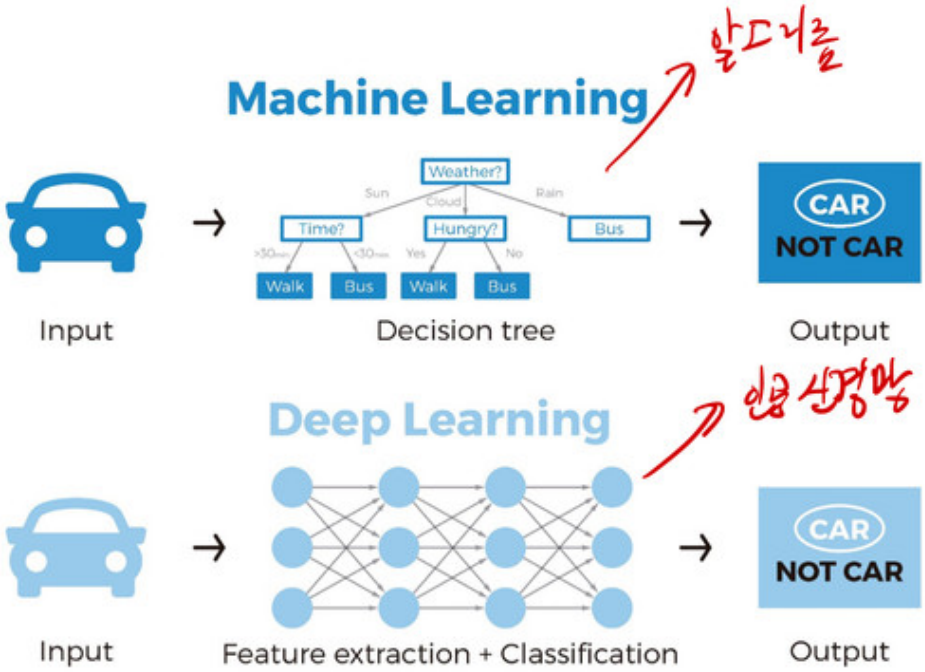
2021 딥러닝 중간고사 과제

인공지능(AI)란

- 컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 모든 기술
- 머신러닝
  - 머신러닝은 기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야
  - SVM(Support Vector Machine): 수학적인 방식의 학습 알고리즘
  - 딥러닝
    - 다중 계층의 신경망 모델을 사용하는 머신러닝의 일종.

머신러닝과 딥러닝의 차이

- 머신러닝
  - 머신러닝은 알고리즘을 통하여 학습시켜 작업 수행 방법을 익히는것
- 딥러닝
  - 딥러닝은 인공신경망에서 발전한 형태의 인공 지능으로, 뇌의 뉴런과 유사한 정보 입출력 계층을 활용해 데이터를 학습하는것
  - 딥러닝은 스스로 학습이 가능



인공신경망과 DDN

- 인공신경망(ANN)
  - 기계학습과 인지과학에서 생물학의 신경망(동물의 중추신경계중 특히 뇌)에서 영감을 억은 수리적 학습 알고리즘
- 심층신경망(DNN)
  - 학습 성능을 높이는 고유 특징만 스스로 추출하여 학습하는 알고리즘
  - 입력 값에 대해 여러 단계의 심층신경망을 거쳐 자율적으로 사고 및 결론 도출

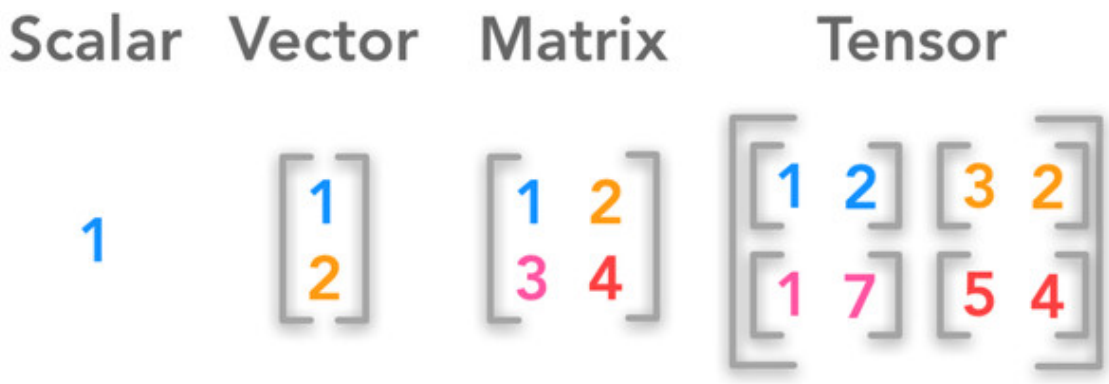
텐서플로(TensorFlow)

- 구글에서 만든 라이브러리
  - 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공
    - 초보자 및 전문가에게 데스크톱, 모바일, 웹, 클라우드 개발용 API를 제공
    - 연구 및 프로덕션용 오픈소스 딥러닝 라이브러리
- 구현 및 사용
  - 텐서플로 자체는 C++로 구현
    - Python, Java, Go등 다양한 언어를 지원
  - 파이썬을 최우선으로 지원
    - 대부분 편한 기능들이 파이썬 라이브러리로만 구현되어 있어 Python에서 개발하는 것이 편함

텐서

- 딥러닝에서 데이터를 표현하는 방식
  - 0-D텐서 : 스칼라
    - 10
  - 1-D텐서 : 벡터
    - [10, 20, 30]
  - 2-D텐서 : 행렬 등
    - [[1,2,3],[4,5,6]]
- n차원 행렬(배열)

- 텐서는 행렬로 표현할 수 있는 n차원 형태의 배열을 높은 차원으로 확장

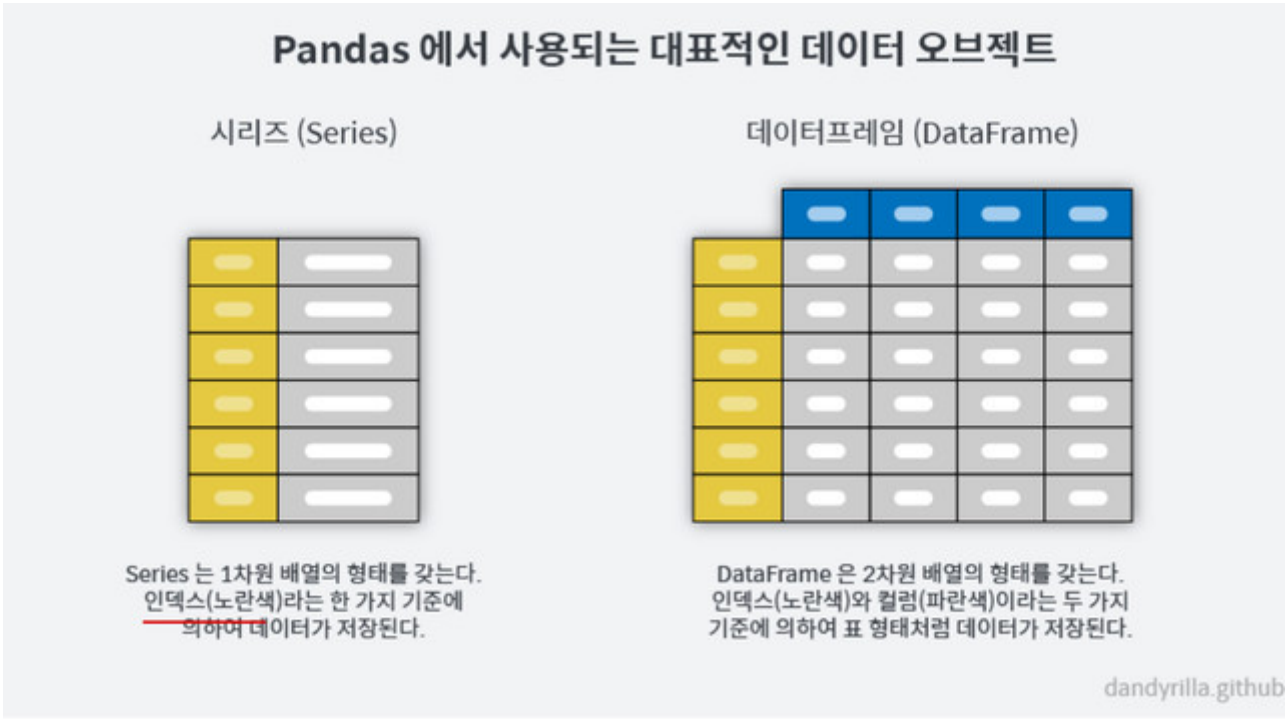



```
a=42#Scalar
b=[1,2,3]#Vactor
c=[[1,2,3],[4,5,6]]#Matrix
d=[[[2],[3],[6]],[[8],[7]]]#n-Tensor
print(a)
print(b)
print(c)
print(d)
```

```
42
[1, 2, 3]
[[1, 2, 3], [4, 5, 6]]
[[[2], [3], [6]], [[8], [7]]]
```

판다스

- 표 형식의 데이터나 다양한 형태의 테이블를 처리하기 위한 라이브러리
- 주 자료 구조




```
import pandas as pd
print(pd.__version__)
data = ["A","B","C","D","E","F","G","H","I","J"] #리스트
print(data)
print("자료형 : ",type(data) )
datapd=pd.Series(data)#자료형 판다스로 변환
print(datapd)
print("자료형 : ",type(datapd))
```

```
1.1.5
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
자료형 :  <class 'list'>
0    A
1    B
2    C
3    D
4    E
5    F
6    G
7    H
8    I
9    J
dtype: object
자료형 :  <class 'pandas.core.series.Series'>
```

```
print(datapd[1])#datapd.loc[1]
print(datapd.loc[1])#datapd[1]
print(datapd[2:4])#datapd[a:b]==>a~b-1출력
print(datapd.loc[2:3])#datapd.loc[a:b]==>a~b출력
```

```
B
B
2    C
```

```
3    D
dtype: object
2    C
3    D
dtype: object
```

DataFrame

- R의 dataframe 데이터 타입을 참고하여 만든 것이 바로 pandas DataFrame
- 행과 열을 인덱스(index)와 칼럼(columns)으로 구분

```
data2=(1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.10)
datapd2=pd.Series(data2)
data3=(["가","나","다","라","마","바","사","아","자","차"])
datapd3=pd.Series(data3)
dict_data={'D1':datapd ,'D2':datapd2,'D3':datapd3}
dataframe=pd.DataFrame(dict_data)
dataframe
```

	D1	D2	D3
0	A	1.1	가
1	B	2.2	나
2	C	3.3	다
3	D	4.4	라
4	E	5.5	마
5	F	6.6	바
6	G	7.7	사
7	H	8.8	아
8	I	9.9	자
9	J	10.1	차

열 제목 변경

```
dataframe.columns = ['string','number','hangul']
dataframe
```

	string	number	hangul
0	A	1.1	가
1	B	2.2	나
2	C	3.3	다
3	D	4.4	라
4	E	5.5	마
5	F	6.6	바

행 인덱스 변경

```
dataframe.index=['r0','r1','r2','r3','r4','r5','r6','r7','r8','r9']
dataframe
```

	string	number	hangul
r0	A	1.1	가
r1	B	2.2	나
r2	C	3.3	다
r3	D	4.4	라
r4	E	5.5	마
r5	F	6.6	바
r6	G	7.7	사
r7	H	8.8	아
r8	I	9.9	자
r9	J	10.1	차

DataFrame 여러 출력 방법

```
print(dataframe.loc['r1','string'])#특정 원소 출력
print("\n")
print(dataframe.loc['r2':'r3', 'number':'hangul'])#dataframe.loc['a':'b', 'A':'B']a
print("\n")
print(dataframe.loc['r2':'r3', 'number'])#특정 열의 원소만 출력
print("\n")
print(dataframe.loc['r2', 'string':'number'])#특정 행의 원소만 출력
print("\n")
print(dataframe.loc[:, 'string'])#특정 열 모든 원소 출력
print("\n")
print(dataframe.loc['r2':'r3', :])#특정 행 모든 원소 출력
```

```
B
```

```
    number hangul
r2     3.3      다
r3     4.4      라


r2    3.3
r3    4.4
Name: number, dtype: float64


string      C
number    3.3
Name: r2, dtype: object


r0    A
r1    B
r2    C
r3    D
r4    E
r5    F
r6    G
r7    H
r8    I
r9    J
Name: string, dtype: object


   string  number hangul
r2      C     3.3      다
r3      D     4.4      라
```

회귀

회귀 모델

- 연속적인 값을 예측
  - ex)캘리포니아의 주택 가격, 사용자가 광고를 클릭할 확률

회귀 분석

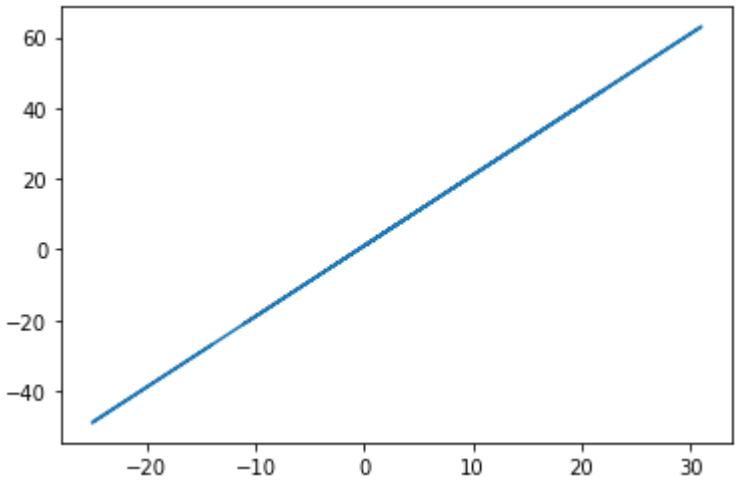
- 관찰된 연속형 변수들에 대해 두 변수 사이의 모형을 구한 뒤 적합도를 측정해 내는 분석 방법
- 회귀분석은 시간에 따라 변화하는 데이터나 어떤 영향, 가설적 실험, 인과 관계의 모델링 등의 통계적 예측에 이용
- 선형 회귀
  - 다순 선형 회귀 분석(Simple Linear Rehression Analysis)
    - 입력: 특징이 하나
    - 출력: 하나의 값
    - H(x) = Wx + b
  - 다중 선형 회귀 분석(Multiple Linear Rehression Analysis)

- 입력: 특징이 여러개
- 출력: 하나의 값
- y=W1x1+W2x2+...Wnxn+b

선형 회귀 구현

```
x = [-3,  31,  -11,  4,  0,  22, -2, -5, -25, -14]
y = [ -5,   63,   -21,   9,  1,   45,  -3,  -9, -49,  -27]
print("x: ",x)
print("y: ",y)
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.show()
```

```
x:  [-3, 31, -11, 4, 0, 22, -2, -5, -25, -14]
y:  [-5, 63, -21, 9, 1, 45, -3, -9, -49, -27]
```




```
import pandas as pd
df = pd.DataFrame({'X':x, 'Y':y})
df
```

	x	y

➕플러스 지식➕

head()와 tail()

- haed(a)는 a만큼 위에서부터 출력해준다.
  - 디폴트값이 있어 매개변수로 아무 값도 넣어주지 않으면 위에서 부터 5개를 출력해준다
- tail(a)는 a만큼 아래에서부터 출력해준다.
  - 디폴트값이 있어 매개변수로 아무 값도 넣어주지 않으면 아래에서 부터 5개를 출력해준다

```
df.head(3)
```

	x	y
0	-3	-5
1	31	63
2	-11	-21

```
df.head()
```

	x	y
0	-3	-5
1	31	63
2	-11	-21
3	4	9
4	0	1

```
df.tail(3)
```

	x	y
7	-5	-9
8	-25	-49
9	-14	-27

```
df.tail()
```

	X	Y
5	22	45
6	-2	-3
7	-5	-9
8	-25	-49

```
##훈련용 데이터 만들기
train_features = ['X']
target_cols = ['Y']
X_train = df.loc[:, train_features]
y_train = df.loc[:, target_cols]
print(X_train.shape, y_train.shape)
```

```
(10, 1) (10, 1)
```

```
X_train
```

