

2022

# 스마트교통 빅데이터 분석

## VDS 데이터 분석을 위한 딥러닝 모델 실습



2022.8.30.

이 홍 석 (hsyi@kisti.re.kr)



# VDS 데이터 분석을 위한 딥러닝 모델 실습

## LAB 04

✓  
1초

```
[3] import pandas as pd
```

✓  
0초

```
[4] from pandas import datetime
```

✓  
0초

```
[5] def parser(x):  
    return datetime.strptime(x, '%Y-%m-%d %H:%M')
```

✓  
0초

```
[6] df = pd.read_csv('./daejeon_vds16.csv', date_parser=parser)
```

✓  
0초

```
[7] df.head()
```

	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
0	2017-04-02 0:00	43	34	9	0	50.3	1.90
1	2017-04-02 0:05	45	32	13	0	58.9	1.84





df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8064 entries, 0 to 8063
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Date        8064 non-null   object
1   ToVol       8064 non-null   int64
2   SmVol       8064 non-null   int64
3   MeVol       8064 non-null   int64
4   LaVol       8064 non-null   int64
5   Speed       8064 non-null   float64
6   Occ.Rate    8064 non-null   float64
dtypes: float64(2), int64(4), object(1)
memory usage: 441.1+ KB
```



df.describe()

	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
<b>count</b>	8064.000000	8064.000000	8064.000000	8064.000000	8064.000000	8064.000000
<b>mean</b>	110.459945	79.353299	29.948537	1.158110	49.327431	6.166941
<b>std</b>	63.954451	46.802106	19.081136	1.530192	7.921856	6.739946
<b>min</b>	6.000000	2.000000	0.000000	0.000000	9.100000	0.230000
<b>25%</b>	50.000000	35.000000	13.000000	0.000000	44.900000	2.140000
<b>50%</b>	122.000000	87.000000	29.000000	1.000000	48.500000	5.550000
<b>75%</b>	155.000000	111.000000	44.000000	2.000000	54.200000	7.290000
<b>max</b>	338.000000	250.000000	145.000000	16.000000	87.800000	82.100000





```
maxs = df.max()
print(maxs)
```

```

↳ Date      2017-04-29 9:55
   ToVol      338
   SmVol      250
   MeVol      145
   LaVol       16
   Speed     87.8
   Occ.Rate   82.1
   dtype: object

```

```
[12] def get_score(v):
      if v < 20:
          score = 'Jam'
      elif v < 40:
          score = 'Slow'
      else :
          score = 'Normal'
      return score
```

```
df["label"] = df["Speed"].apply(lambda v: get_score(v))
df
```



	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label
0	2017-04-02 0:00	43	34	9	0	50.3	1.90	Normal
1	2017-04-02 0:05	45	32	13	0	58.9	1.84	Normal







df.head()



	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label
0	2017-04-02 0:00	43	34	9	0	50.3	1.90	Normal
1	2017-04-02 0:05	45	32	13	0	58.9	1.84	Normal
2	2017-04-02 0:10	46	34	12	0	50.6	1.87	Normal
3	2017-04-02 0:15	45	36	9	0	50.9	1.72	Normal
4	2017-04-02 0:20	27	13	13	1	62.2	1.12	Normal




\*\* 라벨을 위해 'label'의 텍스트는 범주형 혹은 숫자로 인코딩 해야한다



```
[ ] df['label'].unique()

array(['Normal', 'Slow', 'Jam'], dtype=object)
```

```
[ ] #feature_cols = ['ToVol', 'SmVol', 'Speed', 'Occ.Rate']
#feature_cols = ['ToVol', 'SmVol', 'LaVol', 'MeVol']
feature_cols = ['ToVol', 'Occ.Rate']
target_col = 'label'
X = df[feature_cols]
y = df[target_col]
```

 X.head()

	ToVol	Occ.Rate
0	43	1.90
1	45	1.84
2	46	1.87
3	45	1.72
4	27	1.12

[ ] y.head()

```
0    Normal
1    Normal
2    Normal
3    Normal
4    Normal
Name: label, dtype: object
```

## 2) 출력용 라벨을 머신러닝

텍스트를 숫자로 바꾸자

```
class_dic = {'Jam':0, 'Slow':1, 'Normal':2}
y_ohc = y.apply(lambda z: class_dic[z])
```

```
[ ] y_ohc.head()
```

```
0    2
1    2
2    2
3    2
4    2
Name: label, dtype: int64
```

```
[22] from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(X, y_ohc,
                                                         test_size=0.20, random_state=30)
```

