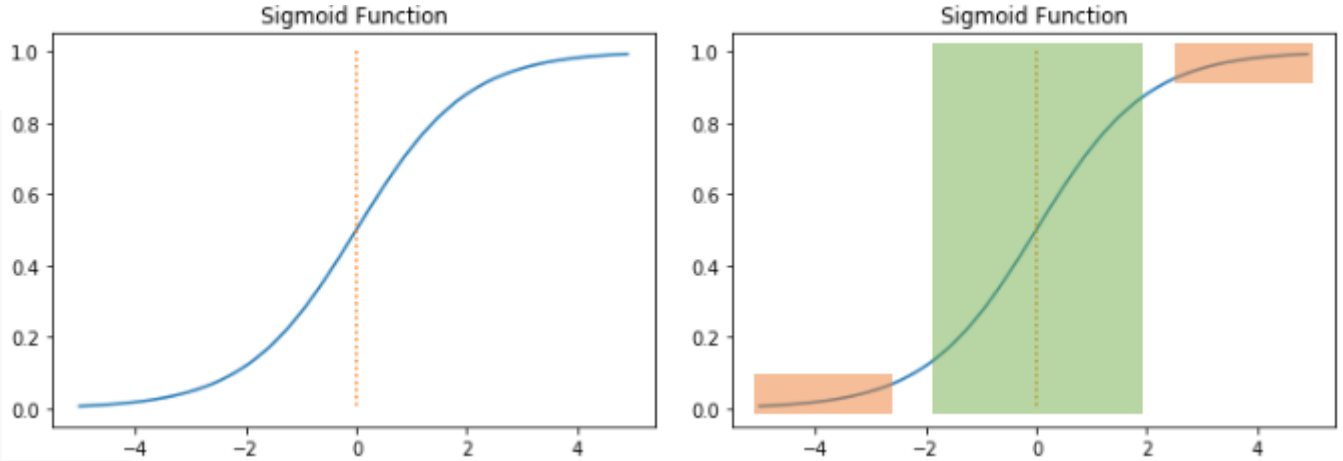
- 인공 신경망의 학습 과정
  - 입력에 대해서 순전파(forward propagation) 연산을 하고,
  - 순전파 연산을 통해 나온 예측값과 실제값의 오차를 손실 함수(loss function)을 통해 계산하고,
  - 이 손실(loss)을 미분을 통해서 기울기(gradient)를 구하고, 이를 통해 역전파(back propagation)를 수행
- 시그모이드 함수의 문제점은 미분을 해서 기울기(gradient)를 구할 때 발생

7. 비선형 활성화 함수(Activation function)

❖ 시그모이드 함수(Sigmoid function)와 기울기 소실

```
import matplotlib.pyplot as plt
# 시그모이드 함수 그래프를 그리는 코드
def sigmoid(x):
    return 1/(1+np.exp(-x))
x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid(x)

plt.plot(x, y)
plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
plt.title('Sigmoid Function')
plt.show()
```