	X
0	-3
1	31
2	-11
3	4
4	0
5	22
6	-2
7	-5
8	-25
9	-14

```
y_train
```

	Y
0	-5
1	63
2	-21

배경지식

- fit()함수는 매겨변수로 훈련용 문제(X_train) 문제에 대한 정답(Y_train)을 넘겨받아 데이터로 훈련시켜 훈련용 모델을 만든다

  X_train==>훈련용 문제
  Y_train==>훈련용 문제에 정답
  X_test==>테스트 문제
  Y_test==>테스트 문제 정답

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)#훈련
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False
```

```
lr.coef_, lr.intercept_
##lr.coef_=기울기 lr.intercept_=절편
```

```
(array([[2.]]), array([1.]))
```

```
print ("기울기: ", lr.coef_[0][0])
print ("y절편: ", lr.intercept_[0])
```

```
기울기:  1.9999999999999998
y절편:  0.9999999999999999
```

배경지식

reshape()이라는 함수는 배열의 차수를 수정해주는 함수이다.

- ex) reshape(a, b)는 > a행 b열의 원소가 a,b두개 여서 2차원으로 만들어주는 함수이다.
  - if a의 자리에 -1이 들어가면 자동으로 행을 맞춰준다

predict()이라는 함수는 만든 모델로 특정값에 맞는 정답을 예측하는 함수이다.

- 매개변수로는 2차원의 값이 들어가

```
import numpy as np
X_new = np.array(-20).reshape(1, 1)
##-20을 배열로 만들고 reshape해서 2차원으로 만들어주는 코드
X_new
```

```
array([[-20]])
```

```
lr.predict(X_new)
##2차원의 데이터가 들어가야한다. 때문에 reshape해주는 과정이 필요!!
```

```
array([[-39.]])
```

```
X_test = np.arange(11, 16, 1).reshape(-1, 1)
X_test
```

```
array([[11],
       [12],
       [13],
       [14],
       [15]])
```

```
y_pred = lr.predict(X_test)
y_pred
```

```
array([[23.],
       [25.],
       [27.],
       [29.],
       [31.]])
```

분류

분류 모델

- 불연속적인 값을 예측
  - ex)붓꽃 예측, 손글씨 예측

분류 분석

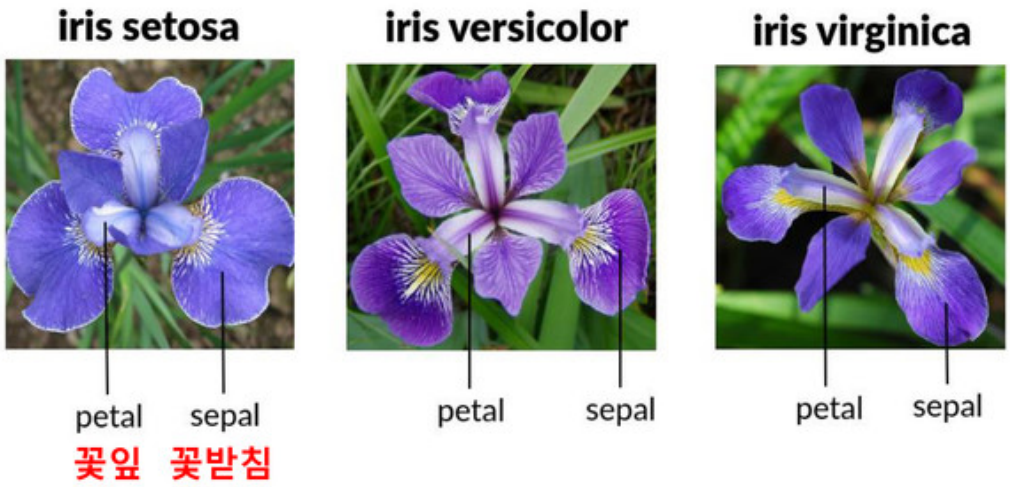
- 다수의 속성또는 변수를 갖는 객체를 사전에 정해진 그룹 또는 범주 중의 하나로 분류하여 분석하는 방법
  - 로지스틱 회귀(Logistic Regression)
    - 이진분류(Binary Classification)
    - 입력: 하나 또는 여러개
    - 출력: 0아니면 1

분류 규현

붓꽃 iris 예측

- 품종 판별
  - of classes = 3 ➤ 3가지로 분류

        - Setosa, Versocolor, Virginica
    - 꽃잎과 꽃바침의 너비와 길이
        - (꽃입,너비) (꽃받침,너비) (꽃입,길이) (꽃받침,길이) ==> 4가지 특징수




```
import pandas as pd
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
#데이터 가져오기
iris
```

```
{'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n*
 'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
```

```
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
```

🖋용어 정리

data ==>실질적인 데이터(문제)
target ==>정답
target_names ==>분류할 꽃 이름
DESCR ==>데이터셋 설명
feature_names ==>특징(꽃받임,꽃입)
filename ==>파일 이름

```
pd.Series(iris)#자료형 판다스로 변환
```

```
data             [[5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [...
target           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
target_names                       [setosa, versicolor, virginica]
DESCR            .. _iris_dataset:\n\nIris plants dataset\n----...
feature_names    [sepal length (cm), sepal width (cm), petal le...
filename         /usr/local/lib/python3.7/dist-packages/sklearn...
dtype: object
```

```
name=iris['target_names']
```

```
# data 속성을 판다스 데이터프레임으로 변환
df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
# 열(column) 이름을 간결하게 변경
df['Target'] = iris['target']
# Target 열 추가
print("데이터프레임의 형태:", df.shape)
df.head(10)
```

```
데이터프레임의 형태: (150, 5)
```

	sepal_length	sepal_width	petal_length	petal_width	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0

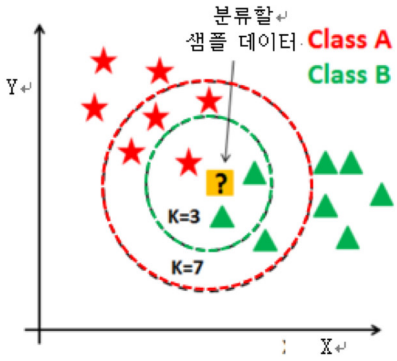
KNN 알고리즘

K 최접근 이웃(KNN) 알고리즘

- 근접 이웃 알고리즘
  - 정답이 있는 지도 학습에 활용, 가장 간단한 알고리즘
  - 회귀와 분류에 모두 사용
- 새로운 입력 자료에 대해 주위에 가장 많은 유형으로 분류

알고리즘 이해

- 기존 훈련 데이터에서 클래스 A에 속하는 것은 붉은 별표, 클래스 B는 녹색 삼각형
- 만일 중아 부근에 위치한 새로운 데이터 "?"는 별표와 삼각형 중 무엇으로 분류해야 할까?



==> k에 따라 정답이 달라질 수 있는 약점이 있다.

학습 데이터와 테스트 데이터 분리

```
from sklearn.model_selection import train_test_split
name=iris['target_names']

X_data = df.loc[:, 'sepal_length':'petal_width']
y_data = df.loc[:, 'Target']

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.2, #20%만 테스트 데이터
                                                    shuffle=True,
                                                    random_state=20)#20개 랜덤 샘플

print(X_train.shape, y_train.shape)#학습 데이터=>120개의 데이터
print(X_test.shape, y_test.shape)#테스트 데이터=>30개의 데이터
```

```
(120, 4) (120,)
(30, 4) (30,)
```

KNN

💡배경지식

KNeighborsClassifier()

- 위 함수는 샘플 데이터 주변으로 몇 개의 데이터가 비교할 것인지 정의해 주는 함수
  - n_neighbors=a를 매개변수로 하여 a 만큼 비교하겠다는 뜻
    accuracy_score()
- 위 함수는 예측값과 정답을 비교하여 모델의 예측률을 보여주는 함수
  - 매개변수로는 실제 정답과 예측값이 들어감

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)#knn 의 k 값을 7로 설정
```

```
knn.fit(X_train, y_train)
#훈련
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

```
y_knn_pred = knn.predict(X_test)
#y_knn_pred=예측에 대한 정답
print("예측값: ", y_knn_pred[:15])
for i in y_knn_pred[:15]:
  print(name[i])
```

```
예측값:  [0 1 1 2 1 1 2 0 2 0 2 1 2 0 0]
setosa
versicolor
versicolor
virginica
versicolor
versicolor
virginica
setosa
virginica
setosa
virginica
versicolor
virginica
setosa
setosa
```

```
#실제 정답
print(y_test[:15].values)
for i in y_test[:15].values:
  print(name[i])
```

```
[0 1 1 2 1 1 2 0 2 0 2 1 2 0 0]
setosa
versicolor
versicolor
virginica
versicolor
versicolor
virginica
setosa
virginica
setosa
virginica
versicolor
virginica
setosa
setosa
```

```
#성능 평가
from sklearn.metrics import accuracy_score
knn_acc = accuracy_score(y_test, y_knn_pred)
print("Accuracy: %.4f" % knn_acc)
```

```
Accuracy: 1.0000
```

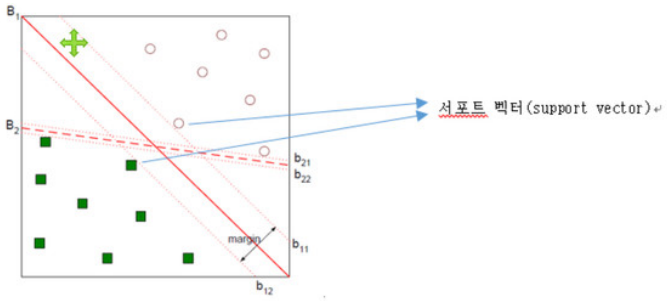
SVM

SVM: Support Vector Machine

- 두 분류 사이의 거리인 마진을 최대화하는 분류 기준 경계인 결정 경계(decision boundary)를 찾는 모델
  - 이진 분류에 주로 사용
  - 딥러닝과 함께 인식률을 매우 좋아 최근까지도 가장 많이 사용되는 알고리즘

알고리즘 이해

- 분류 두점 사이가 가장 넓은 것을 찾고
  그 경계에 있는 서로다른 두점이 각 서포트 벡터로 결정됨



SVM를 활용한 모델학습

```
# 모델 학습
from sklearn.svm import SVC
svc = SVC(kernel='rbf')
svc.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
# 예측
y_svc_pred = svc.predict(X_test)
print("예측값: ", y_svc_pred[:5])
for i in  y_svc_pred[:5]:
  print(name[i])
# 성능 평가
svc_acc = accuracy_score(y_test, y_svc_pred)
print("Accuracy: %.4f" % svc_acc)
```

```
예측값:  [0 1 1 2 1]
setosa
versicolor
versicolor
virginica
```

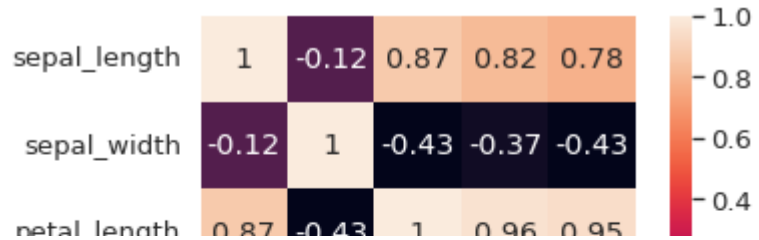
```
versicolor
Accuracy: 1.0000
```

탐색적 데이터 분석(EDA: Exploratory Data Analysis)🌟(중요)

- 정의
  - 수학자 존 튜키가 제안한 데이터 분석 방법으로 통계적 가설 검정 등에 의존한 기존 통계학으로는 새롭게 나오는 많은 양의 데이터의 핵심 의미를 파악하는 데 어려움이 있다고 생각하여 이를 보완한 탐색적 데이터 분석을 도입
  - 탐색적 데이터 분석(EDA: Exploratory Data Analysis)은 데이터를 가지고 유연하게 데이터를 탐색하고, 데이터의 특징과 구조로부터 얻은 정보를 바탕으로 통계모형을 만드는 분석방법입니다. 주로 빅데이터 분석에 사용됩니다.
- 분석방법
  - 데이터에 대한 질문&문제 만들기
  - 데이터를 시각화하고, 변환하고, 모델링하여 그 질문 & 문제에 대한 답을 찾아보기
  - 찾는 과정에서 배운 것들을 토대로 다시 질문을 다듬고 또 다른 질문 & 문제 만들기

[Fig.1] 확증적 데이터 분석(CDA)과 탐색적 데이터 분석(EDA)

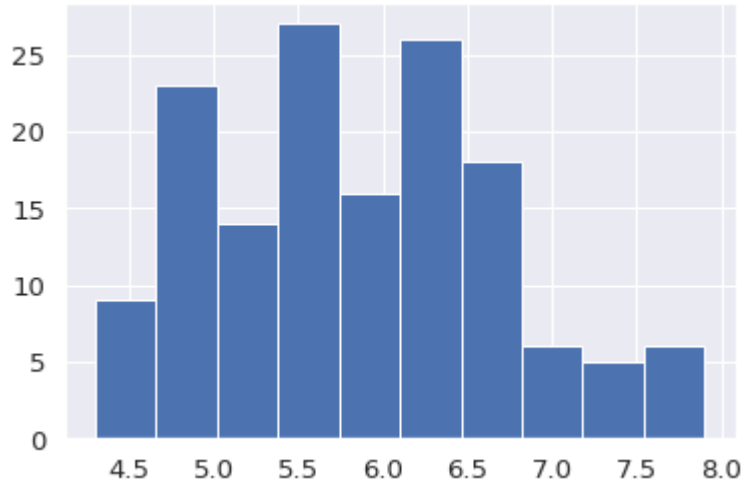
```
# 시각화 라이브러리 설정
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(font_scale=1.2)

# 상관계수 히트맵
sns.heatmap(data=df.corr()
            , square=True#맵 모양
            , annot=True#수표현
            , cbar=True#사이드바
            )
plt.show()
```




