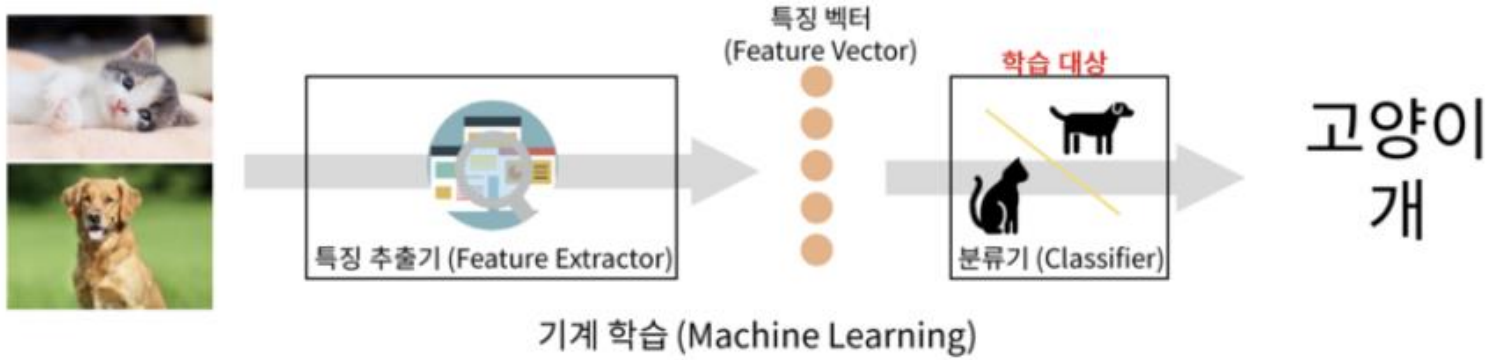
Session 3

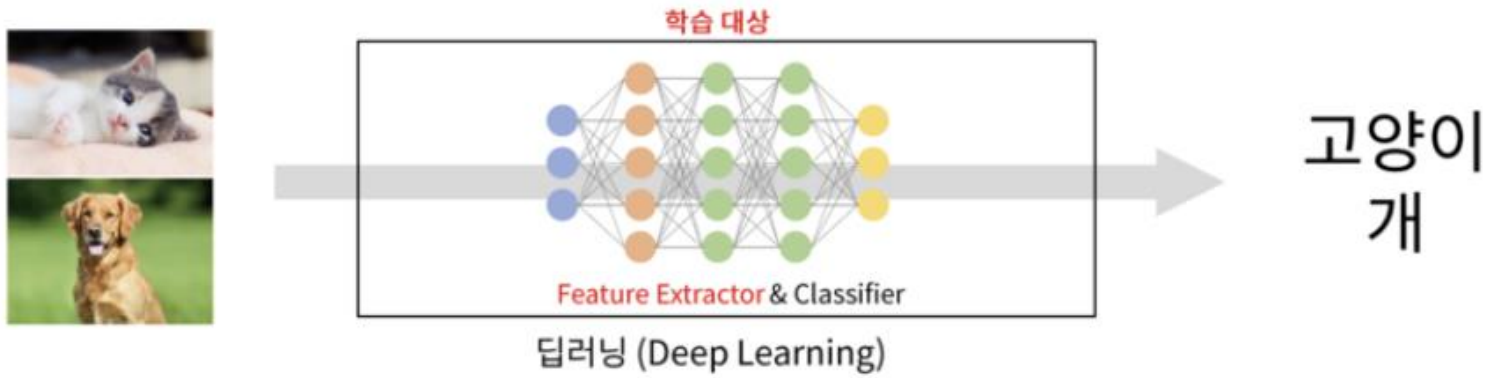
DNN 알고리즘

딥러닝(DNN) 이해

➢ 딥 러닝 vs 머신러닝

- 딥러닝과 머신러닝의 차이점은 학습을 하는 과정이다
- 머신러닝은 특징을 추출하는 부분과 분류를 하는 부분이 별개의 부분으로 동작한다.
- 딥러닝은 인공신경망으로 구성되며, 신경망을 구성할 때 특징 추출에 대한 부분과 분류에 대한 부분이 함께 하나의 모델로 표현되어진다

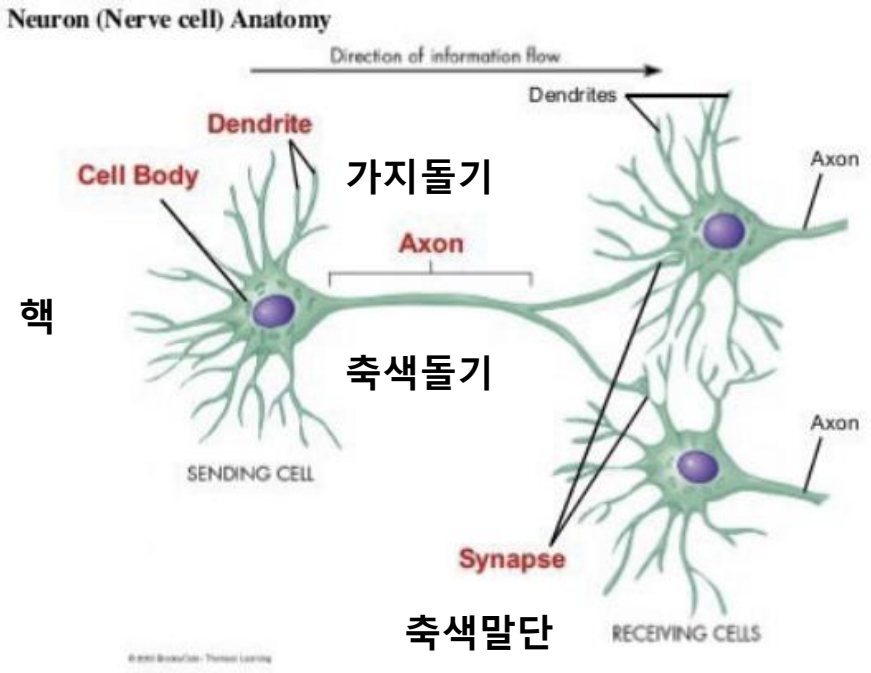




딥러닝(DNN) 이해

➢ 퍼셉트론(Perceptron)

- 인공신경망 뉴런 모델 : 생물학적인 뉴런을 수학적으로 모델링 한 것
- 생물학적인 뉴런 - 다른 여러개의 뉴런으로부터 입력값을 받아서 세포체(cell body)에 저장하다가 자신의 용량을 넘어서면 외부로 출력값을 내보낸다
- 인공신경망 뉴런 - 여러 입력값을 받아서 일정 수준이 넘어서면 활성화되어 하나의 출력값을 내보낸다



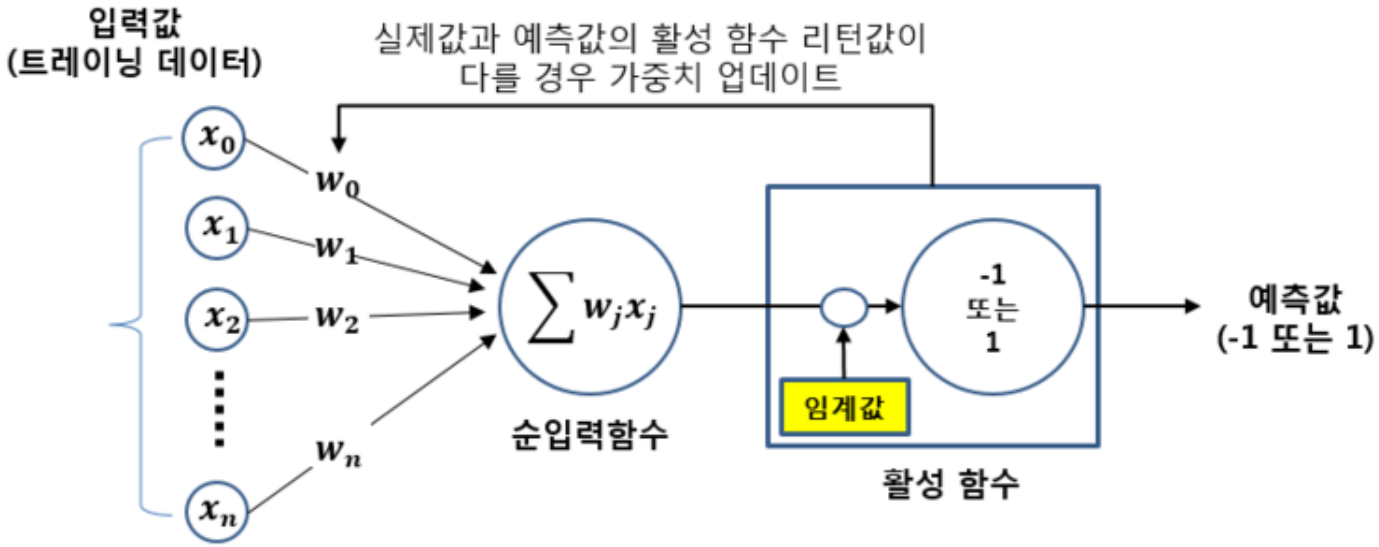
인간의 뇌는 1000억 개가 넘는 신경세포(뉴런)가 100조 개 이상의 시냅스를 통해 병렬적으로 연결되어 있다
각각의 뉴런은 수상돌기(Dendrite)를 통해 다른 뉴런에서 입력 신호를 받아서 축색돌기(Axon)을 통해 다른 뉴런으로 신호를 내보낸다
시냅스(Synapse)는 뉴런과 뉴런을 연결하는 역할을 한다.
출력신호는 입력된 신호가 모여서 일정한 용량을 넘어설 때 일어난다.