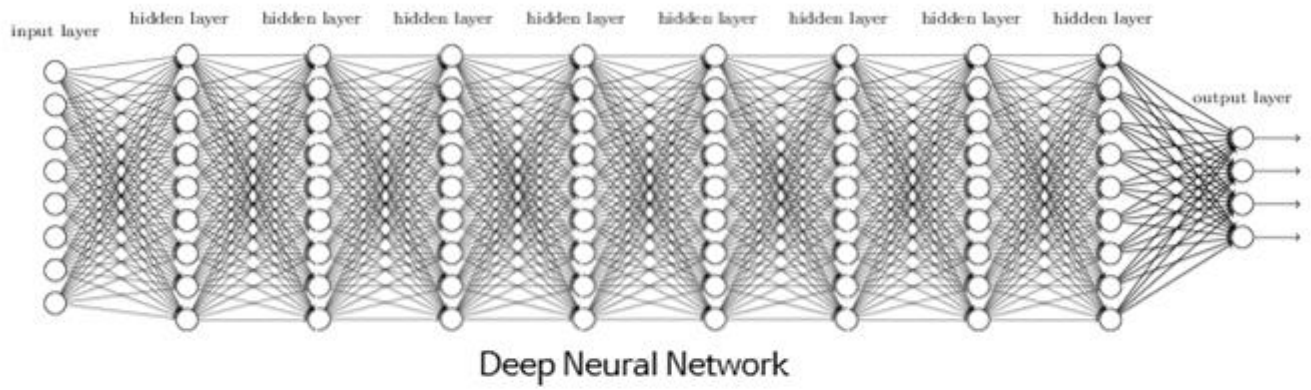
✓ 0.8s



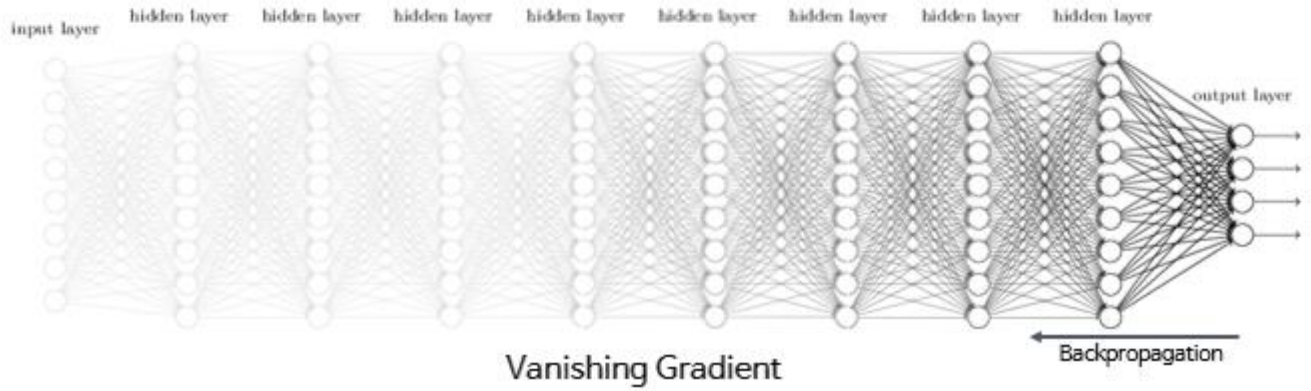
- 시그모이드 함수 그래프 출력 값이 0 또는 1에 가까워 질 수록 기울기가 완만함.
- 역전파 과정에서 0에 가까운 아주 작은 기울기가 곱해지게 되면, 앞 단에는 기울기가 잘 전달되지 않게 됨
- 이러한 현상을 **기울기 소실(Vanishing Gradient) 문제**라고 함

7. 비선형 활성화 함수(Activation function)

❖ 시그모이드 함수(Sigmoid function)와 기울기 소실



- 시그모이드 함수를 사용하는 은닉층의 개수가 다수가 될 경우에는 0에 가까운 기울기가 계속 곱해지면 앞단에서는 거의 기울기를 전파받을 수 없게 됨
- 즉, 매개변수 w가 업데이트 되지 않아 학습이 되지를 않음



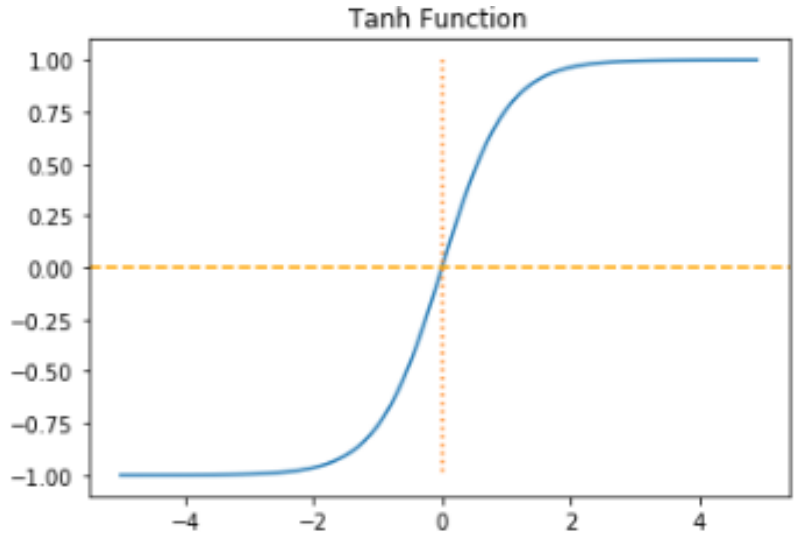
- 기울기 소실 문제로 인해 출력층과 가까운 은닉층에서는 기울기가 잘 전파되지만,
- 앞단으로 갈수록 기울기가 제대로 전파되지 않는 모습을 보여 줌

7. 비선형 활성화 함수(Activation function)