```
# Target 값의 분포 - value_counts 함수
df['Target'].value_counts()
```

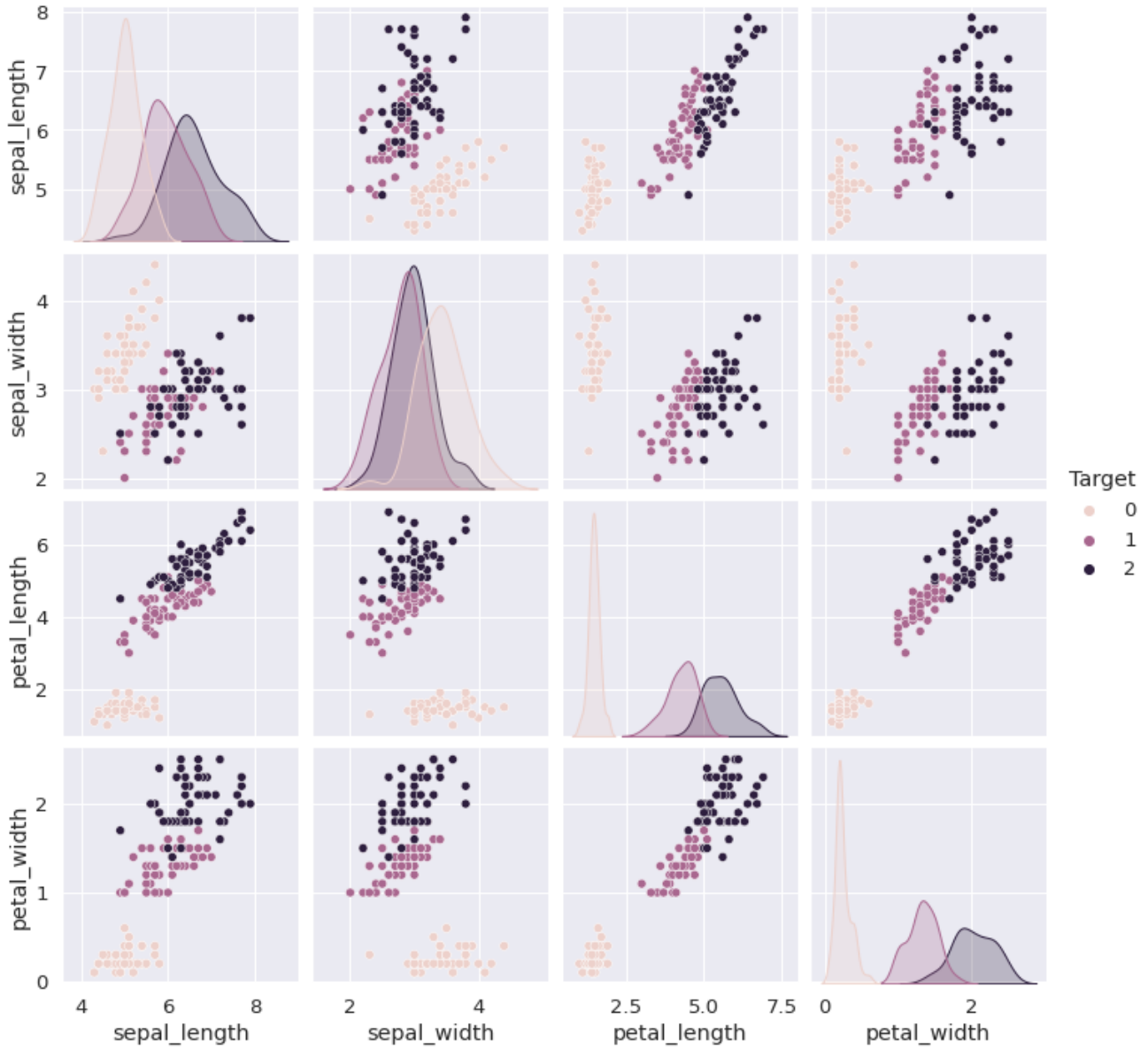
```
2    50
1    50
0    50
Name: Target, dtype: int64
```

```
# sepal_length 값의 분포 - hist 함수
plt.hist(x='sepal_length', data=df)
plt.show()
```




```
# sepal_widgth 값의 분포 - displot 함수 (histogram)
sns.displot(x='sepal_width', kind='hist', data=df)
plt.show()
```


```
# 두 변수 간의 관계
sns.pairplot(df, hue = 'Target', diag_kind = 'kde')
plt.show()
```



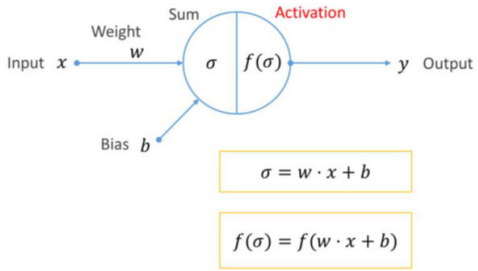
딥러닝

인공신경망

- 생물학적 뉴런에서 모티브한 알고리즘 형태로 수학을 이용하여 인공적으로 뉴런을 만들고 그 뉴런들을 연결시켜 신경망을 이룬것이 인공신경망 이다.

뉴런

- 뉴런은 입력값(input)과 편향(bias)를 받아 뉴런 안에서 w*x+b=σ(w=가중치, x=입력값, b=편향)로 연산되고 활성화 함수를 거쳐 출력값(output)으로 빠져 나간다.

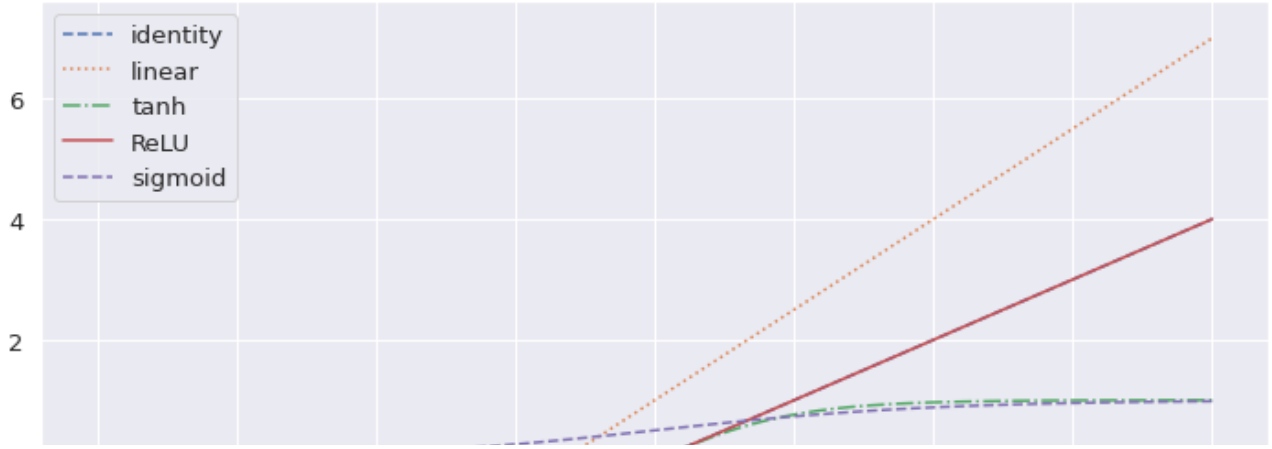


  - 편향
  - 가중치
  - 활성화 함수
    - 정의
      - 입력 신호의 총합을 출력 신호로 변환하는 함수로, 얼마나 출력할지와 기존 퍼셉트론으로 해결할 수 없는 XOR과 같은 non-linear한 문제를 해결할 수 있도록 해준다
        - 퍼셉트론이란?
          - 딥러닝의 기원이 되는 알고리즘으로 가장 간단한 형태의 선형분류기 이다.
    - 함수 종류
      - Sigmoid
        - 출력층 노드에소 0~1사이의 값을 만들고 싶을때 사용
      - Tanh
        - 출력 데이터의 평균이 0으로써 시그모이드보다 대부분의 경우에서 학습이 더 잘 된다.
      - ReLU
        - 대부분의 경우에서 기울기가 0이 되는 것을 막아주기 때문에 학습이 아주 빠르게 잘 되다.
      - Leaky ReLU
        - 입력값이 음수일때 기울기가 0이 아닌 0.01을 갖게 하므로 ReLU보다 학습이 더 잘 된다.
      - Maxout

- ReLU의 장점을 전부 가지고 있으면서, 뉴런이 0을 출력하여 더이상 학습이 안 되는 현상(Dying ReLU)을 완전히 회복한 함수입니다.
  그러나 계산량이 복잡함

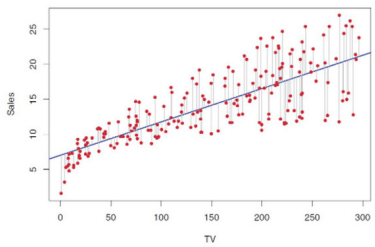
- ELU
  - 가장 최근에 나온 함수이고 Dying ReLU 문제를 해결할 수 있지만 ReLU와 달리 exp함수를 계산하는 비용이 발생한다

```
#활성화 함수 그리기
import numpy as np
import matplotlib.pylab as plt

def identity_func(x): # 항등함수
    return x

def linear_func(x): # 1차함수
    return 1.5 * x + 1 # a기울기(1.5), Y절편b(1) 조정가능

def tanh_func(x): # TanH 함수
    return np.tanh(x)

def relu_func(x): # ReLU(Rectified Linear Unit, 정류된 선형 유닛) 함수
    return np.maximum(0, x)
    #return (x>0)*x # same

def sigm_func(x): # sigmoid 함수
    return 1 / (1 + np.exp(-x))


# 그래프 그리기
plt.figure(figsize=(12, 8))
x = np.linspace(-4, 4, 100)

plt.plot(x, identity_func(x), linestyle='--', label="identity")
plt.plot(x, linear_func(x), linestyle=':', label="linear")
plt.plot(x, tanh_func(x), linestyle='-.', label="tanh")
plt.plot(x, relu_func(x), linestyle='-', label="ReLU")
plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")
plt.legend(loc='upper left');
```



🌟주요 용어 정리🌟

- 가설(Hypothesis)
  - 가중치(weight)와 편향(bias)
  - 기울기와 절편
  - 선형 회귀에서 해야할 일은 결국 적절한 W(가중치), b(편향)를 찹아내는 일
    - 딥러닝 알고리즘이 하는 것이 바로 적절한 W와 b를 찾아내는 일
- 손실 함수(Loss function)
  - 실제 값과 예측 값에 대한 오차에 대한 식(예측 값의 오차를 줄이는 일에 최적화 된 식)
  - W와 b의 값을 찾아내기 위해서 오차의 크기를 측정할 방법이 필요
    - MSE
      - 오차는 실제 데이터(빨간 점)와 예측 선(파란 선)의 차이의 제곱의 합

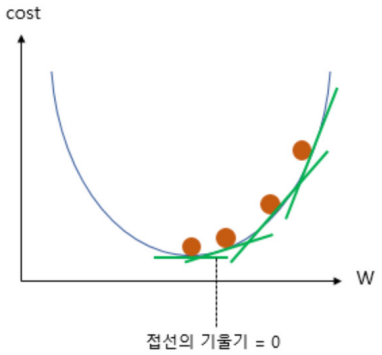


      - 평균 제곱 오차를 W와 b에 의한 비용 함수(Cost funtion)로 재정의
        - 모든 점 들과의 오차가 클수록 평균 제곱 오차는 커지며, 오차가 작아질수록 평균 제곱 오차는 작아짐⭐
      - 평균 제곱 오차
        - cost(W,b)를 최소가 되게 만드는 W와 b를 구하면 결과적으로 y와 x의 관계를 가장 잘 나타내는 직선을 그릴 수 있게 됨

$$W,b \rightarrow \text{minimize } cost(W,b)$$

- 경사 하강법(Gradient Descent)
  - 내리막 경사 따라 가기
  - 접선의 기울기

        - 맨 아래의 볼록한 부분에서는 결국 적선의 기울기가 0
          = 결국 가중치에 대한 손실 값이 최솟값
        - 경사 하강법의 아이디어
            - 비용 함수(Cost funtion)를 미분하여 현재 W에서의 접선의 기울기를 구하고
              접선의 기울기가 낮은 방향으로 W의 값을 변경
              이 과정 반복하여 가중치에 대한 비용 함수가 최솟값을 찾는것



- 학습률(learning rate)
  = 경사 하강법에서 이동 크기
    - 대표적인 하이퍼패러미터
        - 학습에 의해 결정되는 값과 프로그래머가 결정하는 값
    - 얼마나 큰 폭으로 이동할 지를 결정
      (평균적으로 0.001~0.1사이의 값을 사용)
        - 학습률(이동 폭)이 무작정 크면
            - W의 값이 발산하는 상황
        - 학습률(이동 폭)이 지나치게 작으면
            - 학습 속도가 느려짐
- 순전파와 역전파
    - 순전파 -> 역전파 -> 가중치 수정 -> 순전파를 계속 반복해 나가면서 오차 값이 적어지도록 할 수 있다.
        - 엄청난 처리 속도의 증가

딥러닝 구현 과정

1. 모델 생성
2. 훈련방법 설정
3. 훈련
4. 성능평가
5. 결과

🌟사용된 데이터셋은 mnist(손글씨 데이터셋)가 사용 되었습니다.🌟

손글씨 예측 모델 구현하기(분류 분석)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환==>정규화
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

분류 모델 생성(Sequential)

- tensorflow에 있는 keras라이브러리를 활용하여 모델의 층을 구성
- 손글씨 예측 모델은 분류 모델이다.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

➕ input_shape=(28, 28)은 손글씨 데이터가 28x28로 구성되어 있어서이다.

Flatten()

- 함수를 이용하여 입력층을 1차원 배열로 평탄화를 시킨다.

```
import numpy as np
x = np.array([2, 3, 254, 5, 6, 3])
x = x / 255.0
print(x)
x = x.reshape(2, 3)
print(x)
x = x.flatten()
print(x)
```

```
[0.00784314 0.01176471 0.99607843 0.01960784 0.02352941 0.01176471]
[[0.00784314 0.01176471 0.99607843]
```

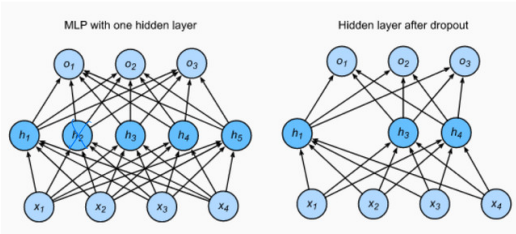
```
[0.01960784 0.02352941 0.01176471]]
[0.00784314 0.01176471 0.99607843 0.01960784 0.02352941 0.01176471]
```

Dense()

- Dense()함수는 핵심 층을 구성하는 중간 층, 신경망이라고 말할 수 있는 층
  - +)매개변수로는 뉴련의 수와 활성화 함수가 들어간다
    - softmax = 시그모이드 함수로부터 유도된 함수 (결과값을 확률값으로 변환해주고 각 값의 차이를 증폭시키는 효과가 있음)

Dropout()

- Dropout()함수는 훈련중 랜덤하게 뉴런을 끊음으로써, 모델을 단순하게 하여 학습률을 증가시키는 기법
  - 매개변수로는 몇 퍼센트를 드롭 시킬지 실수 값이 들어간다
- 오버피팅(overfitting) 문제를 해결하는 정규화(regularization) 목적을 위해서 필요
  - 학습 데이터에 지나치게 집중해 실제 Test에서는 결과가 더 나쁘게 나오는 현상
- h2와 h5를 0으로 지정하여 사용되지 않게 조정



💡배경지식

파라미터(Param)란?

- 전 뉴런과 다음 층의 뉴련이 연결되어 데이터가 지나가는 길

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.summary()
##모델 요약 표시
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0
```

```
 dense (Dense)               (None, 128)               100480

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

훈련방법 설정(compile)

훈련에 사용할 옵티마이저(optimizer)와 손실 함수(loss) 등을 선택

- 옵티마이저(optimizer)
  - 입력된 테이터와 손실 함수를 기반으로 모델(w와 b)을 업데이트하는 메커니즘
- 손실 함수(loss)
  - 훈련 데이터에서 신경망의 성능을 측정하는 방법
  - 모델이 옳은 방향으로 학습될 수 있도록 도와 주는 기준 값
- 출력될 정보(metrics)
  - 보여질 지표값을 설정
    - accuracy= 정확률, mse= 평균 제곱 오차

훈련에 사용할 옵티마이저(optimizer)와 손실 함수(loss), 출력정보를 모델에 설정

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
              # metrics=['accuracy', 'mse'])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
##정확률만  출력
```

훈련(fit)

- x_train
  - 훈련용 문제 데이터
- y_train

- 훈련용 문제 정답 데이터

- epochs=a(자연수)
  - 모든 데이터를 a번의 사이클로 반복 훈련

```
model.fit(x_train, y_train, epochs=20)
```

```
model.fit(x_train, y_train, epochs=6)
```

```
Epoch 1/6
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2994 - accu
Epoch 2/6
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1457 - accu
Epoch 3/6
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1069 - accu
Epoch 4/6
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0889 - accu
Epoch 5/6
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0740 - accu
Epoch 6/6
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0665 - accu
<keras.callbacks.History at 0x7efea536d8d0>
```

성능평가(evaluate)

- x_test
  - 테스트 데이터의 문제
- y_test
  - 테스트 데이터의 정답

모델을 테스트 데이터로 평가

```
model.evaluate(x_test, y_test)
```

```
model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0743 - accura
[0.07430596649646759, 0.9768000245094299]
```

최종 결과

MNIST손글씨 예측과 오류 확인

배경지식

1. numpy 라이브러리에 있는 argmax()함수는 배열안 가장 큰 값의 인덱스를 반환 해준다.