딥러닝(DNN) 이해

➢ 퍼셉트론(Perceptron)

- 입력 값과 활성화 함수를 사용해 출력 값을 다음으로 넘기는 가장 작은 신경망 단위
- N개의 이진수가 하나의 뉴런을 통과해서 가중합 0보다 크면 활성화되는 가장 간단한 신경망 구조
- 초평면(hyperplane)으로 구분되는 두 개의 공간을 분리시키는 역할을 한다
- 모든 입력을 출력에 매핑하는 가중치를 학습할 수 있는 능력
- AND 게이트 , OR 게이트를 만들 수 있다.
- 퍼셉트론의 성능을 개선하기 위해서는 퍼셉트론을 정의하는 파라미터, 즉 가중치를 변경해야 한다
- 학습 목표 : 정답을 최대한 많이 맞출 수 있도록 최적의 가중치 조합을 찾는 것

입력이 들어오면, 들어온 입력의 수만큼 가중치를 생성해서 입력과 곱하고, 곱해진 결과들을 모두 더한 다음, 활성함수를 적용하여 출력값을 생성하는 과정으로 동작한다.



딥러닝(DNN) 이해

➢ 활성화 함수(activation function)

- 출력값을 활성화를 일으키게 할 것인가를 정하고 그 값을 부여하는 함수
- 활성화 함수를 사용하면 입력값에 대한 출력값이 linear하게 나오지 않으므로 linear system을 non-linear한 system으로 변환
- 비선형함수

Step Function	• 출력값이 0이 될지, 1이 될지를 결정 • 계단 모양 함수로, 특정값 이하는 0이고 특정값 이상은 1로 출력하도록 만들어진 함수
Sigmoid	• 입력값 x 값이 작아질수록 0에 수렴하고, 커질수록 1에 수렴한다. (Logistic 함수, S자형 함수) • Vanishing Gradient Problem이 발생한다 (은닉층에서의 사용은 지양됩니다)
Hyperbolic tangent function	• 입력값을 -1과 1사이의 값으로 변환합니다 • tanh 함수는 함수의 중심점을 0으로 옮겨 sigmoid가 갖고 있던 최적화 과정에서 느려지는 문제를 해결 • 0을 중심으로 하고있으며 tanh 함수를 미분했을 때의 최대값은 1로 시그모이드 함수의 최대값인 0.25보다는 큽니다. • 시그모이드 함수보다는 기울기 소실 증상이 적은 편이며 은닉층에서 시그모이드 함수보다는 선호됩니다. • tanh 함수도 -1과 1에 가까운 출력값을 출력할 때, 시그모이드 함수와 같은 문제가 발생합니다

딥러닝(DNN) 이해

➢ 활성화 함수(activation function)

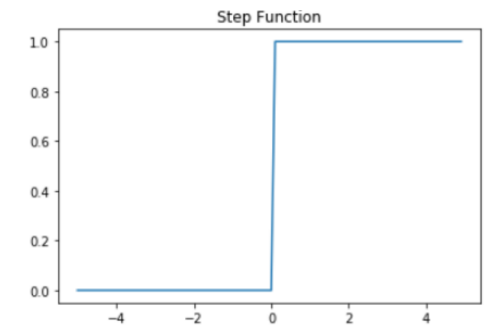
ReLU	• 음수를 입력하면 0을 출력하고, 양수를 입력하면 입력값을 그대로 반환 • x가 0보다 크면 기울기가 1인 직선, 0보다 작으면 함수 값이 0이 된다. • 0이상의 입력값의 경우에는 미분값이 항상 1입니다 • 깊은 신경망의 은닉층에서 시그모이드 함수보다 훨씬 더 잘 작동합니다. • 렐루 함수는 연산 없이 단순 임계값으로 반환하므로 연산 속도도 빠릅니다. • 입력값이 음수면 기울기. 즉, 미분값도 0이 되는 뉴런은 다시 회생하는 것이 매우 어려우며 이 문제를 죽은 렐루(dying ReLU)라고 합니다
Leaky ReLU	• 죽은 렐루를 보완하기 위한 ReLU의 변형 함수 • Leaky ReLU는 입력값이 음수일 경우에 0이 아니라 0.001과 같은 매우 작은 수를 반환하도록 되어있습니다. •입력값이 음수라도 기울기가 0이 되지 않으면 ReLU는 죽지 않습니다.
softmax	• 출력층에서 주로 사용 • 세 가지 이상의 (상호 배타적인) 선택지 중 하나를 고르는 다중 클래스 분류(MultiClass Classification) 문제에 주로 사용됩니다

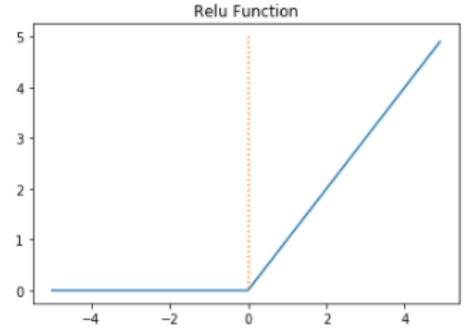
딥러닝(DNN) 이해

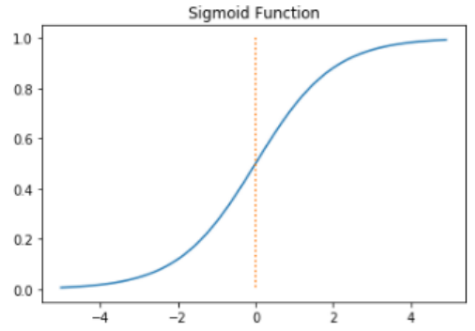
➢ 활성화 함수(activation function)

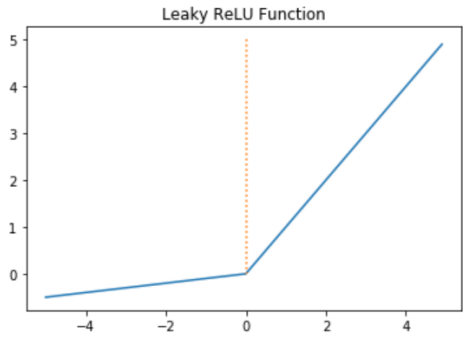
Step Function	def step(x): return np.array(x > 0, dtype=np.int)
Sigmoid	def sigmoid(x): return 1/(1+np.exp(-x))
Hyperbolic tangent function	np.tanh
ReLU	def relu(x): return np.maximum(0, x)
Leaky ReLU	def leaky_relu(x): return np.maximum(a*x, x)
softmax	np.exp(x) / np.sum(np.exp(x))