❖ 하이퍼볼릭탄젠트 함수(Hyperbolic tangent function)

```
# 하이퍼복릭 탄젠트 그래프 코드
x = np.arange(-5.0, 5.0, 0.1) # -5.0부터 5.0까지 0.1 간격 생성
y = np.tanh(x)

plt.plot(x, y)
plt.plot([0,0],[1.0,-1.0], ':')
plt.axhline(y=0, color='orange', linestyle='--')
plt.title('Tanh Function')
plt.show()
```

✓ 0.1s



- 하이퍼볼릭탄젠트 함수도 -1과 1에 가까운 출력값을 출력할 때, 시그모이드 함수와 같은 문제가 발생
- 하이퍼볼릭탄젠트 함수의 경우에는 시그모이드 함수와는 달리 0을 중심으로 하고 있는데,
- 이때문에 시그모이드 함수와 비교하면 반환 값의 변화폭이 더 큼.
- 그래서 시그모이드 함수보다는 기울기 소실 증상이 적은 편임, 그래서 은닉층에서 시그모이드 함수보다는 많이 사용됨.

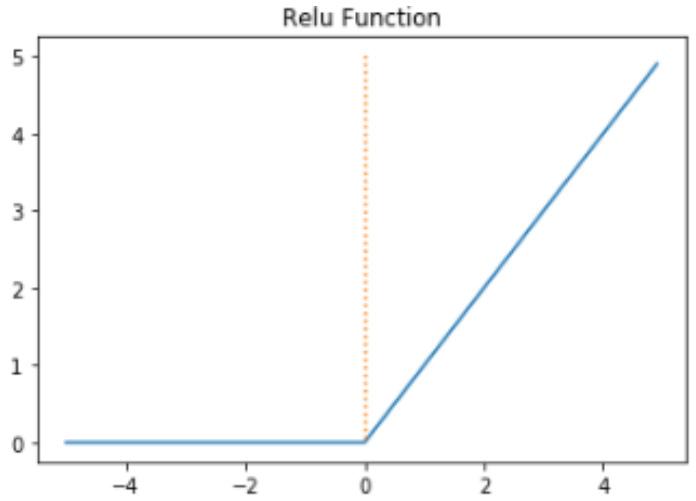
7. 비선형 활성화 함수(Activation function)

❖ 렐루 함수(ReLU)

- 인공 신경망에서 가장 최고의 인기를 얻고 있는 함수
- $f(x) = \max(0, x)$
- 음수를 입력하면 0을 출력하고, 양수를 입력하면 입력값을 그대로 반환
- 특정 양수값에 수렴하지 않으므로 깊은 신경망에서 시그모이드 함수보다 훨씬 더 잘 작동.
- 뿐만 아니라, 렐루 함수는 시그모이드 함수와 하이퍼볼릭탄젠트 함수와 같이 어떤 연산이 필요한 것이 아니라 단순 임계값이므로 연산 속도도 빠름.
- 여전히 문제점이 존재, 입력값이 음수면 기울기도 0이 됨
- 그리고 이 뉴런은 다시 회생하는 것이 매우 어려움
- 이 문제를 죽은 렐루(dying ReLU)라고 함

```
def relu(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)

plt.plot(x, y)
plt.plot([0,0],[5.0,0.0], ':')
plt.title('Relu Function')
plt.show()
```

✓ 0.1s

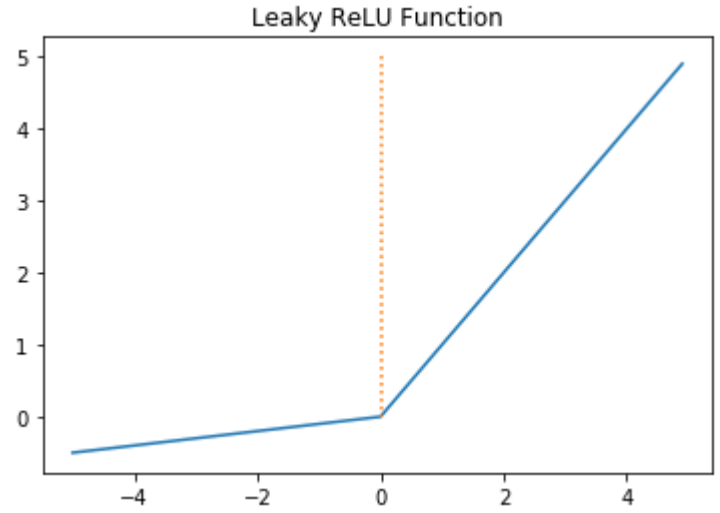


7. 비선형 활성화 함수(Activation function)

❖ 리키 렐루(Leaky ReLU)