```
[23] print(X_train.shape, y_train.shape)
      print(X_test.shape, y_test.shape)
```

```
(6451, 2) (6451,)
(1613, 2) (1613,)
```

```
[24] from sklearn.preprocessing import StandardScaler
      scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
[25] from sklearn.model_selection import train_test_split
```

```
▶ X_train, X_test, y_train, y_test = train_test_split(X, y_ohc,
                                                    test_size=0.2, random_state=33)
```

```
[27] print(X_train.shape, y_train.shape)
      print(X_test.shape, y_test.shape)
```

```
(6451, 2) (6451,)
(1613, 2) (1613,)
```

```
[35] X_train.head(5)
```

	ToVol	Occ.Rate
1503	58	2.96
6340	20	1.09
6164	85	3.56
4215	152	6.33
3404	121	5.97





```
import tensorflow as tf
from tensorflow import keras

from tensorflow.keras import Sequential, optimizers
from tensorflow.keras.layers import Flatten, Dense, Softmax
```

[30]

```
model = keras.Sequential([
    keras.layers.Dense(64, activation = 'relu', input_shape=[2]),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dense(32, activation = 'relu'),
    keras.layers.Dense(3, activation = 'softmax')
])
```

[37]

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	192
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 32)	2080
dense_3 (Dense)	(None, 3)	99

Total params: 6,531

Trainable params: 6,531

Non-trainable params: 0

분류 문제는 분류의 정확도를 봐야하며, 회귀 문제는 오차(비용함수)로 'loss'를 설정한다.

```
[41] model.compile(loss='SparseCategoricalCrossentropy',  
                  optimizer='adam',  
                  metrics=['accuracy'])
```



```
history = model.fit(X_train, y_train, epochs = 100,  
                    validation_data=(X_test, y_test),  
                    batch_size = 128, verbose=2)
```

```
51/51 - 0s - loss: 0.1058 - accuracy: 0.9609 - val_loss: 0.0954 - val_accuracy: 0.9659 - 121ms/epoch - 2ms/step  
Epoch 65/100  
51/51 - 0s - loss: 0.0956 - accuracy: 0.9650 - val_loss: 0.0984 - val_accuracy: 0.9634 - 116ms/epoch - 2ms/step  
Epoch 66/100
```



```
▶ model.evaluate(X_test, y_test)
```



```
51/51 [=====] - 0s 1ms/step - loss: 0.0960 - accuracy: 0.9634  
[0.09600767493247986, 0.9634221792221069]
```



```
[33] history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	192
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 32)	2080
dense_3 (Dense)	(None, 3)	99