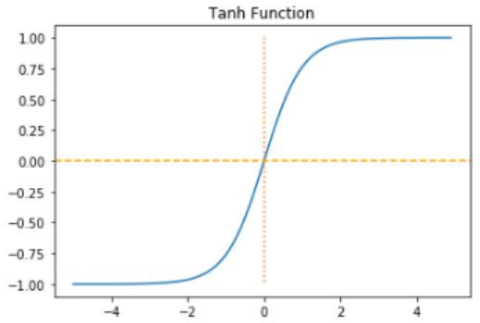
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-5.0, 5.0, 0.1) # -5.0부터 5.0까지 0.1 간격 생성
y = step(x)
plt.title('Step Function')
plt.plot(x,y)
plt.show()
```

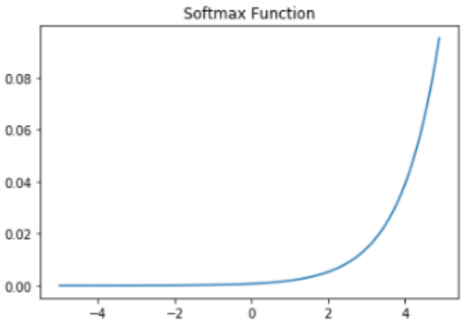








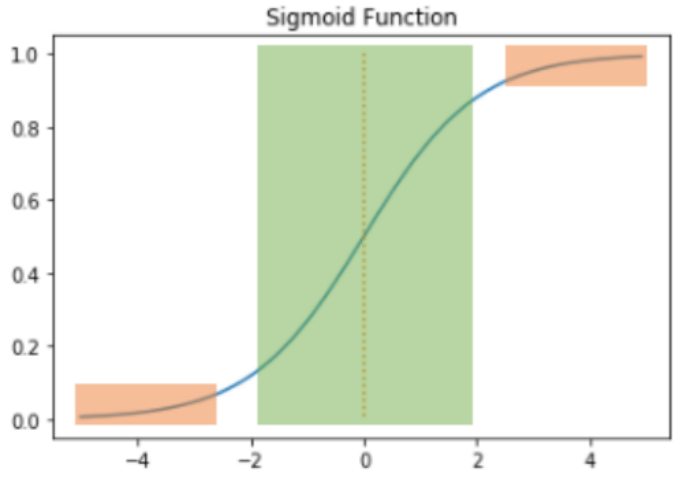


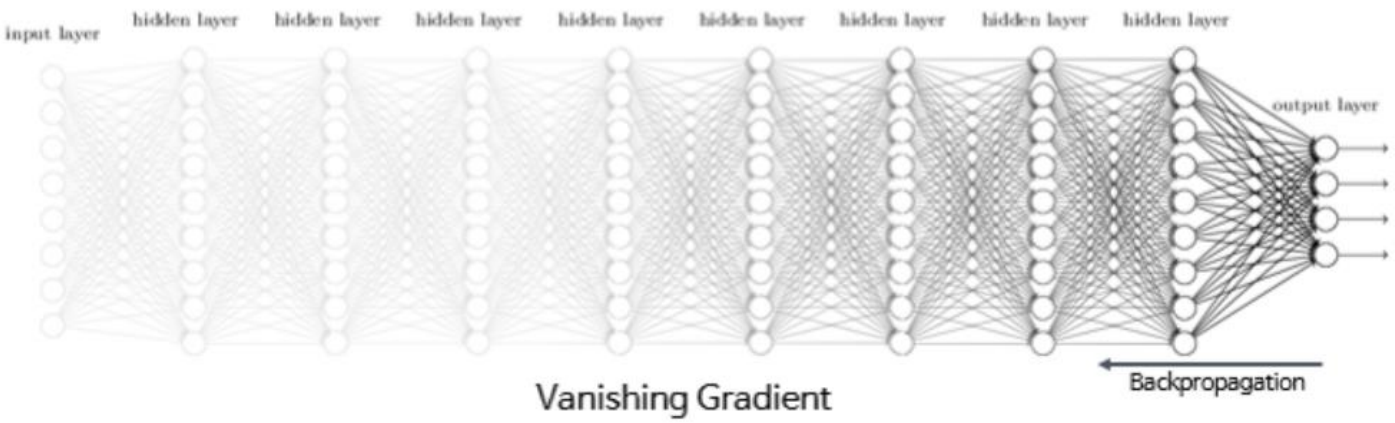


딥러닝(DNN) 이해

➢ 시그모이드 함수(Sigmoid function)와 기울기 소실

- 시그모이드 함수는 한 노드에서 모든 파라미터의 미분 값은 모두 같은 부호를 같게 되는데, 같은 방향으로 update되는 과정은 학습을 지그재그 형태로 만드는 원인을 낳는다.
- 모든 실수 값을 0보다 크고 1보다 작은 미분 가능한 수로 변환하는 특징을 같이 때문에, Logistic Classification과 같은 분류 문제의 가설과 비용 함수(Cost Function)에 많이 사용된다.
- 또한 sigmoid()의 리턴 값이 확률 값이기 때문에 결과를 확률로 해석할 때 유용하다.
- 시그모이드 함수의 출력값이 0 또는 1에 가까워지면, 그래프의 기울기가 완만해지는 모습을 볼 수 있습니다.
- 주황색 구간에서는 미분값이 0에 가까운 아주 작은 값입니다. (초록색 구간에서의 미분값은 최대값이 0.25)
- 시그모이드 함수를 활성화 함수로하는 인공 신경망의 층을 쌓는다면, 가중치와 편향을 업데이트 하는 과정인 역전파 과정에서 0에 가까운 값이 누적해서 곱해지게 되면서, 앞단에는 기울기(미분값)가 잘 전달되지 않게 됩니다. => **기울기 소실(Vanishing Gradient) 문제**

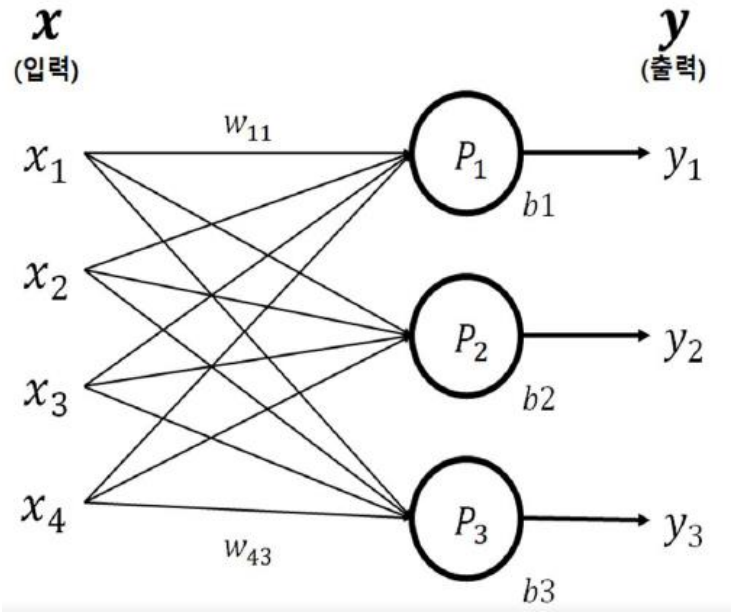




딥러닝(DNN) 이해

➢ SLP(Single Layer Perceptron) - 단층 퍼셉트론

- 입력 벡터 하나로부터 은닉 계층 없이 출력 벡터 하나를 얻어내는 가장 기본적인 신경망 구조
- 단층 퍼셉트론으로 AND, NAND, OR 게이트는 구현가능
- 입력 벡터로부터 출력 벡터를 얻어내려면 출력 벡터의 크기, 즉 출력 벡터가 담고 있어야 할 스칼라 성분의 수만큼의 퍼셉트론이 필요
- 퍼셉트론들은 입력 벡터(x값들)만 공유할 뿐 각자의 가중치 벡터(w)와 편향값(b)에 따라 각자의 방식으로 독립적인 정보를 각각 생산합니다
- 퍼셉트론 끼리 영향을 주고 받을 수 없기 때문에 높은 수준의 문제는 해결할 수 없음



- P1,P2,P3는 크기 3의 출력 벡터를 만들어 내기 위한 퍼셉트론
- 퍼셉트론 사이에는 어떤 연결도 없어서 서로 영향을 주고 받을 수 없습니다.
- 퍼셉트론은 각각의 가중치 벡터와 편향값을 이용하여 입력벡터 x로부터 출력벡터 y를 도출합니다.

딥러닝(DNN) 이해

➢ 퍼셉트론의 인식 능력 이해 와 한계

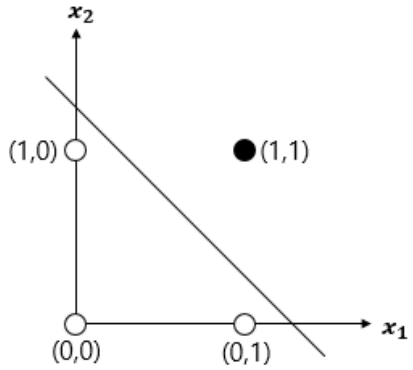
- 선형 영역만 표현할 수 있다.

  데이터들의 차이가 확연히 두드러지지 않는다면 두 가지의 종류조차도 분류하지 못할 수도 있다

  예) AND, NAND, OR 논리회로를 계산할 수 있지만 XOR(배타적 논리합)의 경우 단일 선형 프로세스로 데이터를 분류해낼 수 없다.

- XOR는 여러 개의 선형 분류자를 이용해서 데이터를 나누거나 한 선으로는 곡선으로 나눌 수 있는 비선형 분류이다.

```
def AND_gate(x1, x2):
    w1 = 0.5
    w2 = 0.5
    b = -0.7
    result = x1*w1 + x2*w2 + b
    if result <= 0:
        return 0
    else:
        return 1
AND_gate(0, 0), AND_gate(0, 1), AND_gate(1, 0), AND_gate(1, 1)
```

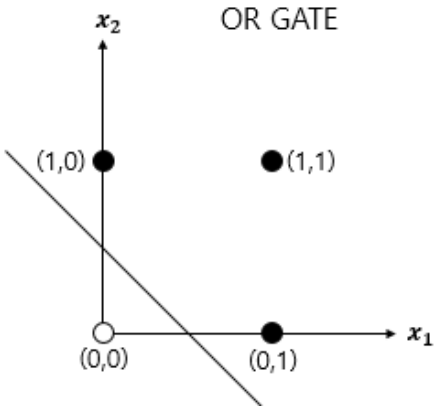


x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

딥러닝(DNN) 이해

➢ 퍼셉트론의 인식 능력 이해 와 한계

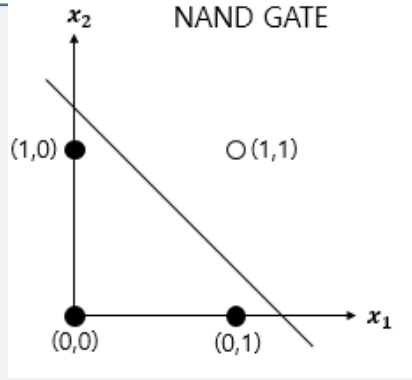
```
def OR_gate(x1, x2):
    w1 = 0.6
    w2 = 0.6
    b = -0.5
    result = x1*w1 + x2*w2 + b
    if result <= 0:
        return 0
    else:
        return 1

OR_gate(0, 0), OR_gate(0, 1), OR_gate(1, 0), OR_gate(1, 1)
```



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

실제 상황에서는 데이터만 주어지므로,
학습 알고리즘으로 가중치(w0,w2,...,wd)를 알아내야 함
신경망의 학습은 최적화 문제이며, 최적화 대상은 신경망의 가중치(매개변수)

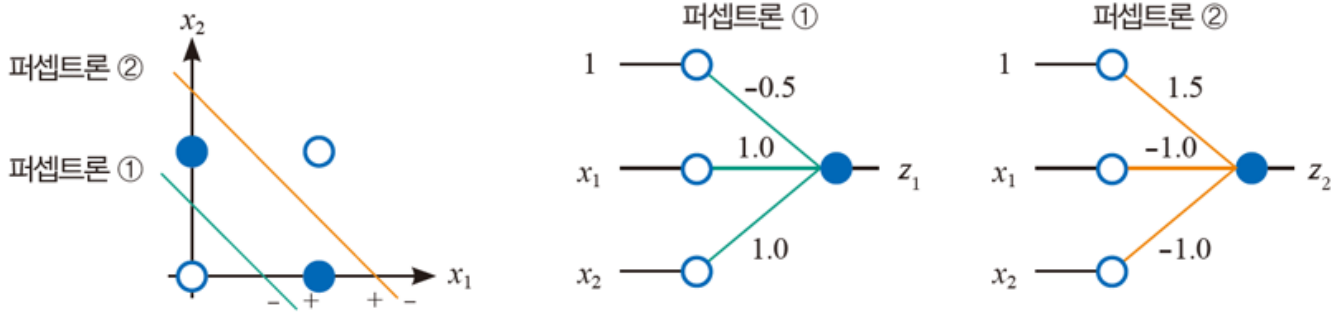
```
def NAND_gate(x1, x2):
    w1 = -0.5
    w2 = -0.5
    b = 0.7
    result = x1*w1 + x2*w2 + b
    if result <= 0:
        return 0
    else:
        return 1