- 죽은 렐루를 보완하기 위해 ReLU의 변형 함수들이 등장
- Leaky ReLU는 입력값이 음수일 경우에 0이 아니라 0.001과 같은 매우 작은 수를 반환하도록 되어있음
- $f(x) = \max(ax, x)$
- a는 하이퍼파라미터로 Leaky('새는') 정도를 결정하며 일반적으로는 0.01의 값을 가짐
- 여기서 말하는 '새는 정도'라는 것은 입력 값의 음수일 때의 기울기를 비유하고 있음
- 입력 값이 음수라도 기울기가 0이 되지 않으면 ReLU는 죽지 않음

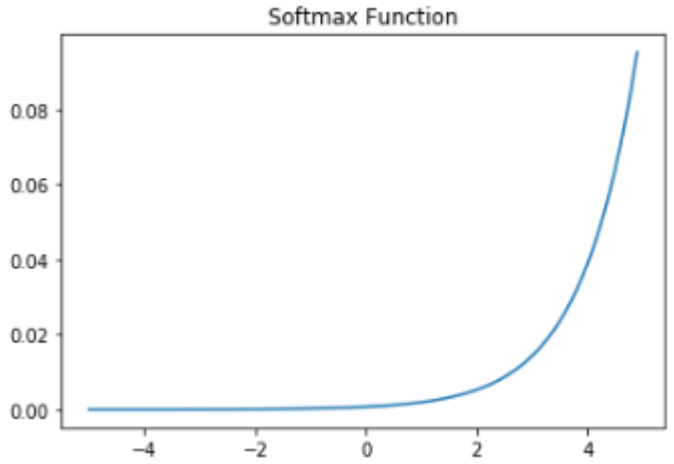
```
# Leaky ReLU)
a = 0.1

def leaky_relu(x):
    return np.maximum(a*x, x)

x = np.arange(-5.0, 5.0, 0.1)
y = leaky_relu(x)

plt.plot(x, y)
plt.plot([0,0],[5.0,0.0], ':')
plt.title('Leaky ReLU Function')
plt.show()
```



7. 비선형 활성화 함수(Activation function)

❖ 소프트 맥스 함수(Softmax function)

```
x = np.arange(-5.0, 5.0, 0.1) # -5.0부터 5.0까지 0.1 간격 생성
y = np.exp(x) / np.sum(np.exp(x))

plt.plot(x, y)
plt.title('Softmax Function')
plt.show()
```



- 은닉층에서 ReLU(또는 ReLU 변형) 함수들을 사용하는 것이 일반적임
- 시그모이드 함수나 소프트맥스 함수는 분류 문제인 로지스틱 회귀와 소프트맥스 회귀를 출력층에 적용함
- 소프트맥스 함수는 시그모이드 함수처럼 출력층의 뉴런에서 주로 사용
- 시그모이드 함수는 두 가지 선택지 중 하나를 고르는 이진 분류 (Binary Classification) 문제에 사용
- 소프트맥스 함수는 세 가지 이상의 (상호 배타적인) 선택지 중 하나를 고르는 다중 클래스 분류(MultiClass Classification) 문제에 주로 사용.

7. 비선형 활성화 함수(Activation function)

❖ 출력층의 활성화 함수와 오차 함수의 관계

문제	활성화 함수	비용 함수
이진 분류	시그모이드	nn.BCELoss()
다중 클래스 분류	소프트맥스	nn.CrossEntropyLoss()
회귀	없음	MSE