Total params: 6,531

Trainable params: 6,531

Non-trainable params: 0

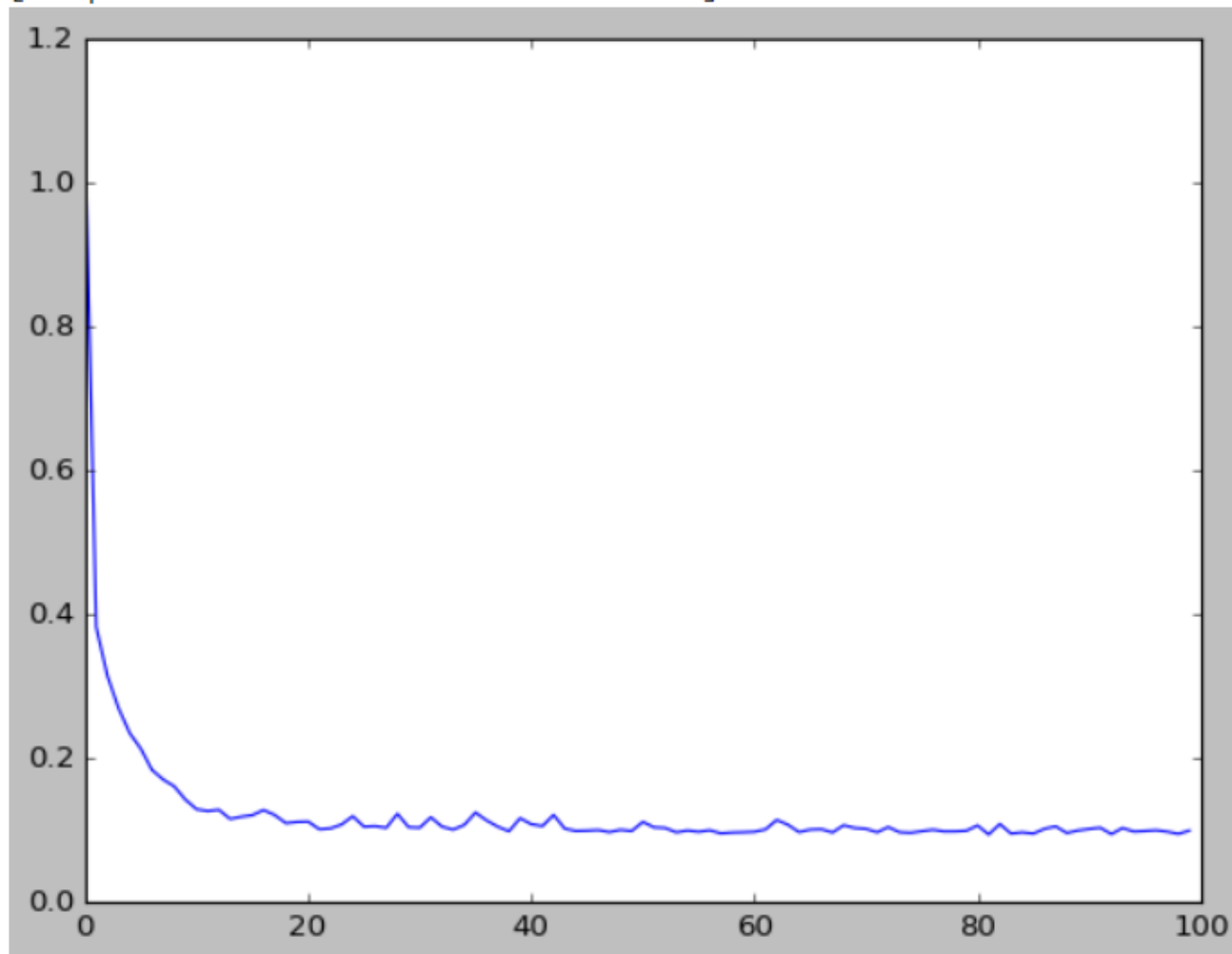
```
[43] import numpy as np
```

```
[44] acc_dnn = history.history['accuracy'][np.argmin(history.history['loss'])]  
print('The accuracy of the Deep Learning is:', acc_dnn)
```