NAND_gate(0, 0), NAND_gate(0, 1), NAND_gate(1, 0), NAND_gate(1, 1)
```



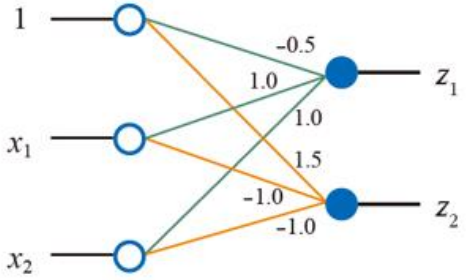
딥러닝(DNN) 이해

➢ 특징 공간 변환

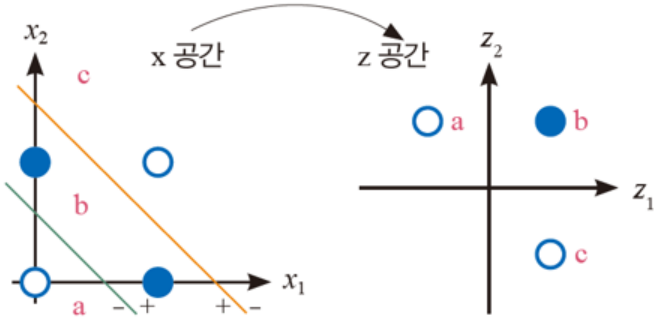
- 퍼셉트론 두 개로 특징 공간을 세 개의 부분 공간으로 나눌 수 있음
- 두 퍼셉트론을 병렬로 결합하면 (x1,x2) 공간을 (z1,z2) 공간으로 변환
- XOR 게이트는 기존의 AND, NAND, OR 게이트를 조합하여 만들 수 있기 때문에 퍼셉트론에서 층을 계속 추가하면서 만들 수 있다.



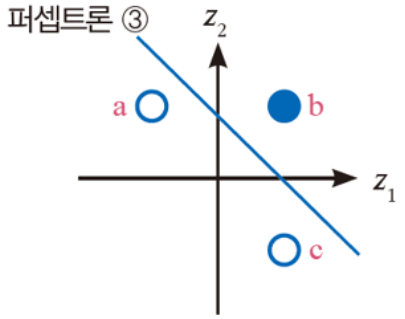
(a) 퍼셉트론 2개를 이용해 부분공간 3개로 분할 (b) 2개의 퍼셉트론



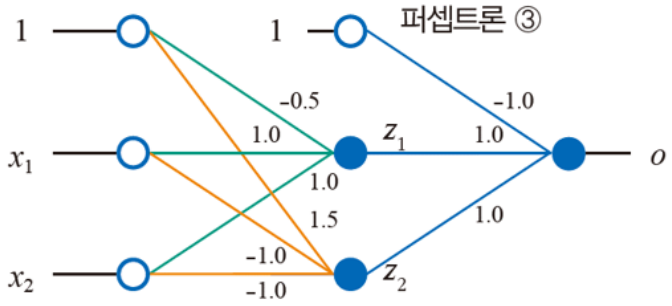
(a) 공간 변환하는 퍼셉트론 2개의 병렬 결합



(b) 특징 공간 변환(세 부분공간을 세 점으로 매핑)



(a) 공간 분할하는 세 번째 퍼셉트론

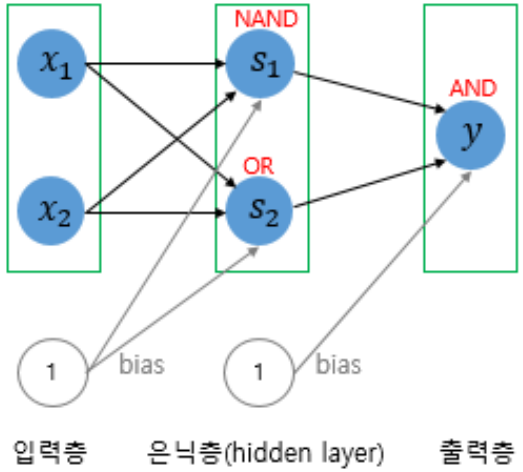


(b) 세 번째 퍼셉트론을 순차적으로 덧붙임

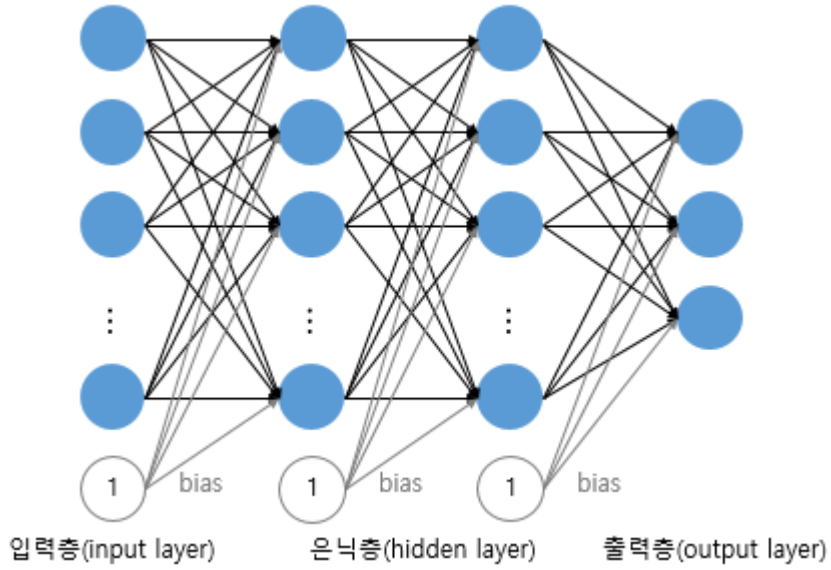
딥러닝(DNN) 이해

➢ MLP(Multi Layer Perceptron) - 다층 퍼셉트론

- 단층 퍼셉트론의 한계를 극복하기 위한 방안으로 입력층과 출력층 사이에 하나 이상의 중간층을 두어 비선형적으로 분리되는 데이터에 대해서도 학습이 가능하도록 고안
- 퍼셉트론(perceptron)을 여러 개 연결하여 층 구조를 갖는 신경망 (현재 인공 신경망과 유사한 구조를 갖게 됨)
- 활성화 함수를 이용한 non-linear 시스템을 여러 layer로 쌓는 개념이다.
- 활성화 함수는 비선형 시스템인 MLP를 이용하여 XOR는 해결될 수 있지만, MLP의 파라미터 개수가 점점 많아지면서 각각의 weight와 bias를 학습시키는 것이 매우 어려워 다시 한 번 침체기를 겪게 되었다. = > 문제 해결 알고리즘 역전파(Back Propagation)



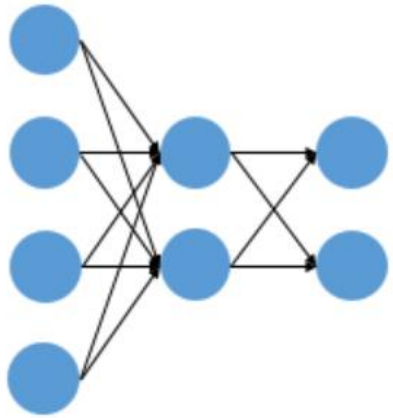
MLP로 XOR 문제를 해결한 예



딥러닝(DNN) 이해

➢ 피드 포워드 신경망(Feed-Forward Neural Network, FFNN)

- 다층 퍼셉트론(MLP)과 같이 오직 입력층에서 출력층 방향으로 연산이 전개되는 신경망



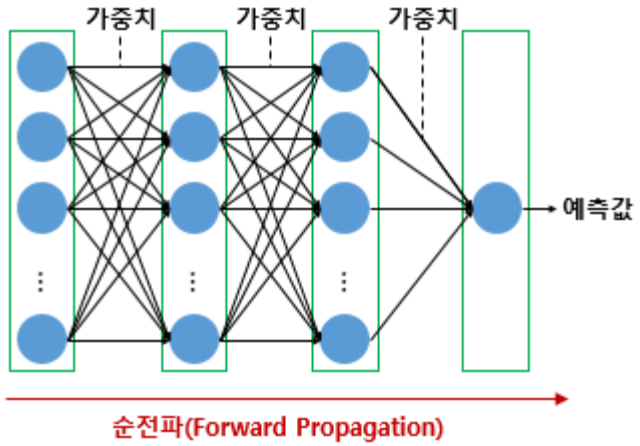
순방향 신경망(Feed-Forward Neural Network)

전결합층(Fully-connected layer, FC, Dense layer) :
어떤 층의 모든 뉴런이 이전 층의 모든 뉴런과 연결돼 있는 층 => 전결합층(Fully-connected layer)
완전연결층
다층 퍼셉트론은 은닉층과 출력층에 있는 모든 뉴런은 바로 이전 층의 모든 뉴런과 연결돼 있는 전결합층입니다.
다층 퍼셉트론의 모든 은닉층과 출력층은 동일한 의미로 밀집층(Dense layer) 이라고 부르기도 하는데, 케라스에서는 밀집층을 구현할 때 Dense()를 사용합니다.

딥러닝(DNN) 이해

➢ 순전파(Foward Propagation)

입력이 입력층으로 들어가서 은닉층을 지나 출력층에서 예측값을 얻는 과정
신경망의 순전파는 행렬의 곱셈으로 이해할 수 있다



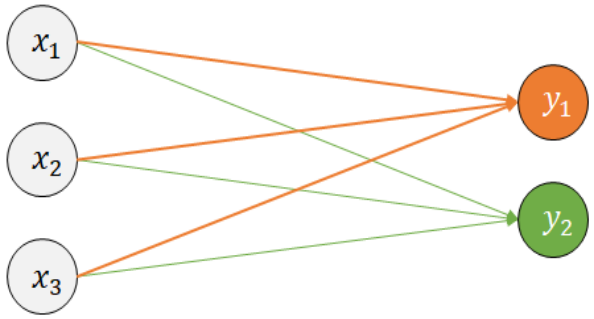
활성화 함수, 은닉층의 수, 각 은닉층의 뉴런 수 등 딥 러닝 모델을 설계하고 나면
입력값은 입력층, 은닉층을 지나면서 각 층에서의 가중치와 함께 연산되며
출력층으로 향합니다