다중 퍼셉트론으로 손글씨(MNIST) 실습

다중 퍼셉트론으로 FASHION-MNIST 실습

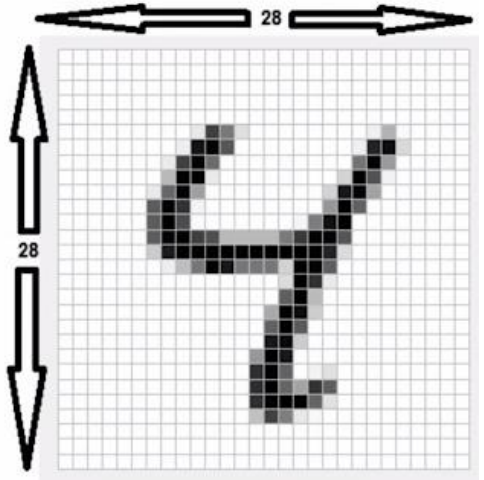
다중 퍼셉트론으로 CIFAR-10 실습

8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

❖ 숫자 필기 데이터

▪ MNIST 데이터 이해하기

- MNIST는 숫자 0부터 9까지의 이미지로 구성된 손글씨 데이터셋으로 과거에 우체국에서 편지의 우편 번호를 인식하기 위해서 만들어진 훈련 데이터.
- 총 60,000개의 훈련 데이터와 레이블, 총 10,000개의 테스트 데이터와 레이블로 구성
- 레이블은 0부터 9까지 총 10개.
- 머신 러닝을 처음 배울 때 접하게 되는 가장 기본적인 샘플
- MNIST 문제는 손글씨로 적힌 숫자 이미지가 들어오면, 그 이미지가 무슨 숫자인지 맞추는 문제
- 예 : 숫자 5의 이미지가 입력으로 들어오면 이게 숫자 5다! 라는 것을 맞춰야 함
- 이 문제는 사람에게는 굉장히 간단하지만 기계에게는 그렇지가 않다.
- MNIST는 각각의 이미지는 오른쪽과 같이 28 픽셀 × 28 픽셀의 이미지로 구성



8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
```

```
# torchvision 에러시
! pip install torchvision
```

```
# matplotlib.pyplot 사용 커널 충돌
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

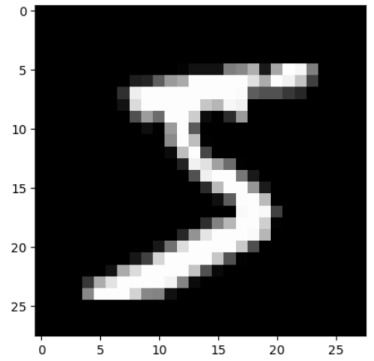
```
mnist_train=datasets.MNIST(root='MNIST_data', train=True, download=True,
                           transform=transforms.Compose([transforms.ToTensor]))

mnist_test=datasets.MNIST('MNIST_data', train=False, download=True,
                           transform=transforms.Compose([transforms.ToTensor]))
```

```
print(mnist_train)
print(mnist_test)
```

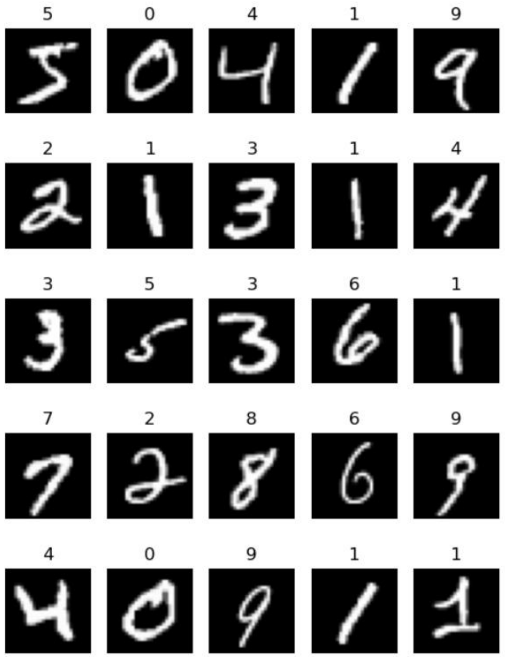
```
print(mnist_train.data.shape)
print(mnist_train.targets.shape)
print(mnist_test.data.shape)
print(mnist_test.targets.shape)
```

8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

```
def plot(x):
    img=(np.array(x, dtype='float')).reshape(28,28)
    plt.imshow(img, cmap='grey')
    plt.show()

plot(mnist_train.data[0])
```



```
def plot(x, y):
    plt.figure(figsize=(6,8))
    for i in range(len(x)):
        plt.subplot(5,5, i+1)
        plt.title(y[i].item())
        img=np.array(x[i]).reshape(28,28)
        plt.imshow(img, cmap='grey')
        plt.axis('off')
    plt.show()

plot(mnist_train.data[:25], mnist_train.targets[:25])
```