- axis=1 > 가로축
  axis=0 > 세로축

```
print(tf.argmax([5, 4, 10, 1, 2]).numpy())
print(tf.argmax([3, 1, 4, 9, 6, 7, 2]))
print(tf.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=
2
3
[1,0,2]
```

2. 예측한 결과물은 softmax함수를 거쳐 10개의 확률 값이 (1,10)의 이차원 배열 형태로 반환된다. 때문에 배열을 모두 더하면 1에 가까운 수가 되고 배열에서 가장 큰 값의 인덱스가 예측한 정답이 된다.

3. 원핫 인코딩(One Hot Encoding)란

- 고유 값에 해당하는 자리에만 1을 표시하고 나머지 칼럼에는 0을 표시하는 방법이다.
  - ex)
    [1,0,0] = 0
    [0,0,1] = 2
    [0,1,0] = 1

```
import numpy as np
pred_result = model.predict(x_test[0:3])
print(pred_result.shape)
for i in {0,1,2}:
  print(pred_result[i])
  one_pred = pred_result[i]
  one = np.argmax(one_pred)
```

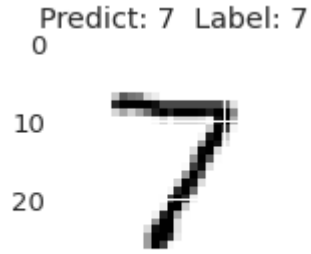
```
  print(one)##예측답
  print(y_test[i])##실제답
```

```
(3, 10)
[1.3415365e-07 3.3426527e-08 1.2939747e-06 9.8139230e-05 1.1791214e-10
 4.3903487e-08 5.0995975e-13 9.9986911e-01 5.7006488e-08 3.1176314e-05]
7
7
[1.9448476e-07 9.5279829e-05 9.9989820e-01 6.0011820e-07 7.9682998e-15
 2.7740333e-07 2.1167021e-07 4.9184593e-16 5.1057987e-06 1.3151846e-14]
2
2
[1.70164014e-07 9.98822749e-01 1.80069314e-04 4.39490350e-06
 1.17138035e-04 4.24924292e-06 8.10183246e-06 6.45730586e-04
 2.17022010e-04 2.71168233e-07]
1
1
```

```
print(one_pred)
##모든 확률값 합산
print(one_pred.sum())
```

```
[1.70164014e-07 9.98822749e-01 1.80069314e-04 4.39490350e-06
 1.17138035e-04 4.24924292e-06 8.10183246e-06 6.45730586e-04
 2.17022010e-04 2.71168233e-07]
0.99999994
```

```
import matplotlib.pyplot as plt
import numpy as np
for i in {0,1,2}:
  one_pred = pred_result[i]
  one = np.argmax(one_pred)
  plt.figure(figsize=(5, 2))
  tmp = "Predict: " + str(one) + "  Label: " + str(y_test[i])
  plt.title(tmp)
  _ = plt.imshow(x_test[i], cmap='Greys')
```

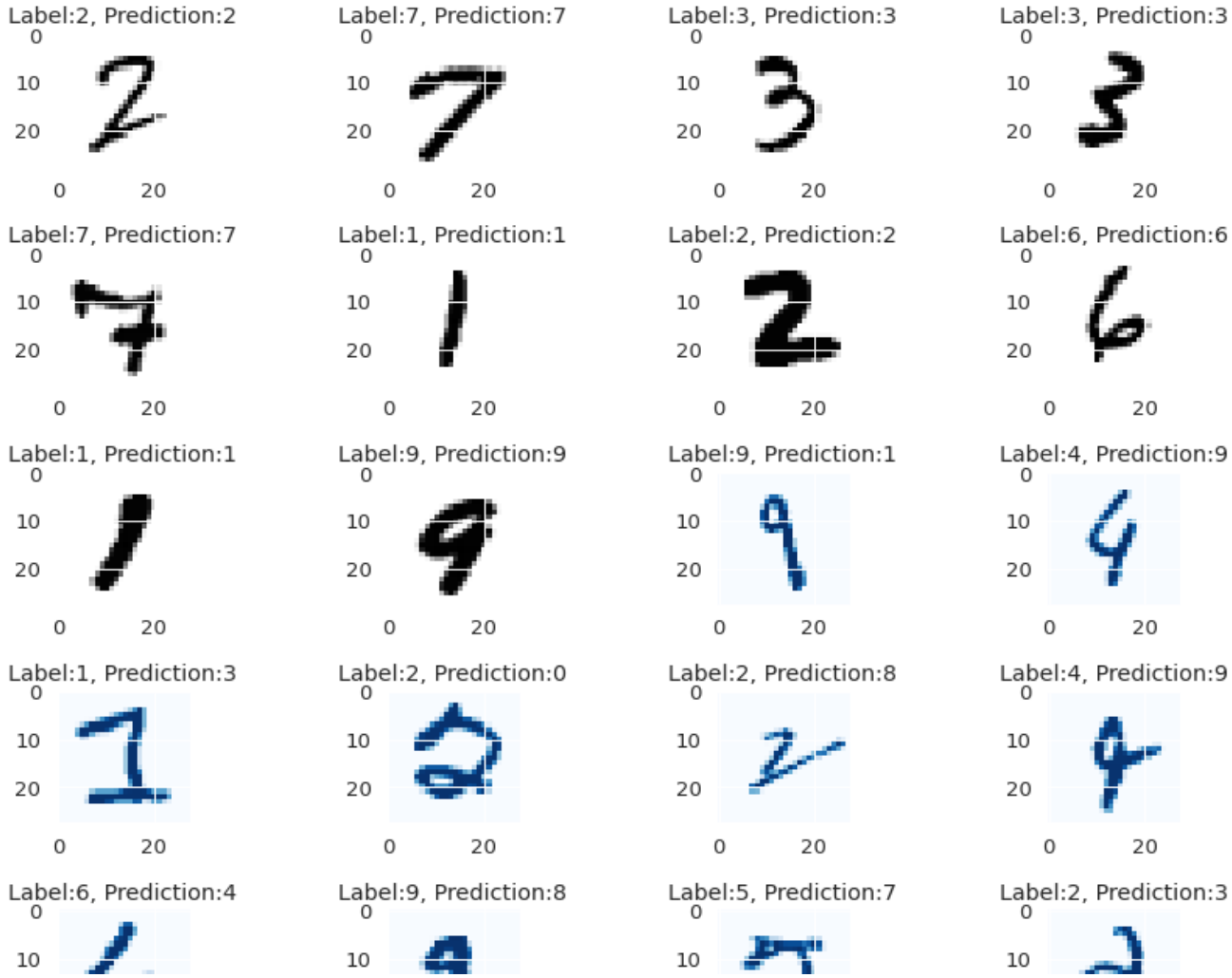


✚플러스 코딩✚

랜덤한 10개 예측와 틀린 예측 10개 출력하기

```
from random import sample
import numpy as np

# 예측한 softmax의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)
# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)

#랜덤하게 20개의 훈련용 자료를 예측 값과 정답, 그림을 그려 보자.
nrows, ncols = 5, 4
samples = sorted(sample(range(len(x_test)), nrows * 2)) # 랜덤한 10개 출력할 첨자 선정

mispred = [n for n in range(0, len(y_test)) if pred_labels[n] != y_test[n]]#예측이 틀
print('정답이 틀린 수', len(mispred))
mispred = sample(mispred, 10)# 예측이 틀린 10개 출력할 첨자 선정
samples=np.concatenate([samples, mispred]) #두 배열 합치기
# 랜덤한 10개 예측와 틀린 예측 10개 출력하기
count = 0
plt.figure(figsize=(12,10))
for n in samples:
    count += 1
    plt.subplot(nrows, ncols, count)
    # 예측이 틀린 것은 파란색으로 그리기
    cmap = 'Greys' if ( pred_labels[n] == y_test[n]) else 'Blues'
    plt.imshow(x_test[n].reshape(28, 28), cmap=cmap, interpolation='nearest')
    tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])
    plt.title(tmp)

plt.tight_layout()
plt.show()
```

```
정답이 틀린 수 232
```



논리 게이트 AND, RO, XRO 신경망 구현 해보기

- AND 게이트
  - 두 값중 하나라도 0이면 출력값이 0
  - AND 게이트 모델 구현 및 훈련

```
# tf.keras 를 이용한 AND 네트워크 계산
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [0], [0], [0]])
model = tf.keras.Sequential([
tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=
])
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.3
model.summary()
history = model.fit(x, y, epochs=100, batch_size=1)
```

- OR 게이트

- 두 값중 하나라도 1이면 출력값이 1
- OR 게이트 모델 구현 및 훈련

```
# tf.keras 를 이용한 OR 네트워크 계산
  import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [1], [1], [0]])
model = tf.keras.Sequential([
tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=
])
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='ms
model.summary()
history = model.fit(x, y, epochs=500, batch_size=1)
```

- XOR 게이트
  - 두 입력값이 같으면 0의 출력값, 다르면 1의 출력값 1
    - XOR 게이트는 하나의 퍼셉트론으로는 구현이 불가능
  - XOR 게이트 모델 구현 및 훈련

```
# 3.27 tf.keras 를 이용한 XOR 네트워크 계산
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])
model = tf.keras.Sequential([
tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=
tf.keras.layers.Dense(units=1, activation='sigmoid')
])
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='ms
model.summary()
history = model.fit(x, y, epochs=500, batch_size=1)
```

AND 구현하기

```
# tf.keras 를 이용한 AND 네트워크 계산
import numpy as np
import tensorflow as tf
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [0], [0], [0]])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=(2,)),
])

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.3), loss='mse')
model.summary()
history = model.fit(x, y, epochs=500, batch_size=1)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_2 (Dense)             (None, 1)                 3         

=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/500
4/4 [==============================] - 0s 3ms/step - loss: 0.2884
Epoch 2/500
4/4 [==============================] - 0s 3ms/step - loss: 0.2764
Epoch 3/500
4/4 [==============================] - 0s 3ms/step - loss: 0.2681
Epoch 4/500
4/4 [==============================] - 0s 3ms/step - loss: 0.2594
Epoch 5/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2513
Epoch 6/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2436
Epoch 7/500
4/4 [==============================] - 0s 2ms/step - loss: 0.2374
Epoch 8/500
4/4 [==============================] - 0s 2ms/step - loss: 0.2318
Epoch 9/500
4/4 [==============================] - 0s 2ms/step - loss: 0.2260
Epoch 10/500
4/4 [==============================] - 0s 2ms/step - loss: 0.2197
Epoch 11/500
4/4 [==============================] - 0s 3ms/step - loss: 0.2154
Epoch 12/500
4/4 [==============================] - 0s 2ms/step - loss: 0.2098
Epoch 13/500
4/4 [==============================] - 0s 3ms/step - loss: 0.2054
Epoch 14/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1999
Epoch 15/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1953
Epoch 16/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1914
Epoch 17/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1866
Epoch 18/500
4/4 [==============================] - 0s 4ms/step - loss: 0.1824
Epoch 19/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1780
Epoch 20/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1735
```

```
Epoch 21/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1694
Epoch 22/500
4/4 [==============================] - 0s 5ms/step - loss: 0.1654
Epoch 23/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1617
Epoch 24/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1582
```

```
model.predict(x)
```

```
array([[0.8714159 ],
       [0.10794088],
       [0.10799989],
       [0.00215706]], dtype=float32)
```

AND 게이트 가중치와 편향 출력해보기

```
print(model.weights[0].numpy().reshape(-1,))#가중치
print(model.weights[1].numpy())#편향
```

```
[4.0248723 4.0254846]
[-6.136821]
```

OR 구현하기

```
# tf.keras 를 이용한 OR 네트워크 계산
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=(2,)),
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()

history = model.fit(x, y, epochs=500, batch_size=1)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 1)                 3

=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/500
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:
  super(SGD, self).__init__(name, **kwargs)
4/4 [==============================] - 0s 4ms/step - loss: 0.2102
```

```
Epoch 2/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1889
Epoch 3/500
4/4 [==============================] - 0s 4ms/step - loss: 0.1738
Epoch 4/500
4/4 [==============================] - 0s 4ms/step - loss: 0.1622
Epoch 5/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1532
Epoch 6/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1459
Epoch 7/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1398
Epoch 8/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1348
Epoch 9/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1305
Epoch 10/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1265
Epoch 11/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1230
Epoch 12/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1199
Epoch 13/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1168
Epoch 14/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1138
Epoch 15/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1112
Epoch 16/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1085
Epoch 17/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1060
Epoch 18/500
4/4 [==============================] - 0s 4ms/step - loss: 0.1036
Epoch 19/500
4/4 [==============================] - 0s 4ms/step - loss: 0.1013
Epoch 20/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0990
Epoch 21/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0968
Epoch 22/500
4/4 [==============================] - 0s 4ms/step - loss: 0.0947
Epoch 23/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0927
```

```
model.predict(x)
```

```
array([[0.9993999 ],
       [0.9338347 ],
       [0.9340685 ],
       [0.10719743]], dtype=float32)
```

OR 게이트 가중치와 편향 출력해보기

```
print(model.weights[0].numpy().reshape(-1,))#가중치
print(model.weights[1].numpy())#편향
```

```
[4.7668357 4.7706265]
```

```
[-2.1196935]
```

XOR 구현하기

```
# 3.27 tf.keras 를 이용한 XOR 네트워크 계산
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')

model.summary()
history = model.fit(x, y, epochs=1000, batch_size=1)
```

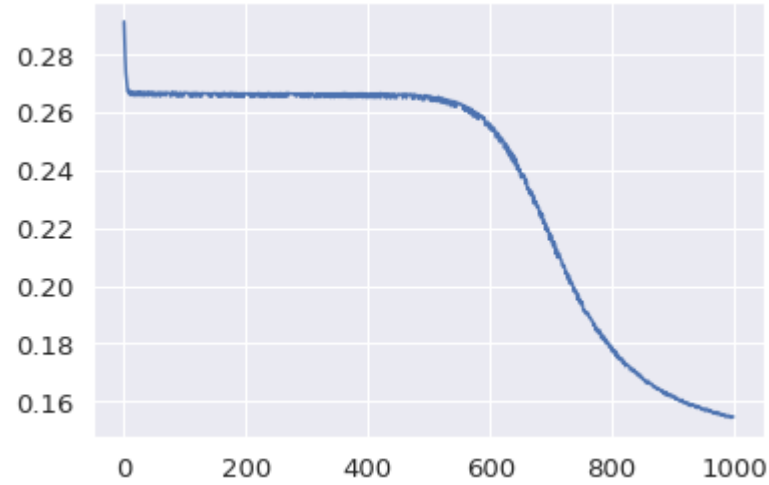
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 2)                 6

 dense_5 (Dense)             (None, 1)                 3

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
Epoch 1/1000
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:
  super(SGD, self).__init__(name, **kwargs)
4/4 [==============================] - 0s 3ms/step - loss: 0.2914
Epoch 2/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2840
Epoch 3/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2784
Epoch 4/1000
4/4 [==============================] - 0s 4ms/step - loss: 0.2736
Epoch 5/1000
4/4 [==============================] - 0s 5ms/step - loss: 0.2713
Epoch 6/1000
4/4 [==============================] - 0s 2ms/step - loss: 0.2696
Epoch 7/1000
4/4 [==============================] - 0s 2ms/step - loss: 0.2674
Epoch 8/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2670
Epoch 9/1000
4/4 [==============================] - 0s 2ms/step - loss: 0.2677
Epoch 10/1000
4/4 [==============================] - 0s 2ms/step - loss: 0.2663
Epoch 11/1000
4/4 [==============================] - 0s 2ms/step - loss: 0.2671
Epoch 12/1000
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.2671
Epoch 13/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2659
Epoch 14/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2669
Epoch 15/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2670
Epoch 16/1000
4/4 [==============================] - 0s 2ms/step - loss: 0.2669
Epoch 17/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2659
Epoch 18/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2658
Epoch 19/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2669
Epoch 20/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2669
Epoch 21/1000
4/4 [==============================] - 0s 3ms/step - loss: 0.2659
Epoch 22/1000
4/4 [==============================] - 0s 2ms/step - loss: 0.2659
```

XOR 손실 값 그래프

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7efea01e6990>]
```




```
model.predict(x)
```

```
array([[0.49879378],
       [0.8746069 ],
       [0.47134516],
       [0.11011434]], dtype=float32)
```

```
print(model.weights[0].numpy().reshape(-1,))#입력층과 은닉층 사이의 가중치
print(model.weights[1].numpy())#입력층과 은닉층 사이의 편향
print(model.weights[2].numpy().reshape(-1,))#은닉층과 출력층 사이의 가중치
print(model.weights[3].numpy())#은닉층과 출력층 사이의 편향
```

```
[-2.4763093  3.2571175  4.6502886  4.760085 ]
```

```
[ 1.8537513 -0.5298243]
[-3.455219   3.9629402]
[-0.5708039]
```