```
#입력의 차원이 3, 출력의 차원이 2인 위 인공 신경망 구현
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(2, input_dim=3, activation='softmax'))
# 케라스 .summary() : 모델에 존재하는 가중치 와 편향 의 개수를 확인
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 8
=================================================================
Total params: 8
Trainable params: 8
Non-trainable params: 0
_________________________________________________________________
```

딥러닝(DNN) 이해

➢ 순전파(Foward Propagation)

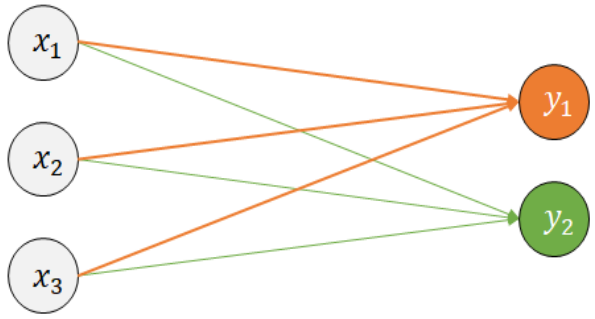


행렬곱 관점에서는 3차원 벡터에서 2차원 벡터가 되기 위해서 3 × 2 행렬을 곱 수행

$$\underset{X}{\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}} \times \underset{W}{\begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix}} + \underset{B}{\begin{bmatrix} b_1 & b_2 \end{bmatrix}} = \underset{Y}{\begin{bmatrix} y_1 & y_2 \end{bmatrix}}$$

$$h_1 = x_1w_1 + x_2w_2 + x_3w_3 + b_1$$
$$h_2 = x_1w_4 + x_2w_5 + x_3w_6 + b_2$$
$$[y_1, y_2] = softmax([h_1, h_2])$$

행렬곱을 사용하면 병렬 연산도 가능하다
'**배치 연산**' - 인공 신경망이 다수의 샘플을 동시에 처리하는 것

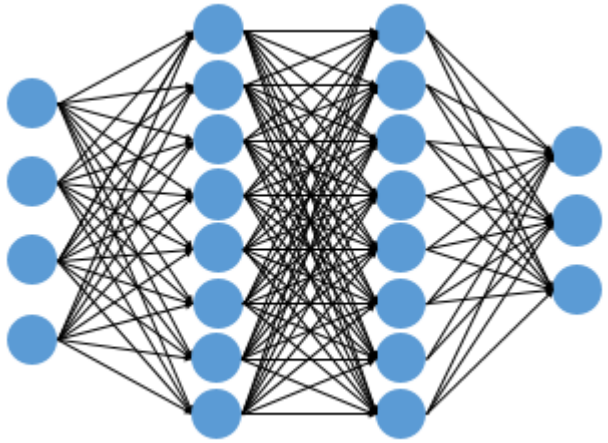


$$\text{Sample \#1}\\ \text{Sample \#2}\\ \text{Sample \#3}\\ \text{Sample \#4}\\ \underset{X}{\begin{bmatrix} x_1 & x_2 & x_3 \\ x_1 & x_2 & x_3 \\ x_1 & x_2 & x_3 \\ x_1 & x_2 & x_3 \end{bmatrix}} \times \underset{W}{\begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix}} + \underset{B}{\begin{bmatrix} b_1 & b_2 \end{bmatrix}} = \underset{Y}{\begin{bmatrix} y_1 & y_2 \\ y_1 & y_2 \\ y_1 & y_2 \\ y_1 & y_2 \end{bmatrix}}$$

4개의 샘플을 동시에 처리하고 있지만, 학습가능한 매개변수의 수는 8개

딥러닝(DNN) 이해

➢ 다층 퍼셉트론의 순전파



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()
# 4개의 입력과 8개의 출력
model.add(Dense(8, input_dim=4, activation='relu'))

model.add(Dense(8, activation='relu')) # 8개의 출력

model.add(Dense(3, activation='softmax'))  # 3개의 출력
```

행렬의 크기
입력층 : 4개의 입력과 8개의 출력
은닉층1 : 8개의 입력과 8개의 출력
은닉층2 : 8개의 입력과 3개의 출력
출력층 : 3개의 입력과 3개의 출력

$$X_{1\times 4}\times W_{4\times 8}+B_{1\times 8}=Y_{1\times 8}$$

$$X_{1\times 8}\times W_{8\times 8}+B_{1\times 8}=Y_{1\times 8}$$

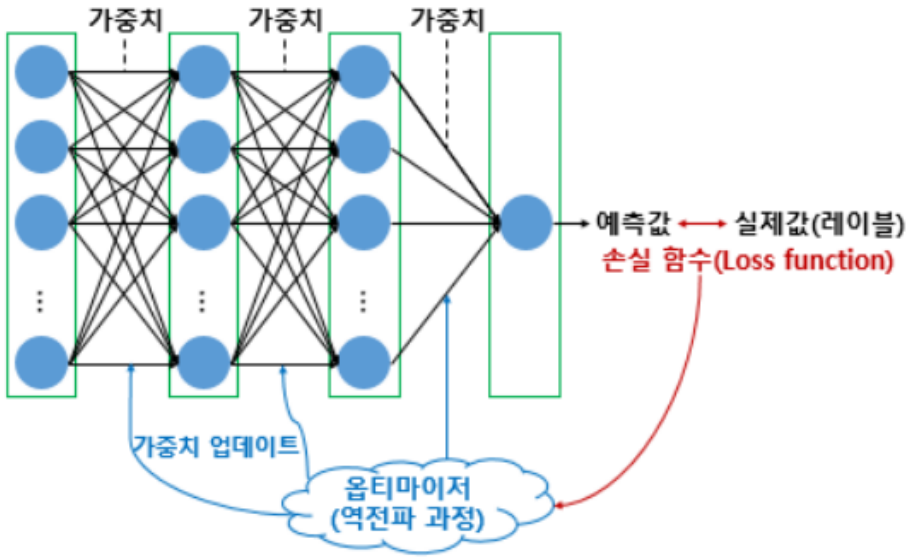
$$X_{1\times 8}\times W_{8\times 3}+B_{1\times 3}=Y_{1\times 3}$$

1. 순전파를 진행하고 예측값을 구하고
2. 예측값과 실제값으로부터 오차를 계산
3. 오차로부터 가중치와 편향을 업데이트 (순전파와는 반대 방향으로 연산을 진행 : **역전파 BackPropagation**)

딥러닝(DNN) 이해

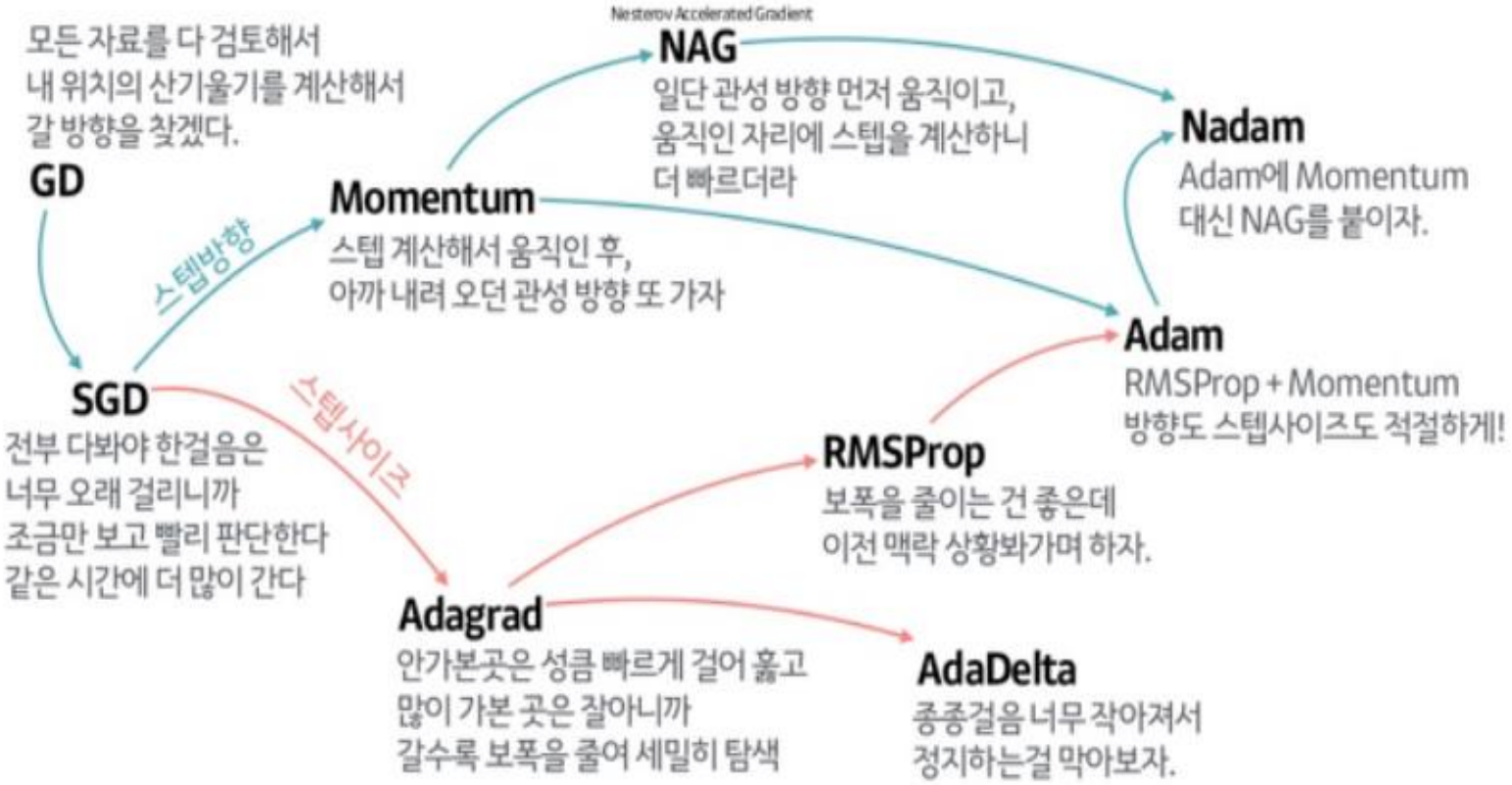
➢ 손실함수(loss function)