8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

```
print(mnist_train.data[1])
print(mnist_train.targets[10])
print(mnist_train.data.shape)
print(mnist_train.targets.shape)
```

```
# 데이터 스케일 조정(0~1 실수로 변경)
x=mnist_train.data.float()/255
print(x[0])
y=mnist_train.targets
y[:10]

print(x.size())
print(y.size())

# 입력데이터 모양 변경(2차원-> 1차원으로 변경)
x=x.view(x.size(0), -1)
input_size=x.size(-1)  # 입력데이터 크기
print(input_size)
output_size=int(max(y))+1 # 출력데이터 크기
print(output_size)
```

```
# 훈련데이터(train)와 검증데이터(valid)로 분리
ratio=[0.8, 0.2]
train_cnt=int(x.size(0)*ratio[0])
valid_cnt=int(x.size(0)*ratio[1])
test_cnt=len(mnist_test.data)

print(train_cnt, valid_cnt, test_cnt)
cnts=[train_cnt, valid_cnt]

indices=torch.randperm(x.size(0))
x=torch.index_select(x, dim=0, index=indices)
y=torch.index_select(y, dim=0, index=indices)

x1=list(x.split(cnts, dim=0))
y1=list(y.split(cnts, dim=0))
print(x1[0].shape, x1[1].shape)
print(y1[0].shape, y1[1].shape)

x1+=[(mnist_test.data.float()/255).view(test_cnt, -1)]
y1+=[mnist_test.targets]

for ii in x1:
    print(ii.shape)
for yi in y1:
    print(yi.shape)
```

8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

```
model=nn.Sequential(
    nn.Linear(input_size, 500),
    nn.LeakyReLU(),
    nn.Linear(500, 400),
    nn.LeakyReLU(),
    nn.Linear(400, 300),
    nn.LeakyReLU(),
    nn.Linear(300, 200),
    nn.LeakyReLU(),
    nn.Linear(200, 100),
    nn.LeakyReLU(),
    nn.Linear(100, 50),
    nn.LeakyReLU(),
    nn.Linear(50, output_size),
    nn.Softmax(dim=-1)
)
model
```

```
crit=nn.CrossEntropyLoss() #
optimizer=optim.Adam(model.parameters())
```

```
device=torch.device('cpu')
if torch.cuda.is_available():
    device=torch.device('cuda')
```

```
model=model.to(device)
x2=[x_i.to(device) for x_i in x1]
y2=[y_i.to(device) for y_i in y1]
```

```
epochs=100
batch_size=256
print_interval=10
```

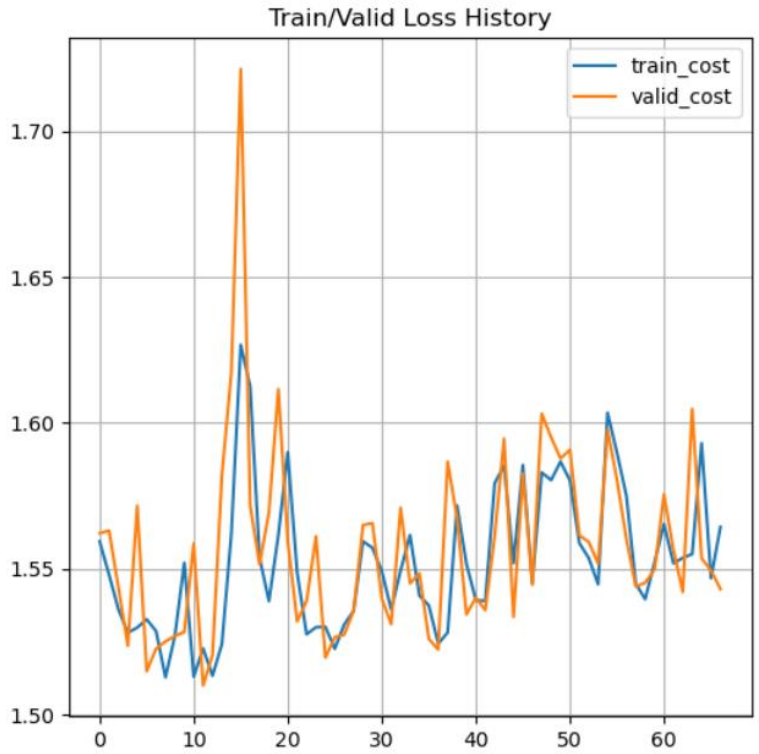
8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

```
train_history, valid_history=[],[]

for i in range(epochs):
    indices=torch.randperm(x2[0].size(0)).to(device)
    x_=torch.index_select(x2[0], dim=0, index=indices)
    y_=torch.index_select(y2[0], dim=0, index=indices)

    x_=x_.split(batch_size, dim=0)
    y_=y_.split(batch_size, dim=0)
    #print(x_[0].shape)
    #print(y_[0].shape)

    train_loss, valid_loss=0,0
    y_hat=[]

    for x_i, y_i in zip(x_, y_):
        y_hat_i=model(x_i)
        loss=crit(y_hat_i, y_i.squeeze())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss+=float(loss)

    # len(x_)=train_loss/256
    train_loss=train_loss/len(x_)
```

```
with torch.no_grad(): #기울기 미분을 하지 않는 다는 뜻
        x_=x2[1].split(batch_size, dim=0)
        y_=y2[1].split(batch_size, dim=0)
        valid_loss=0

        for x_i, y_i in zip(x_, y_):
            y_hat_i=model(x_i)
            loss=crit(y_hat_i, y_i)
            valid_loss+=float(loss)

            y_hat+=[y_hat_i]

    valid_loss=valid_loss/len(x_)

    train_history+=[train_loss]
    valid_history+=[valid_loss]

    if (i+1) % print_interval==0:
        print(i, train_loss, valid_loss, lowest_loss)

    if valid_loss <=lowest_loss:
        lowest_loss=valid_loss
        lowest_epoch=i
        best_model=deepcopy(model.state_dict)
    else:
        if early_stop >0 and lowest_epoch+early_stop < i+1:
            print(f'{lowest_epoch} 이후 {early_stop} epoch 증가하는
동안 손실함수 감소가 없음')
            break

print(f'epoch {lowest_epoch}일때 손실함수 {lowest_loss}가 검증 데이터
로 가장 낮는 손실함수를 가짐')
```