The accuracy of the Deep Learning is: 0.965741753578186

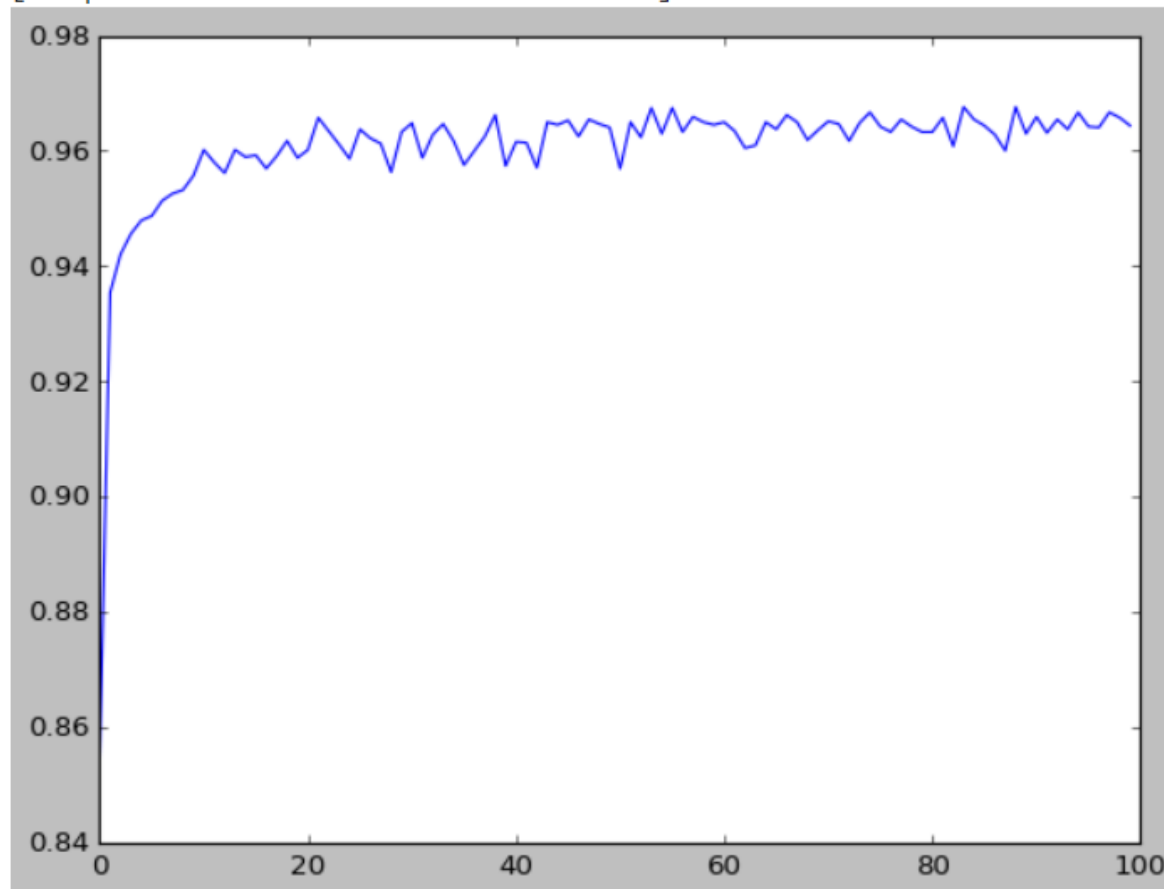
▶ `plt.plot(history.history['loss'])`

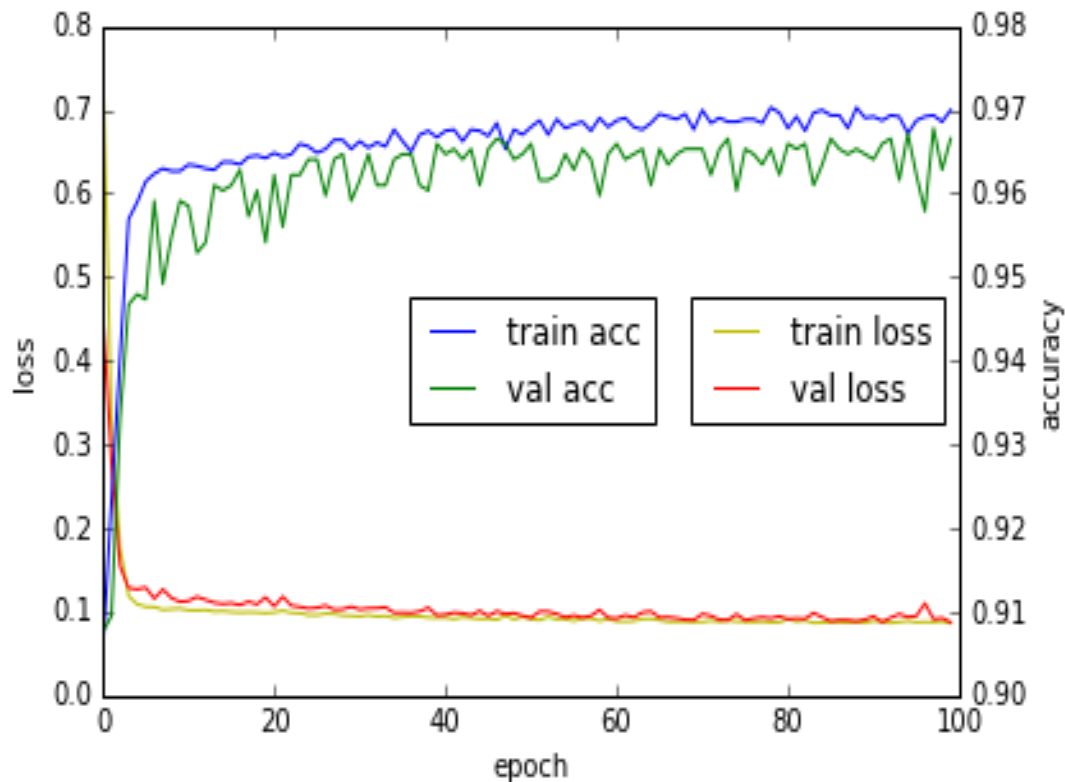
↳ [`matplotlib.lines.Line2D` at 0x7fa59ada6f50>]



```
[46] plt.plot(history.history['accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7fa5992f2910>]
```





```
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')
acc_ax.plot(history.history['accuracy'], 'b', label='train acc')
acc_ax.plot(history.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuracy')

loss_ax.legend(loc='center right')
acc_ax.legend(loc='center')
plt.show()
```



```
In [67]: models = pd.DataFrame({
    'Model': ['Logistic Regression', 'Support Vector Machines','RandomForest',
              'K-Nearest Neighbours', 'Decision Tree','Deep Learning'],
    'Score': [acc_lr, acc_svm, acc_rf, acc_knn, acc_dt, acc_dnn]})
models.sort_values(by='Score', ascending=False)
```

Out [67] :

	Model	Score
5	Deep Learning	0.969927
1	Support Vector Machines	0.967142
3	K-Nearest Neighbours	0.967142
2	RandomForest	0.961562
4	Decision Tree	0.961562
0	Logistic Regression	0.960322

# 딥러닝 기초

# 선형회귀 소개 및 실습

회귀의 목적함수로는 무엇을 사용하는가?