- 실제값과 예측값의 차이(loss, cost)를 수치화 해주는 함수
- 오차가 클수록 손실 함수의 값이 크고, 오차가 작을수록 손실 함수의 값이 작아진다.
- 손실 함수의 값을 최소화 하는 W, b를 찾아가는 것이 학습 목표이다.
- 평균제곱오차 : 회귀 , 연속형 변수를 예측할 때 사용
- 시그모이드 함수를 사용하는 이진 분류 (Binary Classification)의 경우 binary_crossentropy를 사용
- 소프트맥스 함수를 사용하는 다중 클래스 분류(Multi-Class Classification)일 경우 categorical_crossentropy를 사용



딥러닝(DNN) 이해

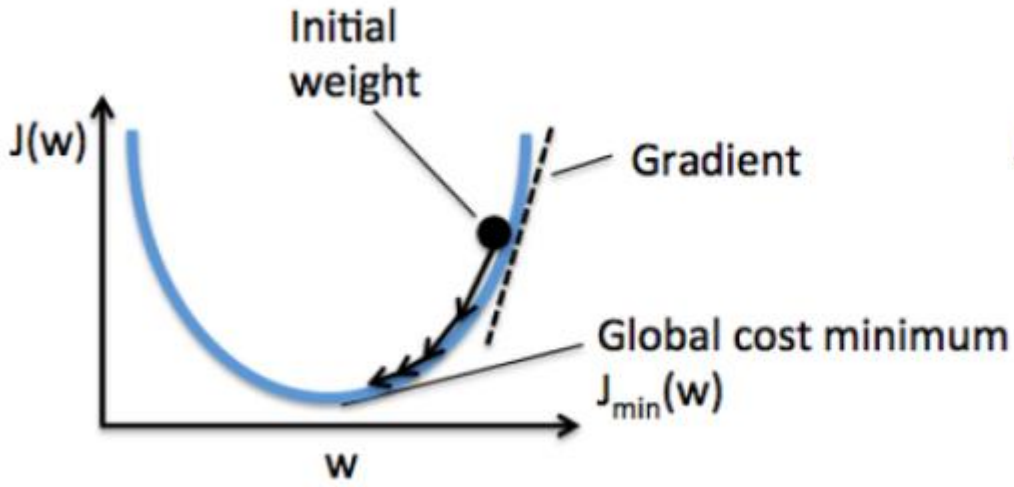
- **최적화 (Optimizer) 알고리즘**
  - 손실함수의 결과값을 최소화하는 함수

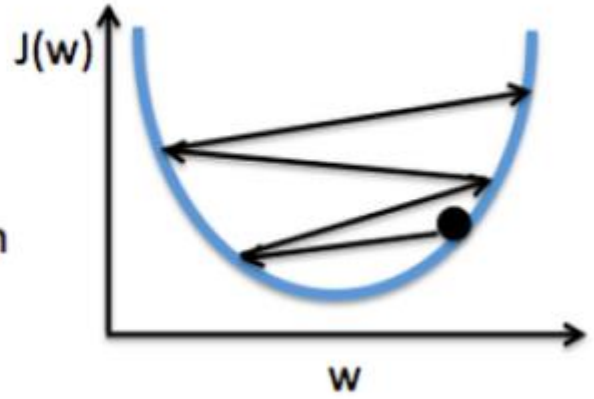


딥러닝(DNN) 이해

➢ 경사 하강법

- 가장 기본적인 optimizer 알고리즘
  경사를 따라 내려가면서 가중치(w) 업데이트 시킨다.
  손실함수를 최소화하기 위하여 반복적으로 파라미터를 조정해나가는 방법
  학습률(learning rate)이 너무 크면 학습시간이 짧아지나 전역 최솟값에서 멀어질 수 있음
  학습률(learning rate)이 너무 작으면 학습시간이 오래걸리고 지역 최솟값에 수렴할 수 있음





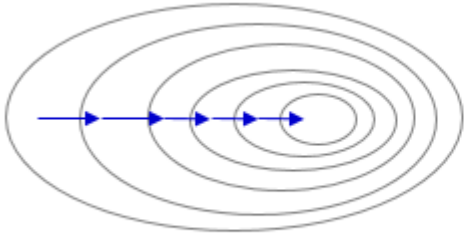
- 기울기에 따라 함수값이 낮아지는 방향으로 이동하는 기법
- 순전파와 역전파를 반복하여 진행하여 신경망 구조를 원하는 방향으로 바꿔나간다.
- 순전파 : 입력 데이터에 대해 신경망 구조를 따라가면서 현재의 파라미터값들을 이용해 손실 함수값을 계산하는 과정
- 역전파 : 순전파의 계산과정을 역순으로 거슬러 가면서 손실 함수값에 직간접적으로 영향을 미친 모든 성분에 대해 손실 기울기를 계산하는 과정

딥러닝(DNN) 이해

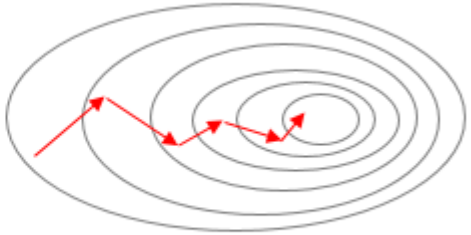
➢ 배치 크기(Batch Size)에 따른 경사 하강법

- 배치는 가중치 등의 매개 변수의 값을 조정하기 위해 사용하는 데이터의 양

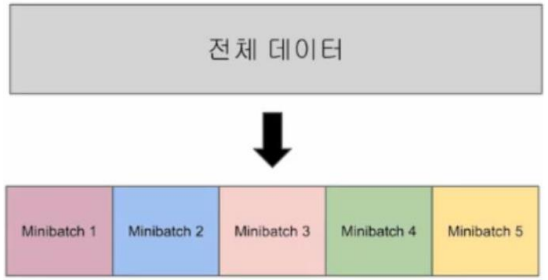
배치 경사 하강법(Batch Gradient Descent)	• 가장 기본적인 경사 하강법으로 오차(loss)를 구할 때 전체 데이터를 고려합니다 • 한 번의 에포크(훈련횟수)에 모든 매개변수 업데이트를 단 한 번 수행 • 한 번의 매개 변수 업데이트에 시간이 오래 걸리며, 메모리를 크게 요구한다 model.fit(X_train, y_train, batch_size=len(X_train))
배치 크기가 1인 확률적 경사 하강법(Stochastic Gradient Descent, SGD)	• 랜덤으로 선택한 하나의 데이터에 대해서만 계산하는 방법 • 더 적은 데이터를 사용하므로 더 빠르게 계산할 수 있습니다. model.fit(X_train, y_train, batch_size=1)
MiniBatch Gradient Descent	• 배치 크기를 지정하여 해당 데이터 개수만큼에 대해서 계산하여 매개 변수의 값을 조정하는 경사 하강법 •전체 데이터를 계산하는 것보다 빠르며, SGD보다 안정적 •배치 크기는 일반적으로 2의 n제곱에 해당하는 숫자로 선택하는 것이 보편적 •배치 크기기본값은 32 model.fit(X_train, y_train, batch_size=128)



경사 하강법



SGD



딥러닝(DNN) 이해

➢ 최적화 (Optimizer) 알고리즘

모멘텀(Momentum)	• 경사 하강법 + 관성 • 경사하강법에서 계산된 접선의 기울기에 한 시점(step)전의 접선의 기울기 값을 일정한 비율만큼 반영, 지역 최솟값에 빠지더라도 관성의 힘으로 빠져나올 수 있다. tf.keras.optimizers.SGD(lr=0.01, momentum=0.9)
아다그라드(Adagrad)	• 각 매개변수에 서로 다른 학습률을 적용시킵니다. • 변화가 많은 매개변수는 학습률이 작게 설정되고 변화가 적은 매개변수는 학습률을 높게 설정시킵니다. tf.keras.optimizers.Adagrad(lr=0.01, epsilon=1e-6)
알엠에스프롭(RMSprop)	• 학습이 진행됨에 따라 parameter사이 차별화는 유지하되 학습속도가 지속적으로 줄어들어 0에 수렴하는 것은 방지 할 수 있게 된다. tf.keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-06)
아담(Adam)	• RMSprop + 모멘텀 • 고정된 학습률의 단점을 보완하기 위하여 학습 초반에는 큰 학습률을 사용하고, 바닥점에 가까워 질수록 학습률을 줄이는 기법 adam = tf.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['acc'])

딥러닝(DNN) 이해

➢ MLP(Multi Layer Perceptron) 문제점 Overfitting을 줄여 Regularization 수행 방법

1. **데이터의 양을 늘릴 수록** 모델은 데이터의 일반적인 패턴을 학습하여 과적합을 방지할 수 있습니다.
2. 인공 신경망의 복잡도 **은닉층(hidden layer)의 수나 매개변수의 수 줄이기**
3. **가중치 규제(Regularization) 적용**
   L1 규제 (L1 norm) : 가중치 w들의 절대값 합계를 비용 함수에 추가합니다.
   L2 규제 (L2 norm) : 모든 가중치 w들의 제곱합을 비용 함수에 추가합니다.
   L1 규제를 사용하면 비용 함수가 최소가 되게 하는 가중치와 편향을 찾는 동시에 가중치들의 절대값의 합도 최소가 되어야 합니다.
   L2 규제는 L1 규제와는 달리 가중치들의 제곱을 최소화하므로 w의 값이 완전히 0이 되기보다는 0에 가까워지기는 경향을 띕니다.
   경험적으로는 L2 규제가 더 잘 동작하므로 L2 규제(가중치 감쇠 weight decay)를 더 권장합니다.

데이터 증식 또는 증강(Data Augmentation) :
데이터의 양이 적을 경우에는 의도적으로 기존의 데이터를 조금씩 변형하고 추가하여 데이터의 양을 늘림
이미지의 경우 데이터 증식이 많이 사용되는데 이미지를 돌리거나 노이즈를 추가하고, 일부분을 수정하는 등으로 데이터를 증식시킵니다.
텍스트 데이터의 경우에는 번역 후 재번역을 통해 새로운 데이터를 만들어내는 역번역(Back Translation) 등의 방법으로 증식시킵니다.