회귀 모델을 이용한 딥러닝 구현

- 단순 선형 회귀 분석
  - 입력값 하나 그에 대한 출력값 하나
    - 선형 회귀
- 다중 선형 회귀 분석
  - 입력값 여러개 그에 대한 출력값 하나
    - 주택가격 예측

선형 회귀 케라스를 이용하여 구현

- 하나의 Dense층
  - 입/출력 1차원
- 활성화 함수 linear(항등함수)
  - 디폴트 값, 입력 뉴런과 가중치로 계산된 결과값이 그대로 출력
- 옵티마이저 SGD(확률적 경사하강법)
  - 경사하강법의 계산량을 줄이기 위해서 확률적 방법으로 경사하강법을 사용
    - 전체를 계산하지 않고 확률적으로 일부 샘플로 계산

+)mae는 평균 절대 오차

- 모든 예측과 정답과의 오차 합의 평균

```
import tensorflow as tf

# ① 문제와 정답 데이터 지정
#x_train = [1, 2, 3, 4]
#y_train = [3, 5, 7, 9]
x_train =[4,  0, 3, 1]
y_train = [9,  1, 7, 3]

# ② 모델 구성(생성)
model = tf.keras.models.Sequential([
    #                   출력, 입력=(*, 1)            그대로 출력
    tf.keras.layers.Dense(1, input_shape=(1, ), activation='linear')
    #Dense(1, input_dim=1)
])
# ③ 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력 정보를 지정
```

```
# Mean Absolute Error, Mean Squared Error
model.compile(optimizer='SGD', loss='mse', metrics=['mae', 'mse'])
#model.compile(optimizer='SGD', loss='mse', metrics=['mae'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #   
=================================================================
 dense_6 (Dense)              (None, 1)                 2         

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(x_train, y_train, epochs=500)
```

```
Epoch 1/500
1/1 [==============================] - 0s 330ms/step - loss: 46.7885 - mae: 5.
Epoch 2/500
1/1 [==============================] - 0s 5ms/step - loss: 34.3674 - mae: 4.91
Epoch 3/500
1/1 [==============================] - 0s 5ms/step - loss: 25.2449 - mae: 4.22
Epoch 4/500
1/1 [==============================] - 0s 4ms/step - loss: 18.5449 - mae: 3.63
Epoch 5/500
1/1 [==============================] - 0s 7ms/step - loss: 13.6241 - mae: 3.12
Epoch 6/500
1/1 [==============================] - 0s 7ms/step - loss: 10.0101 - mae: 2.68
Epoch 7/500
1/1 [==============================] - 0s 5ms/step - loss: 7.3557 - mae: 2.310
Epoch 8/500
1/1 [==============================] - 0s 6ms/step - loss: 5.4062 - mae: 1.989
Epoch 9/500
```

```
#손실(loss)그래프
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['mae'], label='mae')

plt.legend(loc='best')
plt.xlabel('epoch')
plt.ylabel('loss')
```

```
1/1 [==============================] - 0s 10ms/step - loss: 0.8601 - mae: 0.82
```

```
# ⑤ 테스트 데이터로 성능 평가
x_test = [1.2, 2.3, 3.4, 4.5]
y_test = [3.4, 5.6, 7.8, 10.0]

print('손실:', model.evaluate(x_test, y_test))
```

```
model.predict(x_test).flatten()
```

```
print(model.weights[0].numpy().reshape(-1,))#가중치
print(model.weights[1].numpy())#편향
```

보스턴 주택 가격 예측

- 주요 활성화 함수
  - ReLU
  - Sigmoid
  - Tanh

```
# 활성화 함수
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import math
def sigmoid(x):
```

```
    return 1 / (1 + math.exp(-x))

x = np.arange(-5, 5, 0.01)#x축 좌표 범위-5~5까지 0.01간격으로 그래프를 그린다
sigmoid_x = [sigmoid(z) for z in x]
tanh_x = [math.tanh(z) for z in x]
relu = [0 if z < 0 else z for z in x]#z가 0이하이면 0, 0이상이면  z=x 일차함수

plt.figure(figsize=(8, 6))

plt.axhline(0, color='gray')
plt.axvline(0, color='gray')
plt.plot(x, sigmoid_x, 'b-', label='sigmoid')
plt.plot(x, tanh_x, 'r--', label='tanh')
plt.plot(x, relu, 'g.', label='relu')
plt.legend()
plt.show()
```

1978년 보스톤 지역 주택 가격 데이터 불러오기

- 마지막 변수 주택가격(중앙값)을 나머지 13개의 변수로 예측하는 문제
- 특징(속성)
  - CRIM: 자치시(town) 별 1인당 범죄율
  - ZN: 25,000 평방피트를 초과하는 거주지역의 비율
  - INDUS: 비소매상업지역이 점유하고 있는 토지의 비율
  - CHAS: 찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
  - NOX: 10ppm 당 농축 일산화질소
  - RM:주택1가구당평균방의개수
  - AGE: 1940년 이전에 건축된 소유주택의 비율
  - DIS: 5개의 보스턴 직업센터까지의 접근성 지수
  - RAD: 방사형 도로까지의 접근성 지수
  - TAX: 10,000 달러 당 재산세율
  - PTRATIO: 자치시(town)별 학생/교사 비율
  - B: 1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함.
  - LSTAT: 모집단의 하위계층의 비율(%)
- 정답
  - MEDV: 주택가격(중앙값) (단위: $1,000)

```
1/1 [==============================] - 0s 7ms/step - loss: 0.0079 - mae: 0.071
```

```
from tensorflow.keras.datasets import boston_housing
(train_X, train_Y), (test_X, test_Y) = boston_housing.load_data()#8:2 비율로 훈련데이터

print(train_X.shape, test_X.shape)#8:2 비율
print(train_X[0])#13가지 데이터
print(train_Y[0])#금액($1,000 단위)
```

✚플러스 코딩✚

자료의 표준화

- 표준화의 필요
  - 특성의 단위가 다르기 때문에 단위를 맞춰주기 위해
  - 표준화(standardization)가 학습 효율을 높혀줌
- 표준화 방법
  - 학습데이터
    - (train_x - 학습데이터 평균)/학습 데이터 표준편차
  - 테스트 데이터
    - (test_x - 학습데이터 평균)/학습 데이터 표쥰편차

```
1/1 [==============================] - 0s 5ms/step - loss: 0.0064 - mae: 0.064
```

```
x_mean = train_X.mean(axis=0)#평균
x_std = train_X.std(axis=0)#표준편차
train_X -= x_mean
train_X /= x_std
test_X -= x_mean
test_X /= x_std

y_mean = train_Y.mean(axis=0)
y_std = train_Y.std(axis=0)
train_Y -= y_mean
train_Y /= y_std
test_Y -= y_mean
test_Y /= y_std

print(train_X[0])
print(train_Y[0])#각 데이터의 단위가 같아짐
```

```
1/1 [==============================] - 0s 7ms/step - loss: 0.0056 - mae: 0.060
```

회귀 모델 생성

- 주택가격 예측 모델은 회귀 모델이다.

```
1/1 [==============================] - 0s 5ms/step - loss: 0.0053 - mae: 0.058
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),#따로 Flat
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dense(units=1)#활성화 함수가 정의되지 않으면 디폴트값 = liner로 정해진다.
])
#입력층->3층 구조의 은닉층(52개의 뉴런->39개의 뉴런->26개의 뉴런으로 구성)->출력층으로 구성

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')#학습률 0.07로
#model.compile(optimizer=tf.keras.optimizers.Adam(), loss='mse')

model.summary()
```

```
Epoch 91/500
```

학습

- 배치 사이즈와 검증 데이터
  - 배치 사이즈(batch_size)
    - 훈련에서 가중치와 편향의 패러미터를 수정하는 데이터 단위수
      - 10개의 데이터에서 배치 사이즈를 3으로 하면
        10/3의 몫값에 나머지가 있으면 +1 한것이 가중치와 편향의 수정 횟수 이다.
        10/3 +1 = 4
  - 검증 데이터
    - 모델의 성능을 평가하기 위해 훈련용 데이터에서 약간의 데이터를 떼어 만든것
      - 테스트 데이터와 검증 데이터를 따로 두는 이유
        - 검증(Validation)은 모델의 성능을 평가하고, 그 결과를 토대로 모델을 수정하는 작업을 진행한다.
        - 평가(test)는 최종적으로 해당 모델의 성능을 확인하는 단계이다.

```
1/1 [==============================] - 0s 6ms/step - loss: 0.0040 - mae: 0.051
```

```
history = model.fit(train_X, train_Y, epochs=50, batch_size=25, validation_split=0.
#배치 사이즈 = 25 ,검증 데이터 25%
```

```
1/1 [==============================] - 0s 5ms/step - loss: 0.0039 - mae: 0.050
```

손실함수 시각화 및 평가

- loss
  - 손실
    - 검증 손실이 적을수록 평가의 손실도 적음
- val_loss
  - 검증 손실
    - 일반적으로 loss 보다 높음
    - 항상 감소하지 않음

```
Epoch 114/500
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

```
Epoch 118/500
```

```
model.evaluate(test_X, test_Y)
```

```
1/1 [==============================] - 0s 7ms/step - loss: 0.0032 - mae: 0.043
Epoch 120/500
```

예측 결과 시각화

- 테스트 데이터의 예측과 실제 주택 가격 비교
  - 각 점들이 점선의 대각선에 있어야 좋은 예측

```
1/1 [==============================] - 0s 8ms/step - loss: 0.0031 - mae: 0.044
```

```
import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(8,8))
plt.plot(test_Y, pred_Y, 'b.')
# x, y축 범위 설정
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

# y=x에 해당하는 대각선
plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".5")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```

```
Epoch 133/500
```

➕플러스 코딩➕

예측 성적을 높히는 두가지 방법

- 중간에 학습 중단
  - 검증 손실(val_loss)이 적을수록 테스트 평가의 손실도 적음
  - 감증 데이터에 대한 성적이 좋도록 유도
    - 과적합에 의해 검증 손실이 증가하면 학습을 중단되도록 지정
    - 함수 callbacks 사용
      - 일찍 멈춤 기능
        - monitor : 지켜볼 기준 값(ex.val_loss : 지켜볼 값이 검증 손실)
        - patience : epochs를 실행중에 기준 값(patience)동안 최고 기록을 갱신하지 못하면(손실값이 낮아지지 않으면) 학습 중단
- Dropout

```
3240     except  core. NotOkStatusException as e:
```

중간에 학습 중단

- epochs를 실행중에 기준 값(patience)동안 최고 기록을 갱신하지 못하면(손실값이 낮아지지 않으면) 학습 중단

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dense(units=1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')
```

```
history = model.fit(train_X, train_Y, epochs=100, batch_size=32, validation_split=0
                    callbacks=[tf.keras.callbacks.EarlyStopping(patience=15, monit
model.evaluate(test_X, test_Y)
```

손실함수와 예측 시각화

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

```
import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(8,8))
plt.plot(test_Y, pred_Y, 'b.')
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".3")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```

Dropout

- 랜덤하게 뉴런을 끊음으로써, 모델을 단순하게 하여 학습률을 증가시키는 기법

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
    tf.keras.layers.Dropout(.1),
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dropout(.1),
    tf.keras.layers.Dense(units=1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')

# history = model.fit(train_X, train_Y, epochs=50, batch_size=32, validation_split=
#                     callbacks=[tf.keras.callbacks.EarlyStopping(patience=8, monit
history = model.fit(train_X, train_Y, epochs=35, batch_size=32, validation_split=0.


model.evaluate(test_X, test_Y)
```

손실함수와 예측 시각화

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

```
import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(8,8))
plt.plot(test_Y, pred_Y, 'r.')
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".3")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```

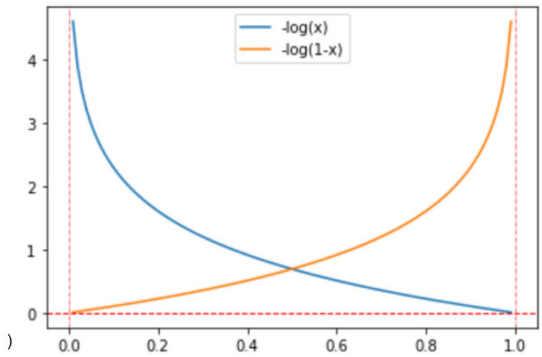
💡배경지식

크로스 엔트로피

- 실제 데이터의 결과 값인 y
  - y=1일 때
    - 예측 값이 1에 가까워질수록
      - cost function의 값은 작아져야함
    - 예측 값이 0에 가까워질수록
      - cost function의 값이 무한대로 증가하게 됨
      - 예측이 틀렸다는 것을 보여주어야 함
  - y=0일 때
    - 예측이 0으로 맞게 되면
      - cost function은 매우 작은 값을 가지고
    - 예측이 1로 맞게 되어 예측이 실패 할 경우
      - cost 값이 무한대로 증가
      - 틀렸다는 것을 알 수 있게 해야 함

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

alpha = 0.1e-1
```

```
    x = np.linspace(0+alpha, 1-alpha, 100)
    y1 = -np.log(x)
    y2 = -np.log(1-x)

    plt.axhline(y=0, color='r', linestyle='--',linewidth=1)
    plt.axvline(x=1, color='r', linestyle='-.',linewidth=.5)
    plt.axvline(x=0, color='r', linestyle='-.',linewidth=.5)
    plt.plot(x, y1, label='-log(x)')
    plt.plot(x, y2, label='-log(1-x)')
    plt.legend(loc='best')
    plt.show()
```



)

one_hot()과 keras.utils.to_categorical()함수

- 두 함수 모두 일반 값을 원핫으로 변환 해주는 함수이다.
- 두 함수에 각각 쓰이는 dtype(one_hot)과 num_classes(to_categorical)는 결과값의 수를 정해주는 변수이다.
- one_hot()
    - 차수를 직접 마춰야함
- keras.utils.to_categorical()
    - 자동으로 차수를 마춰 변환

```
import tensorflow as tf
import numpy as np
#one_hot
y_true = [[1], [2]]
# y_true = [1, 2]
tf.one_hot(y_true, depth=3)
```

```
    <tf.Tensor: shape=(2, 1, 3), dtype=float32, numpy=
    array([[[0., 1., 0.]],