8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

```
plot_from=0
plt.figure(figsize=(6,6))
plt.grid(True)
plt.title('Train/Valid Loss History')

plt.plot(range(plot_from, len(train_history)),
train_history[plot_from:], label='train_cost')

plt.plot(range(plot_from, len(valid_history)),
valid_history[plot_from:], label='valid_cost')

plt.legend()
plt.show()
```



8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류: MNIST 실습

```
# 테스트 데이터로 모델 평가
test_loss=0
y_hat=[]

with torch.no_grad():
    x_=x2[-1].split(batch_size, dim=0)
    y_=y2[-1].split(batch_size, dim=0)

    for x_i, y_i in zip(x_, y_):
        y_hat_i=model(x_i)
        loss=crit(y_hat_i, y_i.squeeze())

        test_loss+=loss
        y_hat+=[y_hat_i]

test_loss=test_loss/len(x_)
y_hat=torch.cat(y_hat,dim=0)
print('test loss:', test_loss)
```

```
# 정확도 평가
correct_cnt=(y2[-1].squeeze()==torch.argmax(y_hat, dim=-
1)).sum()

print(correct_cnt/10000)
```

```
# confusion_matrix 작성

import pandas as pd
from sklearn.metrics import confusion_matrix

df=pd.DataFrame(confusion_matrix(y2[-1],
                                 torch.argmax(y_hat, dim=-1)),
                                 index=['true_%d'%i for i in range(10)],
                                 columns=['pred_%d'%i for i in range(10)])
df
```

8. 다층 퍼셉트론으로 MNIST, Fashion MNIST, cifar-10 분류실습

❖ 딥러닝 분야 이미지 데이터 셋

- 참고 페이지 url: https://developer-together.tistory.com/49
- 대표적으로 CIFAR-10, CIFAR-100, STL-10, MNIST, FASHION-MNIST, SVHN, ImageNet

	해상도	클래스 개수	학습 데이터 수	테스트 데이터 수
CIFAR-10	32 X 32 X 3	10개	50,000개 (클래스당 5,000개)	10,000개 (클래스당 1,000개)
CIFAR-100	32 X 32 X 3	100개	50,000개 (클래스당 500개)	10,000개 (클래스당 100개)
STL-10	96 X 96 X 3	10개	5,000개 (클래스당 500개)	8,000개 (클래스당 800개)
MNIST	28 X 28 X 1	10개	60,000개	10,000개
FASHION-MNIST	28 X 28 X 1	10개	60,000개	10,000개
SVHN	32 X 32 X 3	10개	73,257개	26,032개

9. 과적합(Overfitting)을 막는 방법들

❖ 과적합

- 모델이 과적합되면 훈련 데이터에 대한 정확도는 높을지라도, 새로운 데이터. 즉, 검증 데이터나 테스트 데이터에 대해서는 제대로 동작하지 않음

1. **데이터의 양을 늘리기**
2. **모델의 복잡도 줄이기**
3. **가중치 규제(Regularization) 적용하기**
4. **드롭아웃(Dropout)**