딥러닝(DNN) 이해

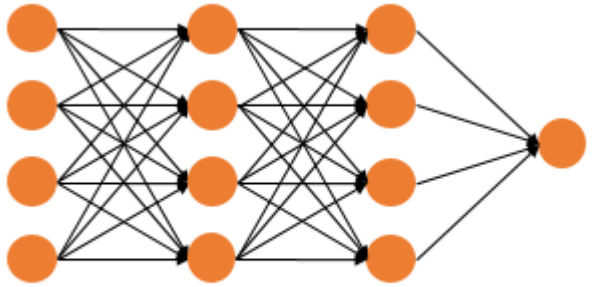
➢ MLP(Multi Layer Perceptron) 문제점 Overfitting을 줄여 Regularization 수행 방법

4. **드롭아웃(Dropout)**

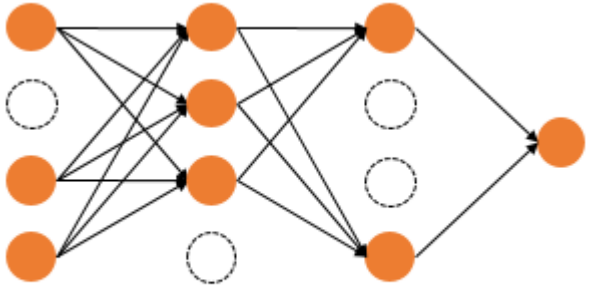
학습 과정에서 신경망의 일부를 사용하지 않는 방법

드롭아웃은 신경망 학습 시에만 사용하고, 예측 시에는 사용하지 않는 것이 일반적입니다.

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 방지합니다.



드롭아웃 적용 전



드롭아웃 적용 후

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Dense

max_words = 10000
num_classes = 46

model = Sequential()
model.add(Dense(256, input_shape=(max_words,), activation='relu'))
model.add(Dropout(0.5)) # 드롭아웃 추가. 비율은 50%
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) # 드롭아웃 추가. 비율은 50%
model.add(Dense(num_classes, activation='softmax'))
```

딥러닝(DNN) 이해

➢ SLP(Single Layer Perceptron) vs MLP(Multi Layer Perceptron)

특징	신경망(ANN)	심층 신경망 (DNN)
층의 수	1개 또는 소수의 은닉층	다수의 은닉층
모델 복잡도	상대적으로 단순	복잡하며 많은 뉴런과 층을 포함
데이터 처리 능력	비교적 단순한 문제에 적합	복잡한 데이터 패턴과 대규모 데이터셋 처리에 적합
학습 시간 및 자원	상대적으로 짧고 적은 자원 사용	긴 학습 시간과 많은 컴퓨팅 자원 필요
주요 응용 분야	기본적인 패턴 인식 문제, 작은 데이터셋 처리	이미지 인식, 자연어 처리, 음성 인식 등 복잡한 문제 처리
학습 방법	주로 역전파	역전파, 최적화 기법, 정규화 기법 등 다양한 방법 사용

딥러닝(DNN) 이해

➢ DNN 모델 적용

```
# 1. 데이터 - 예를 들어 (공부시간, 공부량) 데이터를 넣어 (점수, 등수)를 예측할때, 사용할 수 있는 모델
import numpy as np
x = np.array([range(1,101), range(101,201)])       # 데이터가 두개!
y = np.array([range(1,101), range(101,201)])
print(x)
print(x.shape) # (2,100) 2행 100열

x = np.transpose(x)
y = np.transpose(y)
print(x.shape) # (100,2)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=66, test_size=0.4)
x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, random_state=66, test_size=0.5)

# 2. 모델 구성
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(10, input_shape=(2, ), activation='relu'))
model.add(Dense(10))
model.add(Dense(8))
model.add(Dense(2)) # input col 2개 이므로 output도 2개
```

딥러닝(DNN) 이해

➢ DNN 모델 적용

```
# 3. 훈련
model.compile(loss='mse', optimizer='adam', metrics=['mse'])
model.fit(x_train, y_train, epochs=100, batch_size=1, validation_data=(x_val, y_val))

# 4. 평가 예측
loss, mse = model.evaluate(x_test, y_test, batch_size=1)
print('acc : ', mse)

y_predict = model.predict(x_test)
print(y_predict)

# RMSE 구하기
from sklearn.metrics import mean_squared_error
def RMSE(y_test, y_predict):
    return np.sqrt(mean_squared_error(y_test, y_predict))
print('RMSE : ', RMSE(y_test, y_predict))

# R2 구하기
from sklearn.metrics import r2_score
r2_y_predict = r2_score(y_test, y_predict)
print('R2 : ', r2_y_predict)
```

딥러닝(DNN) 이해

➢ DNN 모델 적용 와인 분류

- 레드와인 샘플 1,599개를 등급과 맛, 산도를 측정해 분석하고 화이트와인 샘플 4,898개를 분석
- 레드와인과 화이트와인을 구분하는 실험을 진행

	0	1	2	3	4	5	6	7	8	9	10	11	12
964	8.5	0.47	0.27	1.9	0.058	18	38	0.99518	3.16	0.85	11.1	6	1
664	12.1	0.4	0.52	2	0.092	15	54	1	3.03	0.66	10.2	5	1
1692	6.9	0.21	0.33	1.8	0.034	48	136	0.9899	3.25	0.41	12.6	7	0
5801	6.7	0.24	0.31	2.3	0.044	37	113	0.99013	3.29	0.46	12.9	6	0
2207	6.1	0.28	0.25	17.75	0.044	48	161	0.9993	3.34	0.48	9.5	5	0

0열 : 주석산 농도
1열 : 아세트산 농도
2열 : 구연산 농도
3열 : 잔류 당분 농도
4열 : 염화나트륨 농도
5열 : 유리 아황산 농도
6열 : 총 아황산 농도
7열 : 밀도
8열 : pH
9열 : 황산칼륨 농도
10열 : 알코올 도수
11열 : 와인의 맛(0~10등급)
12열 : class (1: 레드와인, 0: 화이트와인)

딥러닝(DNN) 이해

➢ DNN 모델 적용 와인 분류

```
# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)
# 데이터 입력
df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=1)
dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
# 모델 설정
model = Sequential()
model.add(Dense(30,  input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
#모델 컴파일
model.compile(loss='binary_crossentropy',    optimizer='adam',
metrics=['accuracy'])
# 모델 실행
model.fit(X, Y, epochs=200, batch_size=200)
# 결과 출력
print("₩n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```

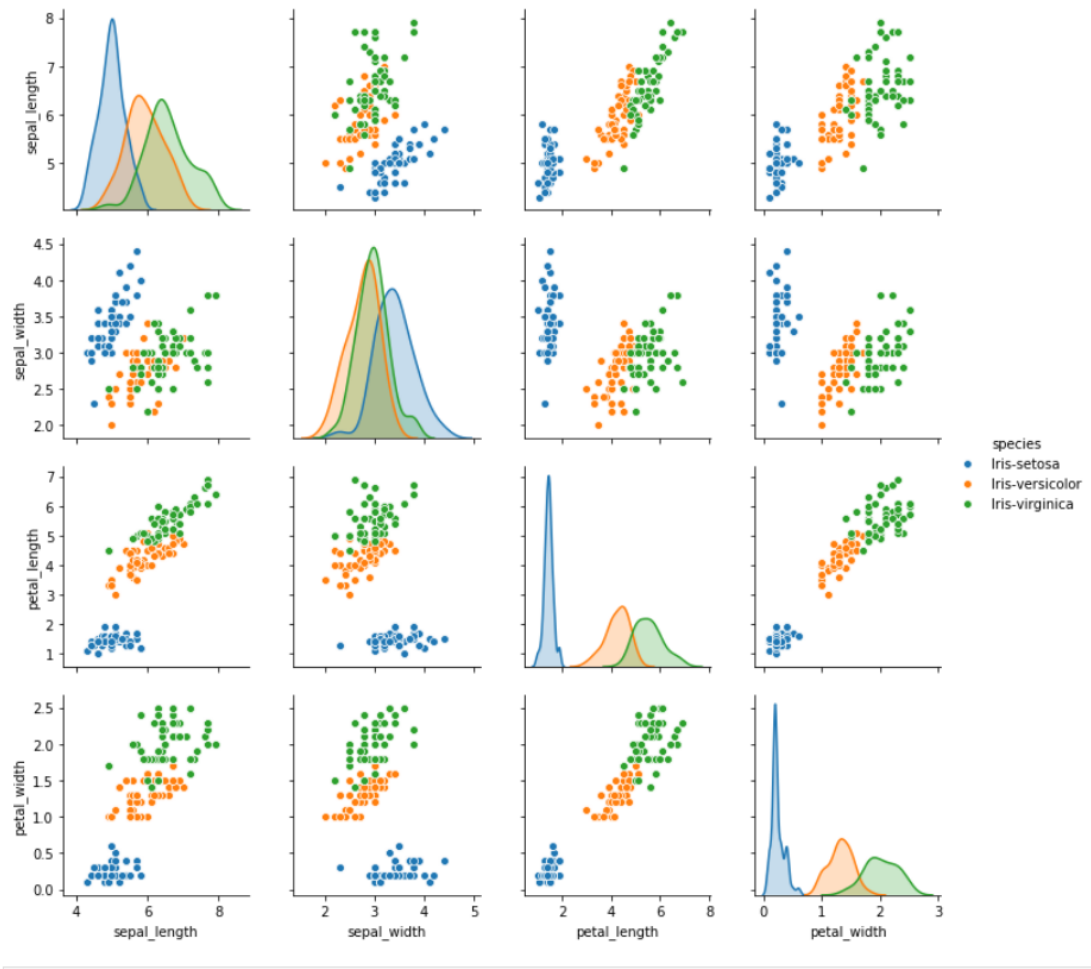
```
Epoch 1/200
6497/6497 [==============================] - 0s 26us/step - loss: 3.5323 - accuracy: 0.2461
Epoch 2/200
6497/6497 [==============================] - 0s 6us/step - loss: 0.4326 - accuracy: 0.8110
Epoch 3/200
6497/6497 [==============================] - 0s 7us/step - loss: 0.2418 - accuracy: 0.9257
Epoch 4/200
6497/6497 [==============================] - 0s 7us/step - loss: 0.2142 - accuracy: 0.9300
Epoch 5/200
6497/6497 [==============================] - 0s 7us/step - loss: 0.2040 - accuracy: 0.9304
```

```
Epoch 198/200
6497/6497 [==============================] - 0s 6us/step - loss: 0.0496 - accuracy: 0.9858
Epoch 199/200
6497/6497 [==============================] - 0s 7us/step - loss: 0.0499 - accuracy: 0.9848
Epoch 200/200
6497/6497 [==============================] - 0s 6us/step - loss: 0.0493 - accuracy: 0.9865
6497/6497 [==============================] - 0s 18us/step

 Accuracy: 0.9858
```