## 회귀의 역사

- ✓ 영국의 통계학자 갈톤(Galton)의 유전적 특성중에 부모와 자식의 키 관계
- ✓ "사람의 키는 평균 키로 회귀(Regression)하려는 경향을 가진다는 자연의 법칙이 있다"
- ✓ 회귀 분석은 데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법

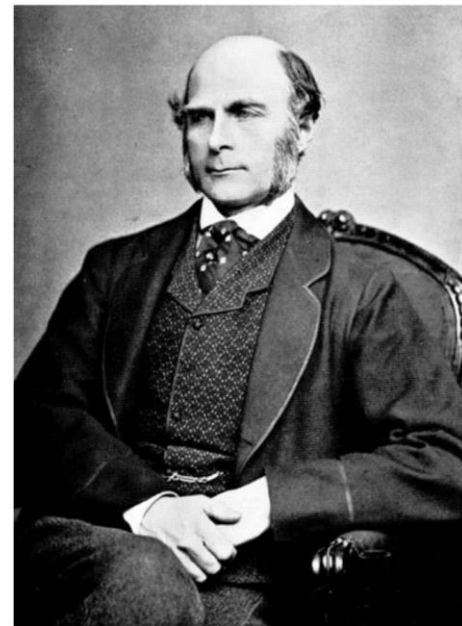
## 지도학습은 2가지 유형으로 나뉨

- ✓ 회귀는 연속적인 숫자 값
- ✓ 분류는 예측값이 카테고리과 같은 이산형 클래스 값

## 선형회귀

- ✓ 실제 값과 예측 값의 차이를 최소화하는 직선형 회귀선을 최적화하는 방식
- ✓ 단순 선형회귀는 1개의 독립변수, 1개의 종속변수.
- ✓ 예) 주택 가격이 주택의 크기로만 결정된다고 해보면,

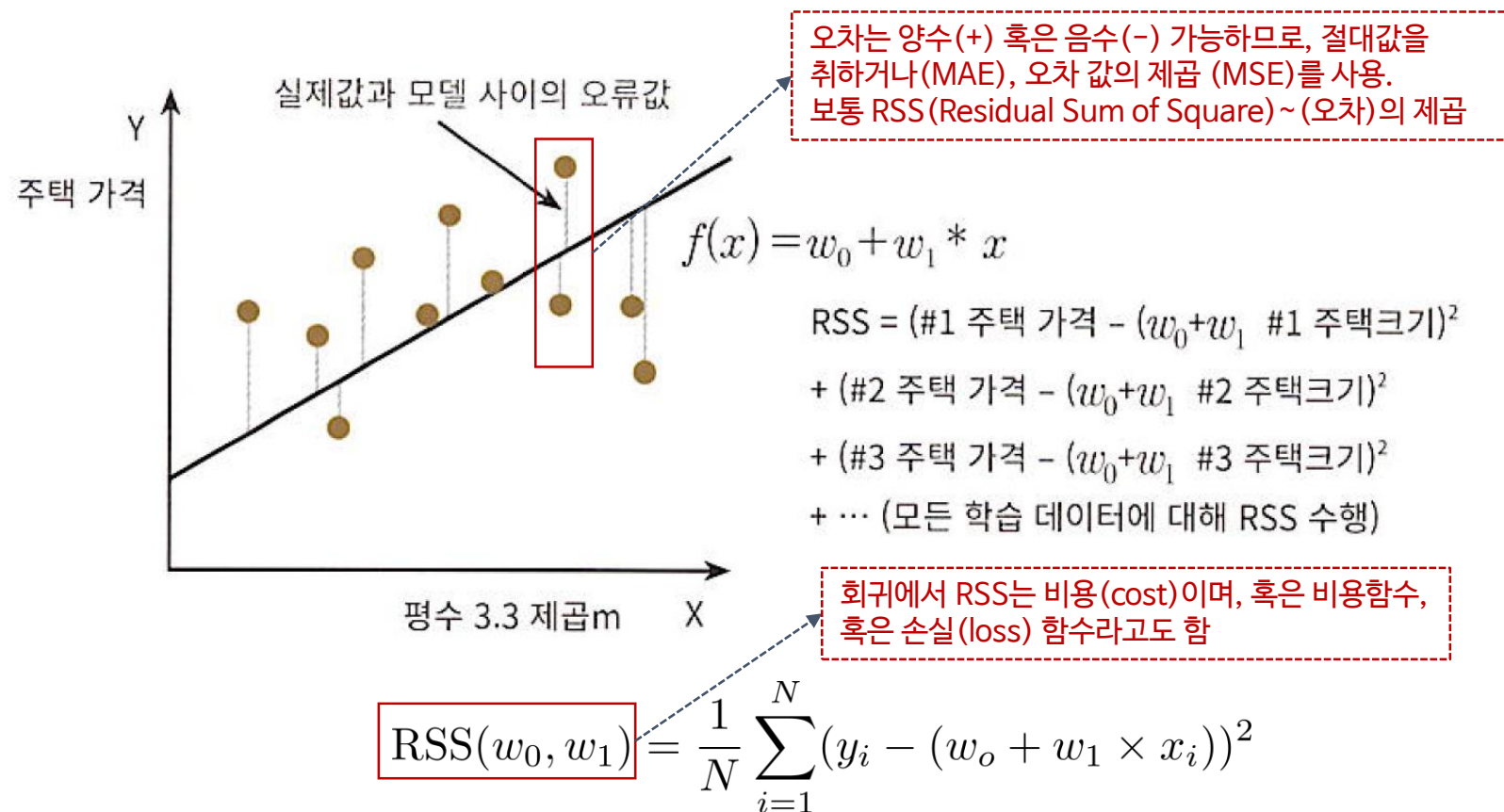
## 대부분 알고리즘은 블랙박스처럼 사용 가능, 하지만, 기본적인 모델이 작동하는 방식을 이해해야 함