           [[0., 0., 1.]]], dtype=float32)>
```

```
#to_categorical
#y_true = [[0], [0.5],[1]]
```

```
y_true = [1, 2]
tf.keras.utils.to_categorical(y_true, num_classes=3)
```

```
array([[0., 1., 0.],
       [0., 0., 1.]], dtype=float32)
```

분류 모델을 이용한 딥러닝 구현

- 이진(이항) 분류
  - 두 가지 중 하나로 분류 하는 방법
  - 로지스틱 회귀라고도 부름
    - 와인 분류(화이트 와인, 레드 와인)
- 다항 분류
  - 패션, 손글씨등 분류 세가지 이상으로 분류 해야할때 사용하는 방법
    - 패션(옷) 분류, 손글씨 분류, 와인 품질 분류

와인 분류(이항 분류)

와인 데이터셋 불러오기

```
import pandas as pd
red = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-qu
white = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-
#타입 데이터 추가하기
red['type'] = 0 #0은 레드
white['type'] = 1 #1은 화이트
wine = pd.concat([red, white])
print(red.head())
print(white.head())
```

```
   fixed acidity  volatile acidity  citric acid  ...  alcohol  quality  type
0            7.4              0.70         0.00  ...      9.4        5     0
1            7.8              0.88         0.00  ...      9.8        5     0
2            7.8              0.76         0.04  ...      9.8        5     0
3           11.2              0.28         0.56  ...      9.8        6     0
4            7.4              0.70         0.00  ...      9.4        5     0

[5 rows x 13 columns]
   fixed acidity  volatile acidity  citric acid  ...  alcohol  quality  type
0            7.0              0.27         0.36  ...      8.8        6     1
1            6.3              0.30         0.34  ...      9.5        6     1
2            8.1              0.28         0.40  ...     10.1        6     1
3            7.2              0.23         0.32  ...      9.9        6     1
4            7.2              0.23         0.32  ...      9.9        6     1

[5 rows x 13 columns]
```

데이터 확인

```
print(red.shape)
print(white.shape)
print(wine.info())
```

```
(1599, 13)
(4898, 13)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6497 entries, 0 to 4897
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  type                  6497 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 710.6 KB
None
```

```
import matplotlib.pyplot as plt
wine = pd.concat([red, white])
plt.hist(wine['type'])
# plt.xticks([0, 1])
plt.xticks([0.05, 0.95], ['red', 'white'])
plt.show()

print(wine['type'].value_counts())
##그래프를 그리는 또다른 방법
# import seaborn as sns
# # plt.hist(wine['type'], bins=2, rwidth=0.8)
# # 히스토그램과 비슷한 Count 그래프. 이산값을 나타낸다.
# ax = plt.subplots()
# ax = sns.countplot(x='type', data=wine)
# ax = sns.countplot(x='type', data=wine)
# ax.set_title('Count of wine types')
# ax.set_xlabel('red or white')
# ax.set_ylabel('Frequency')
```

데이터 정규화 작업

```
wine_norm = (wine - wine.min()) / (wine.max() - wine.min())
print(wine_norm.head())
print(wine_norm.describe())
```

```
   fixed acidity  volatile acidity  citric acid  ...   alcohol   quality  type
0       0.297521          0.413333     0.000000  ...  0.202899  0.333333   0.0
1       0.330579          0.533333     0.000000  ...  0.260870  0.333333   0.0
2       0.330579          0.453333     0.024096  ...  0.260870  0.333333   0.0
3       0.611570          0.133333     0.337349  ...  0.260870  0.500000   0.0
4       0.297521          0.413333     0.000000  ...  0.202899  0.333333   0.0

[5 rows x 13 columns]
       fixed acidity  volatile acidity  ...      quality         type
count    6497.000000       6497.000000  ...  6497.000000  6497.000000
mean        0.282257          0.173111  ...     0.469730     0.753886
std         0.107143          0.109758  ...     0.145543     0.430779
min         0.000000          0.000000  ...     0.000000     0.000000
25%         0.214876          0.100000  ...     0.333333     1.000000
50%         0.264463          0.140000  ...     0.500000     1.000000
75%         0.322314          0.213333  ...     0.500000     1.000000
max         1.000000          1.000000  ...     1.000000     1.000000

[8 rows x 13 columns]
```

데이터 섞기

- sample(frac=1)
  - 데이터를 썩는 함수
    frac가 1이면 섞은후 모든 데이터 반환
    0.8이면 섞은후 80%만 반환
- to_numpy()
  - 데이터를 numpy array로 변환

```
import numpy as np
wine_shuffle = wine_norm.sample(frac=1)
print(wine_shuffle.head())
wine_np = wine_shuffle.to_numpy()
print(wine_np[:5])
```

```
      fixed acidity  volatile acidity  citric acid  ...   alcohol   quality  t
```

```
3427       0.165289          0.106667     0.156627  ...  0.463768  0.500000
85         0.272727          0.240000     0.373494  ...  0.101449  0.500000
1561       0.330579          0.346667     0.156627  ...  0.275362  0.333333
267        0.338843          0.180000     0.277108  ...  0.695652  0.833333
3597       0.231405          0.073333     0.168675  ...  0.405797  0.500000

[5 rows x 13 columns]
[[0.16528926 0.10666667 0.15662651 0.14493865 0.04983389 0.21527778
  0.359447   0.12801234 0.47286822 0.15730337 0.46376812 0.5
  1.        ]
 [0.27272727 0.24       0.37349398 0.17177914 0.05813953 0.17708333
  0.33640553 0.20030846 0.31007752 0.13483146 0.10144928 0.5
  1.        ]
 [0.33057851 0.34666667 0.15662651 0.02147239 0.1179402  0.10416667
  0.28801843 0.17563139 0.37984496 0.16853933 0.27536232 0.33333333
  0.        ]
 [0.33884298 0.18       0.27710843 0.04601227 0.11461794 0.04861111
  0.07142857 0.19645267 0.48837209 0.35955056 0.69565217 0.83333333
  0.        ]
 [0.23140496 0.07333333 0.1686747  0.17177914 0.05481728 0.18402778
  0.30184332 0.15056873 0.35658915 0.08426966 0.4057971  0.5
  1.        ]]
```

데이터 분리

- train_idx = int(len(wine_np) * 0.8)
  - 데이터의 80%지점에 인덱스를 반환
- 훈련 데이터
  - train_X = wine_np[:train_idx, :-1]
    - 행은 처음부터 80%지점 까지 / 열은 정답 데이터 전까지
  - train_y = wine_np[:train_idx, :-1]
    - 행은 처음부터 80%지점 까지 / 열은 정답 데이터만
- 테스트 데이터
  - test_X = wine_np[:train_idx, :-1]
    - 행은 80%지점부터 끝까지 / 열은 정답 데이터 전까지
  - test_y = wine_np[:train_idx, :-1]
    - 행은 80%지점부터 끝까지 / 열은 정답 데이터만

```
import tensorflow as tf
train_idx = int(len(wine_np) * 0.8)##데이터의 80%지점에 인덱스를 반환
train_X, train_Y = wine_np[:train_idx, :-1], wine_np[:train_idx, -1]#훈련 데이터
#train_X = wine_np[:train_idx, :-1] =>행은 처음부터 80%지점 까지 / 열은 정답 데이터 전까지
#train_Y = wine_np[:train_idx, -1] =>행은 처음부터 80%지점 까지 / 열은 정답 데이터만
test_X, test_Y = wine_np[train_idx:, :-1], wine_np[train_idx:, -1]#테스트 데이터
#test_X = wine_np[train_idx:, :-1] =>행은 80%지점부터 끝까지 / 열은 정답 데이터 전까지
#test_Y = wine_np[train_idx:, -1] =>행은 80%지점부터 끝까지 / 열은 정답 데이터만
print(train_X[0])
print(train_Y[0])
```

```
print(test_X[0])
print(test_Y[0])
#train_Y = tf.keras.utils.to_categorical(train_Y, num_classes=2)
#test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=2)

train_Y = tf.keras.utils.to_categorical(train_Y, num_classes=2)
test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=2)
```

```
[0.16528926 0.10666667 0.15662651 0.14493865 0.04983389 0.21527778
 0.359447   0.12801234 0.47286822 0.15730337 0.46376812 0.5       ]
1.0
[0.21487603 0.12666667 0.29518072 0.10276074 0.06146179 0.18055556
 0.46082949 0.16367843 0.40310078 0.11797753 0.17391304 0.5       ]
1.0
```

이항 분류 모델 생성

- 와인 데이터셋 분류 모델
- 사용 활성화 함수
  - 소프트맥스 함수(Softmax)
    - 결과의 총합은 1
    - 큰 값을 강조하고 작은 값은 약화 시키는 효과
- 층구조
  - 5층구조
    - 입력층(12데이터 input) -> 3층의 은닉층(48뉴런 > 24뉴런 > 12뉴런) -> 출력층(2개의 output)으로 구성

```
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=48, activation='relu', input_shape=(12,)),
    tf.keras.layers.Dense(units=24, activation='relu'),
    tf.keras.layers.Dense(units=12, activation='relu'),
    tf.keras.layers.Dense(units=2, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='categorical_crosse

model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_40 (Dense)             (None, 48)                624

 dense_41 (Dense)             (None, 24)                1176

 dense_42 (Dense)             (None, 12)                300
```

```
 dense_43 (Dense)             (None, 2)                 26

=================================================================
Total params: 2,126
Trainable params: 2,126
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWar
  super(Adam, self).__init__(name, **kwargs)
```

학습

- epochs=30
  - 전체 데이터 30번 반복 학습
- batch_size=25
  - 데이터를 25개씩 나눠 가중치와 편향을 수정하며 학습
- validation_split=0.2
  - 훈련 데이터에서 20%를 검증 데이터로 활용

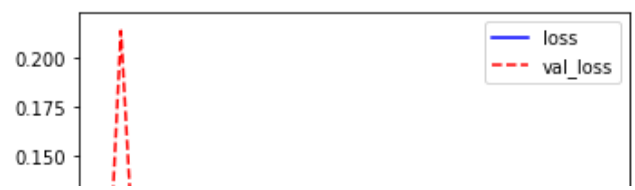
```
history = model.fit(train_X, train_Y, epochs=30, batch_size=25, validation_split=0.
```

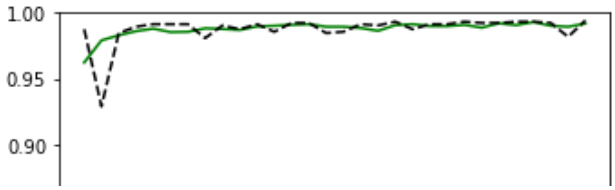
```
Epoch 1/30
167/167 [==============================] - 1s 3ms/step - loss: 0.1160 - accura
Epoch 2/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0679 - accura
Epoch 3/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0579 - accura
Epoch 4/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0505 - accura
Epoch 5/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0420 - accura
Epoch 6/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0505 - accura
Epoch 7/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0487 - accura
Epoch 8/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0496 - accura
Epoch 9/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0437 - accura
Epoch 10/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0500 - accura
Epoch 11/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0382 - accura
Epoch 12/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0379 - accura
Epoch 13/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0357 - accura
Epoch 14/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0374 - accura
Epoch 15/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0385 - accura
Epoch 16/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0364 - accura
Epoch 17/30
```

```
167/167 [==============================] - 0s 2ms/step - loss: 0.0448 - accura
Epoch 18/30
167/167 [==============================] - 0s 3ms/step - loss: 0.0460 - accura
Epoch 19/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0364 - accura
Epoch 20/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0369 - accura
Epoch 21/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0439 - accura
Epoch 22/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0399 - accura
Epoch 23/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0383 - accura
Epoch 24/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0372 - accura
Epoch 25/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0335 - accura
Epoch 26/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0351 - accura
Epoch 27/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0321 - accura
Epoch 28/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0395 - accura
Epoch 29/30
167/167 [==============================] - 0s 2ms/step - loss: 0.0401 - accura
Epoch 30/30
```

손실값과 정확도 시각화 및 평가

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()
```






```
model.evaluate(test_X, test_Y)
```

```
41/41 [==============================] - 0s 1ms/step - loss: 0.0224 - accuracy
[0.02244471199810505, 0.994615375995636]
```

랜덤 30개 예측해보기

```
import numpy as np
predict_y=model.predict(test_X)
```

```
s=sample(range(len(test_X)), 30)
for i in s:
  p1=np.argmax(predict_y[i])
  py1=np.argmax(test_Y[i])
  print("예측 :",p1,"정답 :",py1)
  if(p1 == py1):
     print("예측 성공\n")
  else:
    print("예측 실패\n")
```

```
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 0 정답 : 0
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공
```

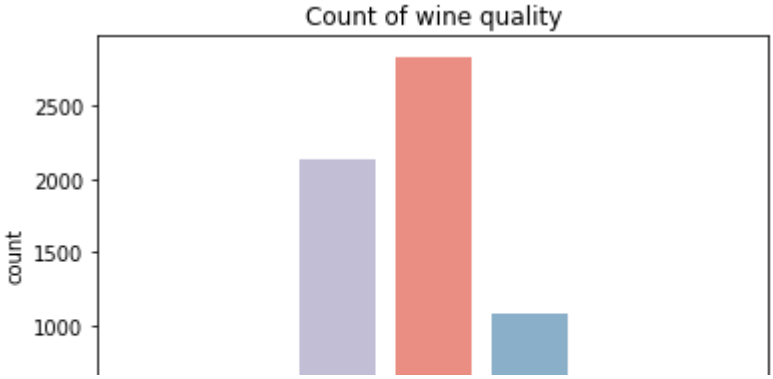
```
예측 : 0 정답 : 0
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 0 정답 : 0
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1
예측 성공

예측 : 1 정답 : 1

예측 성공
```

와인 분류(다항분류)

품질(quality) 데이터 확인

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
red = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-qu
white = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-
red['type'] = 0 #0은 레드
white['type'] = 1 #1은 화이트
wine = pd.concat([red, white])
print(wine['quality'].describe())
print(wine['quality'].value_counts())

sns.countplot(x='quality', data=wine, palette="Set3");
plt.title('Count of wine quality') ;
```

```
count    6497.000000
mean        5.818378
std         0.873255
min         3.000000
25%         5.000000
50%         6.000000
75%         6.000000
max         9.000000
Name: quality, dtype: float64
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```



데이터 재분류

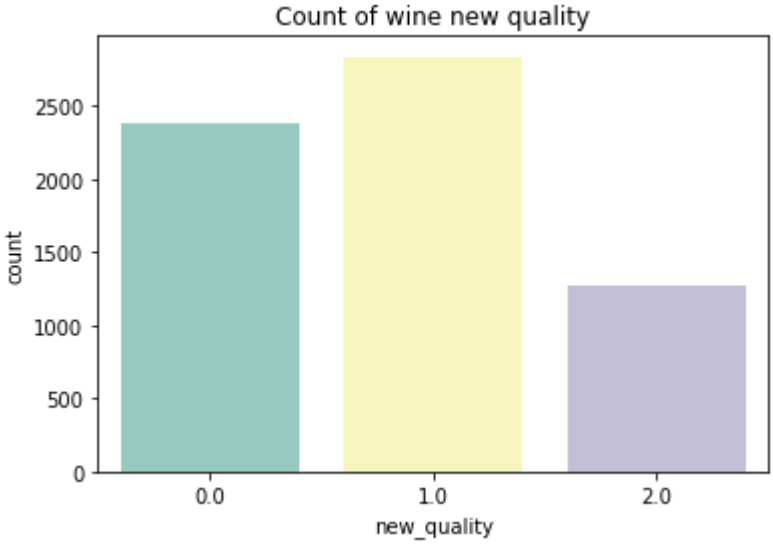
- 3~9등급 데이터를 하급,중급,상급으로 재분류
  - 3 ~ 4, 7 ~ 9급의 데이터량이 작아 딥러닝 모델을 구현하기 어렵움이 있어 데이터를 합쳐 좀더 학습률이 좋은 모델을 만들기 위해서

```
wine.loc[wine['quality'] <= 5, 'new_quality'] = 0#하급
wine.loc[wine['quality'] == 6, 'new_quality'] = 1#중급
wine.loc[wine['quality'] >= 7, 'new_quality'] = 2#상급

print(wine['new_quality'].describe())
print(wine['new_quality'].value_counts())
wine.shape
```

```
count    6497.000000
mean        0.829614
std         0.731124
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         2.000000
Name: new_quality, dtype: float64
1.0    2836
0.0    2384
2.0    1277
Name: new_quality, dtype: int64
(6497, 14)
```

```
sns.countplot('new_quality', data=wine, palette="Set3");
plt.title('Count of wine new quality') ;
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnir
  FutureWarning
```