9. 과적합(Overfitting)을 막는 방법들

❖ 데이터의 양 늘리기

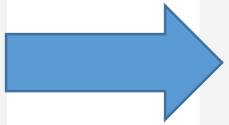
- 모델은 데이터의 양이 적을 경우, 해당 데이터의 특정 패턴이나 노이즈까지 쉽게 암기하기 되므로 과적합 현상이 발생할 확률이 늘어남
- 데이터의 양을 늘릴 수록 모델은 데이터의 일반적인 패턴을 학습하여 과적합을 방지할 수 있음
- 데이터의 양이 적을 경우, 기존의 데이터를 조금씩 변형하고 추가하여 데이터의 양을 늘리기도 하는데 이를 데이터 증식 또는 증강(Data Augmentation)이라고 함
- 이미지의 경우에는 데이터 증식이 많이 사용되는데 이미지를 돌리거나 노이즈를 추가하고, 일부분을 수정하는 등으로 데이터를 증식시킴

9. 과적합(Overfitting)을 막는 방법들

❖ 모델의 복잡도 줄이기

- 인공 신경망의 복잡도는 은닉층(hidden layer)의 수나 매개변수의 수 등으로 결정됨
- 과적합 현상이 포착되었을 때, 인공 신경망 모델에 대해서 할 수 있는 한 가지 조치는 인공 신경망의 복잡도를 줄이는 것

```
class Architecture1(nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(Architecture1, self).__init__()
    self.fc1 = nn.Linear(input_size, hidden_size)
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(hidden_size, hidden_size)
    self.relu = nnReLU()
    self.fc3 = nn.Linear(hidden_size, num_classes)

  def forward(self, x):
    out = self.fc1(x)
    out = self.relu(out)
    out = self.fc2(out)
    out = self.relu(out)
    out = self.fc3(out)
    return out
```



```
class Architecture1(nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(Architecture1, self).__init__()
    self.fc1 = nn.Linear(input_size, hidden_size)
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(hidden_size, num_classes)

  def forward(self, x):
    out = self.fc1(x)
    out = self.relu(out)
    out = self.fc2(out)
    return out
```

9. 과적합(Overfitting)을 막는 방법들

❖ 가중치 규제(Regularization) 적용하기

- 복잡한 모델이 간단한 모델보다 과적합될 가능성이 높음
- 간단한 모델은 적은 수의 매개변수를 가진 모델
- 복잡한 모델을 좀 더 간단하게 하는 방법으로 가중치 규제(Regularizaiton)가 있음

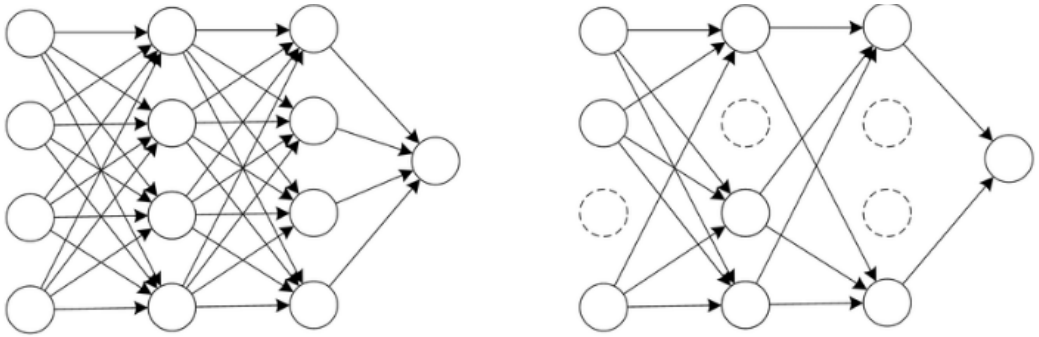
- L1 규제 : 가중치 w들의 절대값 합계를 비용 함수에 추가합니다. L1 규범(norm)이라고도 합니다.
- L2 규제 : 모든 가중치 w들의 제곱합을 비용 함수에 추가합니다. L2 규범(norm)이라고도 합니다.

```
model = Architecture1(10, 20, 2)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-5)
```

9. 과적합(Overfitting)을 막는 방법들

❖ 드롭아웃(Dropout)

- 드롭아웃은 학습 과정에서 신경망의 일부를 사용하지 않는 방법
- 예를 들어 드롭아웃의 비율을 0.5로 한다면 학습 과정마다 랜덤으로 절반의 뉴런을 사용하지 않고, 절반의 뉴런만을 사용



(a) Standard Neural Network

(b) Network after Dropout

- 드롭아웃은 신경망 학습 시에만 사용하고, 예측 시에는 사용하지 않는 것이 일반적임
- 학습 시에 인공 신경망이 특정 뉴런 또는 특정 조합에 너무 의존적이게 되는 것을 방지해주고, 매번 랜덤 선택으로 뉴런들을 사용하지 않으므로 서로 다른 신경망들을 앙상블하여 사용하는 것 같은 효과를 내어 과적합을 방지함