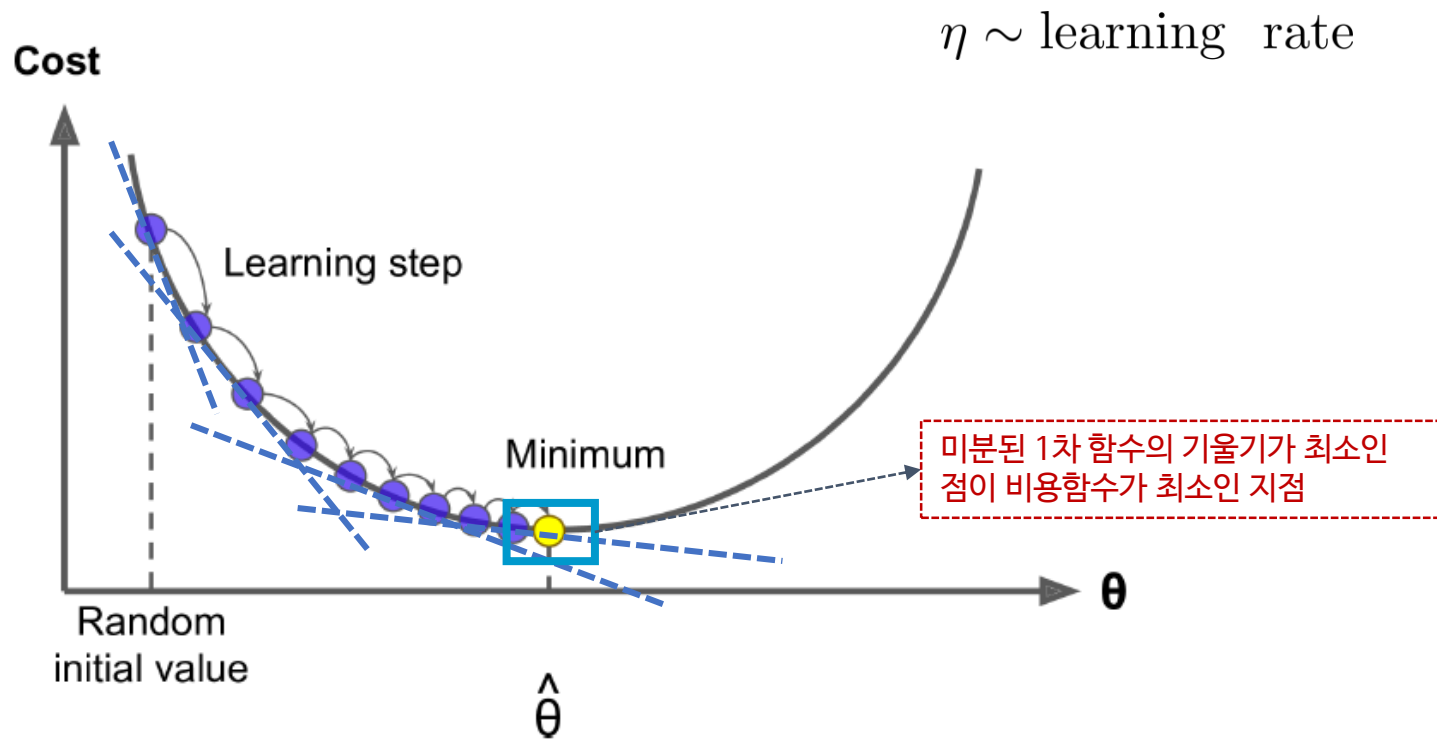
Sir Francis Galton (1822 ~ 1911)