데이터 정규화,데이터 섞기 ,데이터 분리(test, train)

```
print(wine.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6497 entries, 0 to 4897
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  type                  6497 non-null   int64  
 13  new_quality           6497 non-null   float64
dtypes: float64(12), int64(2)
memory usage: 921.4 KB
None
```

```
del wine['quality']#기존 품질 데이터 삭제
```

```
print(wine.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6497 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  new_quality           6497 non-null   float64
dtypes: float64(12)
memory usage: 819.9 KB
None
```

정규화 문제

- 정규화를 하면서 정답(test_Y, train_Y)도 같이 0, 0.5, 1로 정규화되기 때문에 to_categorical를 하였을 때 0과 0.5가 같은 값으로 변환된다.
    - 때문에 x2를 해줌으로써 다시 값을 돌려놓았다.

```
import numpy as np
import tensorflow as tf
wine_norm = (wine - wine.min()) / (wine.max() - wine.min())
wine_shuffle = wine_norm.sample(frac=1)
wine_np = wine_shuffle.to_numpy()

train_idx = int(len(wine_np) * 0.8)
train_X, train_Y = wine_np[:train_idx, :-1], wine_np[:train_idx, -1]
test_X, test_Y = wine_np[train_idx:, :-1], wine_np[train_idx:, -1]
train_Y*=2#정규화된 정답을 다시 원래 값으로 돌려주기
test_Y*=2
#train_Y = tf.keras.utils.to_categorical(train_Y, num_classes=3)
#test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=3)
#print(train_Y)
for i in range(0,10):
  print(train_Y[i])
  #print(np.argmax(train_Y[i]))

#print(test_Y)
```

```
0.0
0.0
1.0
1.0
2.0
```

```
1.0
0.0
0.0
0.0
1.0
```

다항 분류 모델 생성 및 학습

- 와인 데이터셋 분류 모델
- 5층구조
  - 입력층(12데이터 input) -> 3층의 은닉층(48뉴런 > 24뉴런 > 12뉴런) -> 출력층(3개의 output)으로 구성

추가정보

- 손실함수
  - categorical_crossentropy
    - 출력값이 one-hot encoding 된 결과로 나오고 실측 결과와의 비교시에도 실측 결과는 one-hot encoding 형태로 구성
  - sparse_categorical_crossentropy
    - 일반 정수값의 결과와 비교시에도 실측 결과값이 정수 형태로 구성

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=48, activation='relu', input_shape=(11,)),
    tf.keras.layers.Dense(units=24, activation='relu'),
    tf.keras.layers.Dense(units=12, activation='relu'),
    tf.keras.layers.Dense(units=3, activation='softmax')
])

#model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='categorical_cross
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['a
#model.compile(optimizer='SGD', loss='mse', metrics=['accuracy'])
#model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='sparse_categorica


history = model.fit(train_X, train_Y, epochs=150, batch_size=32, validation_split=0
```

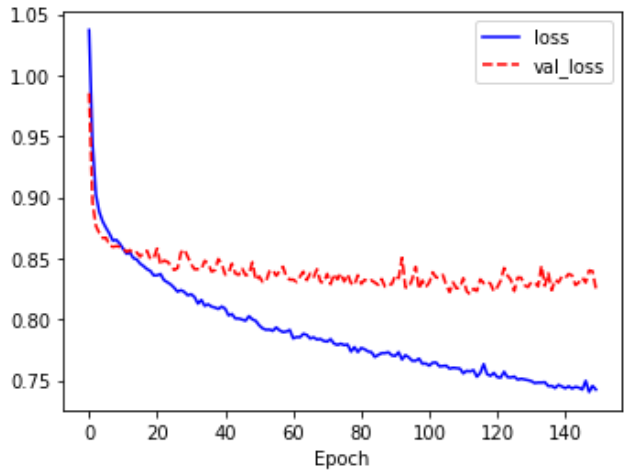
```
Epoch 1/150
122/122 [==============================] - 1s 4ms/step - loss: 0.6271 - accura
Epoch 2/150
122/122 [==============================] - 0s 3ms/step - loss: 0.4432 - accura
Epoch 3/150
122/122 [==============================] - 0s 2ms/step - loss: 0.4200 - accura
Epoch 4/150
122/122 [==============================] - 0s 2ms/step - loss: 0.4121 - accura
Epoch 5/150
122/122 [==============================] - 0s 2ms/step - loss: 0.4067 - accura
Epoch 6/150
122/122 [==============================] - 0s 2ms/step - loss: 0.4025 - accura
Epoch 7/150
```

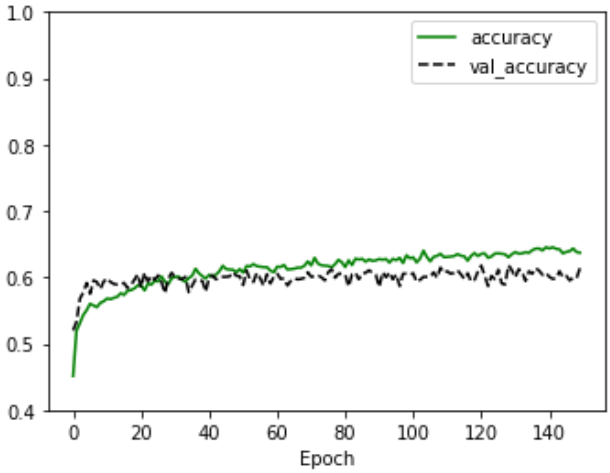
```
122/122 [==============================] - 0s 3ms/step - loss: 0.4014 - accura
Epoch 8/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3962 - accura
Epoch 9/150
122/122 [==============================] - 0s 3ms/step - loss: 0.3941 - accura
Epoch 10/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3934 - accura
Epoch 11/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3927 - accura
Epoch 12/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3903 - accura
Epoch 13/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3890 - accura
Epoch 14/150
122/122 [==============================] - 0s 3ms/step - loss: 0.3894 - accura
Epoch 15/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3853 - accura
Epoch 16/150
122/122 [==============================] - 0s 3ms/step - loss: 0.3865 - accura
Epoch 17/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3839 - accura
Epoch 18/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3838 - accura
Epoch 19/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3801 - accura
Epoch 20/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3810 - accura
Epoch 21/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3783 - accura
Epoch 22/150
122/122 [==============================] - 0s 3ms/step - loss: 0.3781 - accura
Epoch 23/150
122/122 [==============================] - 0s 3ms/step - loss: 0.3773 - accura
Epoch 24/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3758 - accura
Epoch 25/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3741 - accura
Epoch 26/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3733 - accura
Epoch 27/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3726 - accura
Epoch 28/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3731 - accura
Epoch 29/150
122/122 [==============================] - 0s 2ms/step - loss: 0.3711 - accura
Epoch 30/150
```

다항 분류 모델 학습 결과 시각화 및 모델 평가

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
```

```
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.4, 1)
plt.legend()

plt.show()
model.evaluate(test_X, test_Y)
```






```
41/41 [==============================] - 0s 1ms/step - loss: 0.8172 - accuracy
[0.817187488079071, 0.5984615087509155]
```

랜덤한 20개 데이터 예측 결과 보기

```
import numpy as np
from random import sample
predict_y=model.predict(test_X)


s=sample(range(len(test_X)), 20)
for i in s:
  p1=np.argmax(predict_y[i])
  #py1=np.argmax(py[i])
  print(predict_y[i])
  print("예측 :",p1,"정답 :",test_Y[i])
  if(p1 == test_Y[i]):
     print("예측 성공\n")
  else:
    print("예측 실패\n")
```

```
[0.07801843 0.3385176  0.58346397]
예측 : 2 정답 : 2.0
예측 성공

[0.0621475  0.24068056 0.6971719 ]
예측 : 2 정답 : 1.0
예측 실패

[0.11384294 0.876676   0.00948095]
예측 : 1 정답 : 1.0
```

```
예측 성공

[0.16752209 0.684597   0.14788088]
예측 : 1 정답 : 1.0
예측 성공

[0.00376112 0.26975605 0.72648287]
예측 : 2 정답 : 2.0
예측 성공

[6.7824250e-01 3.2130745e-01 4.5000704e-04]
예측 : 0 정답 : 0.0
예측 성공

[0.39285663 0.5271891  0.07995437]
예측 : 1 정답 : 1.0
예측 성공

[0.07526888 0.47559297 0.44913816]
예측 : 1 정답 : 2.0
예측 실패

[0.6391335  0.35969737 0.00116915]
예측 : 0 정답 : 1.0
예측 실패

[7.9617584e-01 2.0374531e-01 7.8862795e-05]
예측 : 0 정답 : 1.0
예측 실패

[7.9094249e-01 2.0861605e-01 4.4150034e-04]
예측 : 0 정답 : 1.0
예측 실패

[0.09058277 0.51797724 0.39144003]
예측 : 1 정답 : 1.0
예측 성공

[0.11599123 0.7021448  0.18186404]
예측 : 1 정답 : 2.0
예측 실패

[0.08464403 0.8343817  0.08097433]
예측 : 1 정답 : 1.0
예측 성공

[0.01797598 0.4746011  0.507423  ]
예측 : 2 정답 : 2.0
예측 성공
```

더블클릭 또는 Enter 키를 눌러 수정

패션 분류(다항 분류)

패션 데이터셋 불러오기

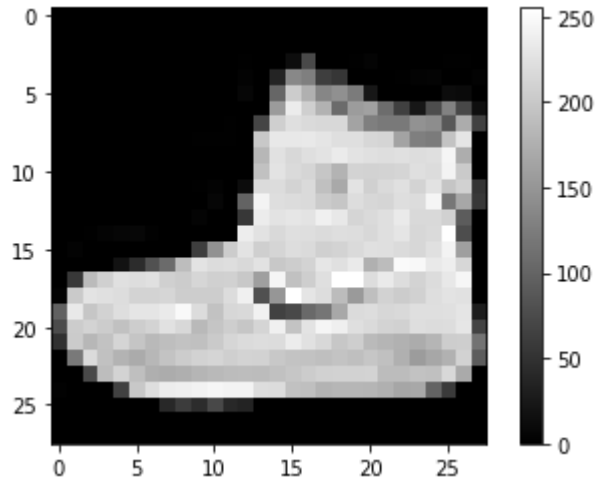
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()
```

```
print(len(train_X), len(test_X))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
32768/29515 [==============================] - 0s 0us/step
40960/29515 [==========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
16384/5148 [===============================================================================
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
60000 10000
```

데이터 확인 및 정규화

```
import matplotlib.pyplot as plt
plt.imshow(train_X[0], cmap='gray')
plt.colorbar()
plt.show()

print(train_Y[0])
```




```
9
```

```
train_X = train_X / 255.0
test_X = test_X / 255.0

print(train_X[0])
```

```
 0.32156863 0.41960784 0.74117647 0.89411765 0.86274451 0.87058824
 0.85098039 0.88627451 0.78431373 0.80392157 0.82745098 0.90196078
 0.87843137 0.91764706 0.69019608 0.7372549  0.98039216 0.97254902
 0.91372549 0.93333333 0.84313725 0.        ]
[0.         0.22352941 0.73333333 0.81568627 0.87843137 0.86666667
 0.87843137 0.81568627 0.8        0.83921569 0.81568627 0.81960784
 0.78431373 0.62352941 0.96078431 0.75686275 0.80784314 0.8745098
 1.         1.         0.86666667 0.91764706 0.86666667 0.82745098
 0.8627451  0.90980392 0.96470588 0.        ]
[0.01176471 0.79215686 0.89411765 0.87843137 0.86666667 0.82745098
 0.82745098 0.83921569 0.80392157 0.80392157 0.80392157 0.8627451
```

```
  0.94117647 0.31372549 0.58823529 1.         0.89803922 0.86666667
  0.7372549  0.60392157 0.74901961 0.82352941 0.8        0.81960784
  0.87058824 0.89411765 0.88235294 0.        ]
 [0.38431373 0.91372549 0.77647059 0.82352941 0.87058824 0.89803922
  0.89803922 0.91764706 0.97647059 0.8627451  0.76078431 0.84313725
  0.85098039 0.94509804 0.25490196 0.28627451 0.41568627 0.45882353
  0.65882353 0.85882353 0.86666667 0.84313725 0.85098039 0.8745098
  0.8745098  0.87843137 0.89803922 0.11372549]
 [0.29411765 0.8        0.83137255 0.8        0.75686275 0.80392157
  0.82745098 0.88235294 0.84705882 0.7254902  0.77254902 0.80784314
  0.77647059 0.83529412 0.94117647 0.76470588 0.89019608 0.96078431
  0.9372549  0.8745098  0.85490196 0.83137255 0.81960784 0.87058824
  0.8627451  0.86666667 0.90196078 0.2627451 ]
 [0.18823529 0.79607843 0.71764706 0.76078431 0.83529412 0.77254902
  0.7254902  0.74509804 0.76078431 0.75294118 0.79215686 0.83921569
  0.85882353 0.86666667 0.8627451  0.9254902  0.88235294 0.84705882
  0.78039216 0.80784314 0.72941176 0.70980392 0.69411765 0.6745098
  0.70980392 0.80392157 0.80784314 0.45098039]
 [0.         0.47843137 0.85882353 0.75686275 0.70196078 0.67058824
  0.71764706 0.76862745 0.8        0.82352941 0.83529412 0.81176471
  0.82745098 0.82352941 0.78431373 0.76862745 0.76078431 0.74901961
  0.76470588 0.74901961 0.77647059 0.75294118 0.69019608 0.61176471
  0.65490196 0.69411765 0.82352941 0.36078431]
 [0.         0.         0.29019608 0.74117647 0.83137255 0.74901961
  0.68627451 0.6745098  0.68627451 0.70980392 0.7254902  0.7372549
  0.74117647 0.7372549  0.75686275 0.77647059 0.8        0.81960784
  0.82352941 0.82352941 0.82745098 0.7372549  0.7372549  0.76078431
  0.75294118 0.84705882 0.66666667 0.        ]
 [0.00784314 0.         0.         0.         0.25882353 0.78431373
  0.87058824 0.92941176 0.9372549  0.94901961 0.96470588 0.95294118
  0.95686275 0.86666667 0.8627451  0.75686275 0.74901961 0.70196078
  0.71372549 0.71372549 0.70980392 0.69019608 0.65098039 0.65882353
  0.38823529 0.22745098 0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.15686275 0.23921569 0.17254902 0.28235294 0.16078431
  0.1372549  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.        ]]
```

다항 분류 모델 생성 및 학습

- 패션 데이터셋 분류 모델
- 3층구조
  - 입력층(28*28 데이터 input) -> 1층의 은닉층(128뉴런) -> 출력층(10개의 output)으로 구성

```
model = tf.keras.Sequential([
```

```
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense_44 (Dense)            (None, 128)               100480

 dense_45 (Dense)            (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

분류 모델 학습

```
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

```
Epoch 1/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.5273 - accu
Epoch 2/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.3934 - accu
Epoch 3/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.3499 - accu
Epoch 4/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.3230 - accu
Epoch 5/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.3046 - accu
Epoch 6/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2870 - accu
Epoch 7/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2755 - accu
Epoch 8/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2628 - accu
Epoch 9/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2516 - accu
Epoch 10/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2427 - accu
Epoch 11/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2319 - accu
Epoch 12/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2245 - accu
Epoch 13/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2168 - accu
Epoch 14/25
```