딥러닝(DNN) 이해

➢ MLP(Multi Layer Perceptron) 모델 적용 아이리스 품종 예측

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import LabelEncoder

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.
np.random.seed(3)
tf.random.set_seed(3)

# 데이터 입력
df = pd.read_csv('../dataset/iris.csv', names = ["sepal_length",
"sepal_width", "petal_length", "petal_width", "species"])

# 그래프로 확인
sns.pairplot(df, hue='species');
plt.show()
```



딥러닝(DNN) 이해

➢ MLP(Multi Layer Perceptron) 모델 적용 아이리스 품종 예측

```
# 데이터 분류
dataset = df.values
X = dataset[:,0:4].astype(float)
Y_obj = dataset[:,4]

# 문자열을 숫자로 변환
e = LabelEncoder()
e.fit(Y_obj)
Y = e.transform(Y_obj)
Y_encoded = tf.keras.utils.to_categorical(Y)

# 모델의 설정
model = Sequential()
model.add(Dense(16,  input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))

# 모델 컴파일
model.compile(loss='categorical_crossentropy',
optimizer='adam',          metrics=['accuracy'])
# 모델 실행
model.fit(X, Y_encoded, epochs=50, batch_size=1)

# 결과 출력
print("₩n Accuracy: %.4f" % (model.evaluate(X, Y_encoded)[1]))
```

```
Train on 150 samples
Epoch 1/50
150/150 [==============================] - 0s 3ms/sample - loss: 1.2541 - accuracy: 0.3733
Epoch 2/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.9932 - accuracy: 0.4200
Epoch 3/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.7919 - accuracy: 0.6400
Epoch 4/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.6278 - accuracy: 0.8467
Epoch 5/50
150/150 [==============================] - 0s 999us/sample - loss: 0.5272 - accuracy: 0.9067
Epoch 6/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.4574 - accuracy: 0.9267
Epoch 7/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.4127 - accuracy: 0.9333
Epoch 8/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.3813 - accuracy: 0.9267
Epoch 9/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.3491 - accuracy: 0.9400
Epoch 10/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.3168 - accuracy: 0.9733

Epoch 46/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.0962 - accuracy: 0.9667
Epoch 47/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.0926 - accuracy: 0.9867
Epoch 48/50
150/150 [==============================] - 0s 993us/sample - loss: 0.0924 - accuracy: 0.9733
Epoch 49/50
150/150 [==============================] - 0s 1ms/sample - loss: 0.0884 - accuracy: 0.9667
Epoch 50/50
150/150 [==============================] - 0s 999us/sample - loss: 0.0894 - accuracy: 0.9733

 Accuracy: 0.9867
```

딥러닝(DNN) 이해

➢ 다층 퍼셉트론(MultiLayer Perceptron, MLP)으로 텍스트 분류

- **texts_to_matrix()** : 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구
- texts_to_matrx()는 'binary', 'count', 'freq', 'tfidf' 4개의 모드를 지원
  binary 모드는 단어의 존재 유무로만 행렬을 표현
  '**tfidf**' 모드에서 TF는 각 문서에서의 각 단어의 빈도에 자연 로그를 씌우고 1을 더한 값으로 정의하고
  idf에서는 로그는 자연 로그를 사용하고, 로그 안의 분수에 1을 추가로 더합니다
  '**freq**' 모드는 각 문서에서의 각 단어의 등장 횟수를 분자로, 각 문서의 크기(각 문서에서 등장한 모든 단어의 개수의 총 합)를 분모로 하는 표현
- 문서 단어 행렬(Document-Term Matrix, DTM)을 생성 결과에서 각 단어에 부여되는 인덱스는 1부터 시작하는 반면에 완성되는 행렬의 인덱스는 0부터 시작합니다.
- DTM은 bag of words를 기반으로 하므로 단어 순서 정보는 보존되지 않습니다.

```
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

newsdata = fetch_20newsgroups(subset = 'train')  #20개의 다른 주제를 가진 18,846개의 뉴스 그룹 이메일 데이터
print(newsdata.keys())
print('훈련용 샘플의 개수 : {}'.format(len(newsdata.data)))
print('총 주제의 개수 : {}'.format(len(newsdata.target_names)))
print(newsdata.target_names)
print('첫번째 샘플의 레이블 : {}'.format(newsdata.target[0]))
```

딥러닝(DNN) 이해

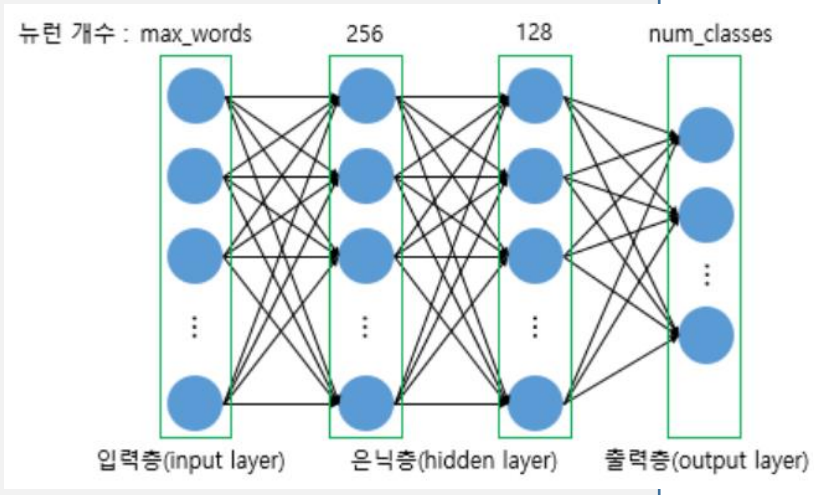
➢ 다층 퍼셉트론(MultiLayer Perceptron, MLP)으로 텍스트 분류

```
print('7번 레이블이 의미하는 주제 : {}'.format(newsdata.target_names[7]))
print(newsdata.data[0]) # 첫번째 샘플 출력
data = pd.DataFrame(newsdata.data, columns = ['email'])
data['target'] = pd.Series(newsdata.target)
data[:5]
data.info()
data.isnull().values.any()
print('중복을 제외한 샘플의 수 : {}'.format(data['email'].nunique()))
print('중복을 제외한 주제의 수 : {}'.format(data['target'].nunique()))
data['target'].value_counts().plot(kind='bar');
print(data.groupby('target').size().reset_index(name='count'))
newsdata_test = fetch_20newsgroups(subset='test', shuffle=True)
train_email = data['email']
train_label = data['target']
test_email = newsdata_test.data
test_label = newsdata_test.target
vocab_size = 10000                #실습에서 사용할 최대 단어 개수를 정의하는 변수
num_classes = 20
def prepare_data(train_data, test_data, mode):                # 전처리 함수
    tokenizer = Tokenizer(num_words = vocab_size)        # vocab_size 개수만큼의 단어만 사용한다.
    tokenizer.fit_on_texts(train_data)
    X_train = tokenizer.texts_to_matrix(train_data, mode=mode)       # 샘플 수 × vocab_size 크기의 행렬 생성
    X_test = tokenizer.texts_to_matrix(test_data, mode=mode)         # 샘플 수 × vocab_size 크기의 행렬 생성
    return X_train, X_test, tokenizer.index_word
```

딥러닝(DNN) 이해

➢ 다층 퍼셉트론(MultiLayer Perceptron, MLP)으로 텍스트 분류

```
X_train, X_test, index_to_word = prepare_data(train_email, test_email, 'binary') # binary 모드로 변환
y_train = to_categorical(train_label, num_classes) # 원-핫 인코딩
y_test = to_categorical(test_label, num_classes) # 원-핫 인코딩
print('훈련 샘플 본문의 크기 : {}'.format(X_train.shape))
print('훈련 샘플 레이블의 크기 : {}'.format(y_train.shape))
print('테스트 샘플 본문의 크기 : {}'.format(X_test.shape))
print('테스트 샘플 레이블의 크기 : {}'.format(y_test.shape))
print('빈도수 상위 1번 단어 : {}'.format(index_to_word[1]))
print('빈도수 상위 9999번 단어 : {}'.format(index_to_word[9999]))
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
def fit_and_evaluate(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Dense(256, input_shape=(vocab_size,), activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1, validation_split=0.1)
    score = model.evaluate(X_test, y_test, batch_size=128, verbose=0)
    return score[1]
```



딥러닝(DNN) 이해

➢ 다층 퍼셉트론(MultiLayer Perceptron, MLP)으로 텍스트 분류

```
modes = ['binary', 'count', 'tfidf', 'freq'] # 4개의 모드를 리스트에 저장.

for mode in modes: # 4개의 모드에 대해서 각각 아래의 작업을 반복한다.
    X_train, X_test, _ = prepare_data(train_email, test_email, mode) # 모드에 따라서 데이터를 전처리
    score = fit_and_evaluate(X_train, y_train, X_test, y_test) # 모델을 훈련하고 평가.
    print(mode+' 모드의 테스트 정확도:', score)
```