- 단순 선형회귀를 통한 회귀의 이해 : 단순 선형회귀는 1개의 독립변수, 1개의 종속변수

최적의 회귀 모델은 전체 데이터의 잔차(오차) 합이 최소가 되는 모델을 만드는 것임!



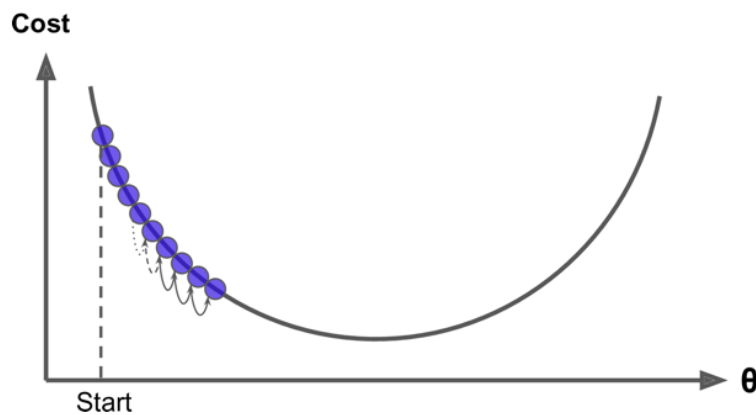
- 경사 하강법 (Gradient Descent)



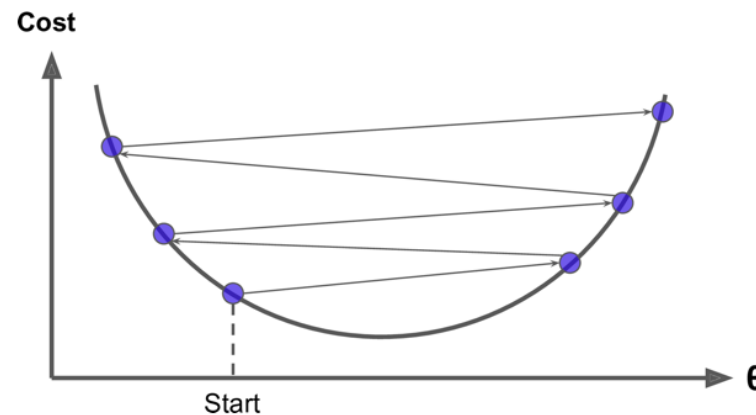
## • 학습률

$$w = w - \boxed{\eta} \frac{2}{N} \sum_{i=1}^N (\text{real}_i - \text{pred}_i)$$