```
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2102 - accu
Epoch 15/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.2046 - accu
Epoch 16/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1988 - accu
Epoch 17/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1947 - accu
Epoch 18/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1879 - accu
Epoch 19/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1798 - accu
Epoch 20/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1789 - accu
Epoch 21/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1728 - accu
Epoch 22/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1694 - accu
Epoch 23/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1656 - accu
Epoch 24/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1578 - accu
Epoch 25/25
1407/1407 [==============================] - 4s 3ms/step - loss: 0.1549 - accu
```

```
model.evaluate(test_X, test_Y)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.4261 - accura
[0.42611318826675415, 0.8745999932289124]
```

```
import numpy as np
from random import sample
predict_y=model.predict(test_X)


s=sample(range(len(test_X)), 20)
for i in s:
  p1=np.argmax(predict_y[i])
  #py1=np.argmax(py[i])
  print(predict_y[i])
  print("예측 :",p1,"정답 :",test_Y[i])
  if(p1 == test_Y[i]):
     print("예측 성공\n")
  else:
    print("예측 실패\n")
```

```
[6.6139866e-10 1.3078871e-07 4.2692985e-11 3.5542144e-19 6.9463241e-11
 1.0566552e-03 1.5997378e-12 9.9438632e-01 2.7779996e-12 4.5569651e-03]
예측 : 7 정답 : 7
예측 성공

[1.2047751e-16 2.6961697e-14 3.1810809e-16 2.8743936e-16 1.4434947e-17
 2.6866898e-09 2.3359764e-14 3.1773836e-04 2.3032624e-15 9.9968231e-01]
예측 : 9 정답 : 9
예측 성공

[4.3845301e-15 2.3737242e-18 2.6336853e-14 5.0074667e-21 1.7689598e-20
 4.3790085e-10 3.2126235e-14 7.8759954e-04 8.0958240e-15 9.9921238e-01]
예측 : 9 정답 : 9
```

```
예측 성공

[3.1620215e-03 1.6925335e-12 9.9679971e-01 3.9074291e-15 3.8219270e-05
 1.7962673e-16 5.6064707e-08 1.1704243e-23 8.3371211e-13 5.8200256e-17]
예측 : 2 정답 : 2
예측 성공

[3.43049720e-07 7.47735385e-07 1.44359691e-09 9.99998808e-01
 1.36538515e-11 3.00822560e-17 1.88538564e-07 2.36876254e-24
 4.79527772e-11 1.18294584e-11]
예측 : 3 정답 : 3
예측 성공


[1.0762182e-10 5.9286912e-14 5.6582890e-13 5.3504228e-16 1.2941586e-15
 3.7565482e-11 3.7311300e-13 9.9999869e-01 3.2487049e-10 1.3331565e-06]
예측 : 7 정답 : 7
예측 성공

[2.72140002e-11 1.00000000e+00 4.07900777e-14 2.81484710e-12
 1.08031687e-13 3.86297611e-20 3.33363614e-11 1.66017877e-27
 1.08923926e-23 1.51655701e-23]
예측 : 1 정답 : 1
예측 성공

[6.7666139e-07 9.4857677e-10 6.5502170e-03 3.6265718e-08 9.9285382e-01
 4.5977643e-14 5.8094668e-04 2.7157986e-19 1.4327278e-05 1.1356662e-16]
예측 : 4 정답 : 4
예측 성공

[1.1596868e-06 8.7605315e-07 2.2720732e-03 9.9743193e-01 1.5645349e-04
 6.7513753e-07 1.3667019e-04 1.9497344e-09 8.2685929e-08 1.6690562e-11]
예측 : 3 정답 : 3
예측 성공

[3.17971430e-12 1.00000000e+00 4.11546373e-14 5.17563006e-11
 3.70807768e-13 2.78355654e-11 3.62798984e-12 2.83021205e-23
 1.08947164e-16 2.50178030e-17]
예측 : 1 정답 : 1
예측 성공

[4.9448927e-06 2.5259701e-06 1.2350867e-03 9.9767011e-01 8.3831523e-04
 3.0835981e-05 2.1795543e-04 2.2052007e-09 1.2848956e-07 1.3348095e-10]
예측 : 3 정답 : 3
예측 성공
```

CNN(합성곱)

- 이미지의 공간 정보를 유지한 상태로 학습이 가능한 모델
  - 특징 추출기 + 분류기 구성
- 일반 Dense() 층과 비교
  - Fully Connected Neural Network와 비교하여 다음과 같은 차별성
    - 각 레이어의 입출력 데이터의 형상 유지

        - 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식
    - 복수의 필터로 이미지의 특징 추출 및 학습
        - 추출한 이미지의 특징을 모으고 강화하는 Pooling 레이어
    - 일반 신경망과 비교하여 학습 패러미터가 매우 적음
        - 필터를 공유 패러미터로 사용하기 때문

- LeCun 1998년
    - LeNet이라는 Network를 1998년에 제안
        - 얀 르쿤(Yann Lecun) 연구팀
    - 이것이 최초의 CNN

CNN층과 분류기

- CNN 구조
    - CNN은 컨볼루션(Convolutiona) 층과 풀링(pooling) 층 그리고 분류기로 구성
    - 컨볼루션 층(합성곱 연산 + 활성화 함수) 다음에는 풀링 층을 추가하는 것이 일반적입니다.
    - 컨볼루션 층
        - 입력층에서 입력 데이터를 받아와서, 그것을 conv 즉 합성곱하고, 그 결과를 relu와 같은 활성화 함수로 계산하고, pooling하는 형식으로 데이터의 '특징'이라는 것을 뽑아옵니다.
        - 각 이미지에서 특정 특징을 활성화하는 컨볼루션 필터 집합에 입력 이미지를 통과
    - 폴링(서브샘플링) 층
        - 최댓값을 구하는 연산으로, 2*2는 대상 영역의 크기를 뜻합니다.
        - 특성 맵을 다운샘플링하여 특성 맵의 크기를 줄이는 풀링 연산이 이루어집니다. => 비선형 다운 샘플링
        - 매개변수가 없음
    - 분류기
        - CNN 마지막 부분에는 이미지를 분류 하기위한 Fully Connected 레이어를 추가
            - 처음은 Flatten(평탄화)
              이미지의 특징을 추출하는 부분과 이미지를 분류하는 부분 사이에 이미지 형태의 데이터를 배열 형태로 변환

- 특징 추출기
    - 자동으로 특징을 추출하는 필터를 생성하는 목적
        - Convolution Layer와 Pooling Layer를 여러 겹 쌓는 형태로 구성
        - Convolution Layer: 입력 데이터에 필터를 적용 후 활성화 함수를 반영하는 필수 요소
        - Pooling Layer: 선택적인 레이어
            - Subsampling, downsampling 이라고도 부름

CNN 구현

컨볼루션 계산 방법

- 필터(가중치)와 편향이 필요
  - 필터 하나당 특징맵(pixel)하나가 나옴
    - 때문에 필터가 여러개면 그에 대응하는 특징맵이 생성
  - 필터 = 커널 = 원도(CNN에서만)

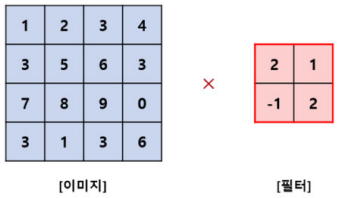
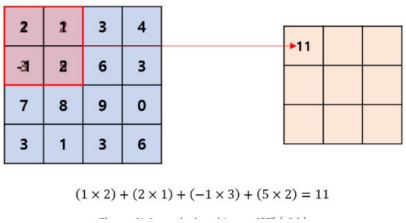


Figure 7: Convolutional Layer 예제



(4x4 흑백 이미지, 2x2 필터)

- 계산 방법은 이미지에 필터를 사진과 같이 올리고 각 자리가 같은 수를 곱한 뒤 모든수를 더하면 첫 칸에 컨볼루션 계산이 완료 된다. 이후 한칸씩(Stride=1) 옆으로 움직여 총 3x3로 다운샘플링이 된다.
  - Stride
    - 필터가 움직이는 간격

컨볼루션의 가중치와 편향

- 가중치
  - 컨볼루션에서 가중치는 필터 값을 가중치라 할 수 있다.
- 편향
  - 편향은 필터마다 하나씩 존재 한다.

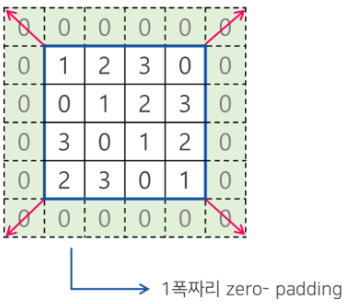
채널

- 이미지에서 겹쳐지는 구분
- 입력 데이터가 여러 채널을 갖는 경우

- 필터는 채널마다 달리 적용
  - 각 채널의 피처 맵을 합산하여 최종 피처 맵으로 반환
  - 하지만 필터는 하나이기 때문에 피처 맵(결과)는 하나만 생성

패딩(padding)

- 합성곱의 결과인 특징 맵
  - Filter와 Stride에 작용으로 Feature Map 크기는 입력 데이터보다 작음
- 패딩
  - 입력 데이터 외각에 지정된 픽셀만큼 특정 값으로 채워 넣는 것 의미
    - 결과인 특징 맵 크기가 줄어드는 것을 방지하는 방법
  - 보통 패딩 값으로 0으로 채워 넣음

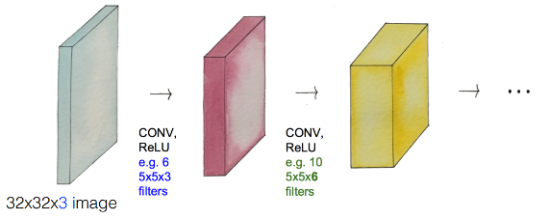


Pooling 레이어

- 데이터의 공간적인 특성을 유지하면서 크기를 줄여주는 층
  - 컨볼루션의 결과인 Activation Map을 줄이거나 특정 데이터를 강조
    - 연속적인 합성곱 층 사이에 삽입
    - 학습할 가중치를 줄이고
    - 과적합(overfitting) 문제도 해결
  - 일반적으로 Pooling 크기와 Stride를 같은 크기로 설정
    - 모든 원소가 한 번씩 처리 되도록 설정
- 풀링의 종류
  - Max Pooling
    - 정사각 행렬의 특정 영역 안에 값의 최댓값
    - 대부분 이것을 사용
  - Average Pooling
    - 정사각 행렬의 특정 영역 안에 평균 값
  - Min Pooling

각 컨볼루션 단계의 패러미터 수

- 패러미터의 수 = 커널의 원소 수 + 편향 수
  - 커널 수(K) * (필터 사이즈(F)2 * 채널 또는 특징맵 수(D) + 1(편향 수))
    - 커널(필터) 수 == 결과의 특징맵 수
- 두 개의 Convolution 층
  - 커널 5 * 5 * 3(채널), 6개를 사용한 컨볼루션 층
    - 6 * ((5*5) * 3 + 1)
  - 커널 5 * 5 * 6(이전 특징맵 수), 10개를 사용한 컨볼루션 층
    - 10 * ((5*5) * 6 + 1)



CNN 실습 해보기

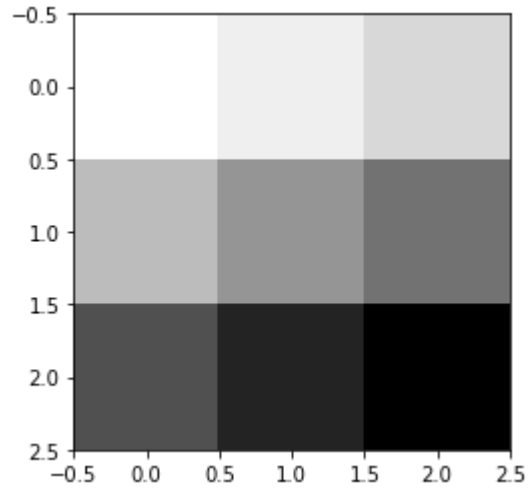
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

CNN 연산을 위해서 3차원의 이미지를 4차원으로 reshape을 해줘야 한다

- 채널 추가

```
image = np.array([[[1,2,3],
                   [4,5,6],
                   [7,8,9]]], dtype=np.float32)
print(image.shape)
image = image.reshape(-1, 3, 3, 1)#채널 추가
'''
image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
'''
print(image.shape)
print(image)
plt.imshow(image.reshape(3,3), cmap='Greys')
```

```
(1, 3, 3)
(1, 3, 3, 1)
[[[[1.]
   [2.]
   [3.]]

  [[4.]
   [5.]
   [6.]]

  [[7.]
   [8.]
   [9.]]]]
<matplotlib.image.AxesImage at 0x7f59d8d02610>
```



커널 2개 적용

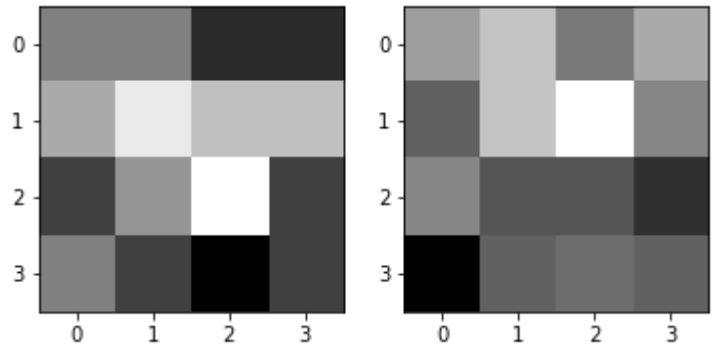
```
kernel_in = np.array([
 [ [[2, 0.1]],[[3, 0.2]] ],
 [ [[0, 0.3]],[[1, 0.4]] ], ])
```

- 2, 3, 0, 1이 하나의 커널
- 0.1, 0.2, 0.3, 0.4가 하나의 커널

```
x_in = np.array([[
  [[2], [1], [2], [0], [1]],
  [[1], [3], [2], [2], [3]],
  [[1], [1], [3], [3], [0]],
  [[2], [2], [0], [1], [1]],
  [[0], [0], [3], [1], [2]], ]])
x = tf.constant(x_in, dtype=tf.float32)

# 2 x 2 커널 2개 적용
kernel_in = np.array([
 [ [[2, 0.1]],[[3, 0.2]] ],
 [ [[0, 0.3]],[[1, 0.4]] ], ])
kernel = tf.constant(kernel_in, dtype=tf.float32)

conv2d = tf.nn.conv2d(x, kernel, strides=[1, 1, 1, 1], padding='VALID')
print("conv2d.shape", conv2d.shape)
```

```
print("결과 자체", conv2d)

conv2d_img = np.swapaxes(conv2d, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(4,4))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(4,4), cmap='gray')
```

```
conv2d.shape (1, 4, 4, 2)
결과 자체 tf.Tensor(
[[[[10.         1.9      ]
   [10.         2.2      ]
   [ 6.         1.6      ]
   [ 6.         2.       ]]

  [[12.         1.4      ]
   [15.         2.2      ]
   [13.         2.7      ]
   [13.         1.7      ]]

  [[ 7.         1.7      ]
   [11.         1.3000001]
   [16.         1.3      ]
   [ 7.         1.       ]]

  [[10.         0.6      ]
   [ 7.         1.4000001]
   [ 4.         1.5      ]
   [ 7.         1.4      ]]]], shape=(1, 4, 4, 2), dtype=float32)
[[10. 10.  6.  6.]
 [12. 15. 13. 13.]
 [ 7. 11. 16.  7.]
 [10.  7.  4.  7.]]
[[1.9       2.2       1.6       2.       ]
 [1.4       2.2       2.7       1.7      ]
 [1.7       1.3000001 1.3       1.       ]
 [0.6       1.4000001 1.5       1.4      ]]
```



✓ 0초 오후 5:24에 완료됨