↓  
학습률 도입  $\eta \sim \text{learning rate}$



a) too small

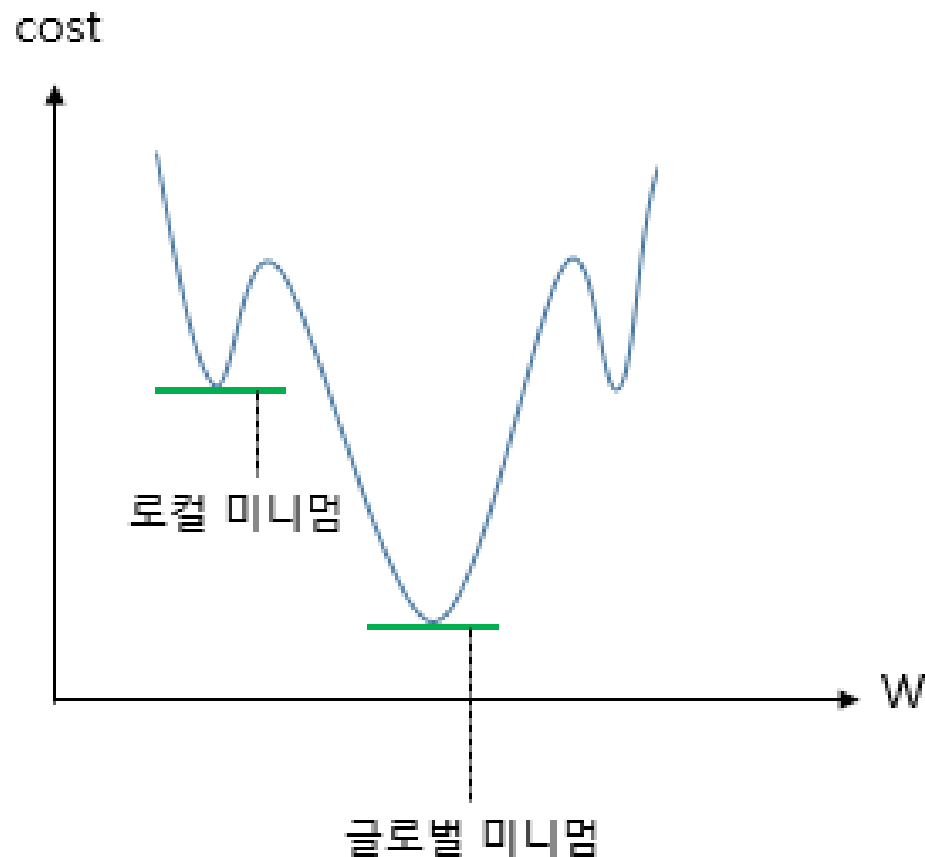


a) too big



## 로컬미니мум의 위험

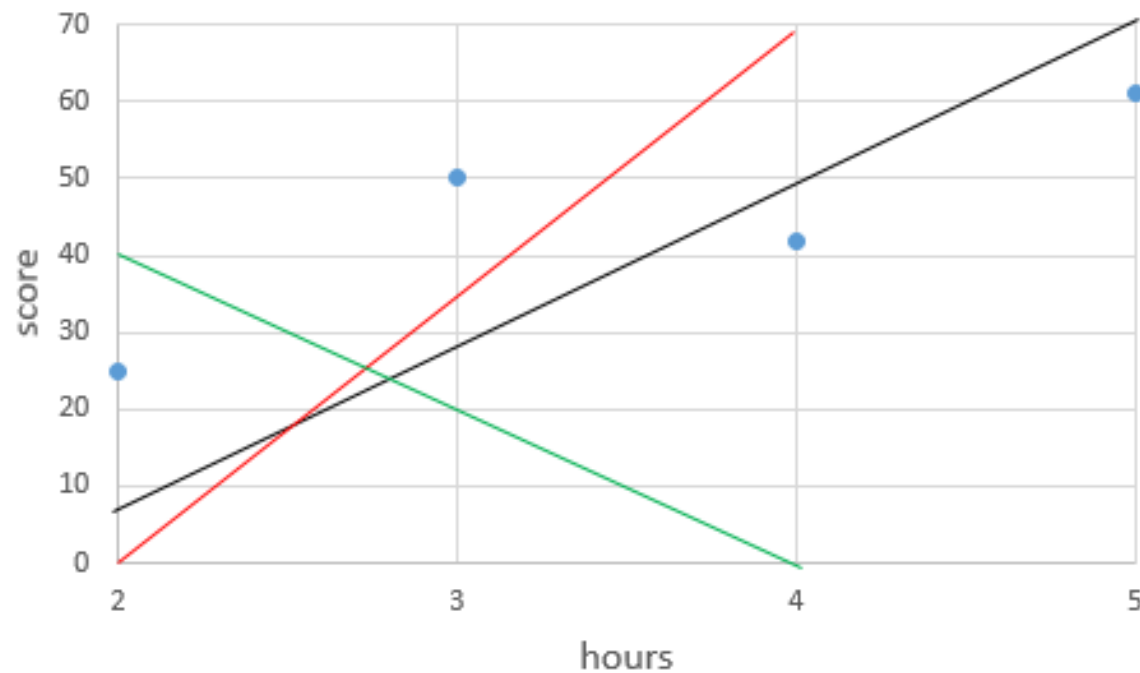
- ✓ 로지스틱 회귀 또한 경사 하강법을 사용하여 가중치를 찾아내지만, 비용 함수로는 평균 제곱 오차를 사용하지 않습니다.
- ✓ 이유는 시그모이드 함수에 비용 함수를 평균 제곱 오차로 하여 그래프를 그리면 다음과 비슷한 형태가 되기 때문입니다.



- 어떤 학생의 공부 시간에 따라서 얻은 점수 데이터
- 예측을 위한 선형 가설을 세우자

$$H(x) = Wx + b$$

hours( $x$ )	score( $y$ )
2	25
3	50
4	42
5	61

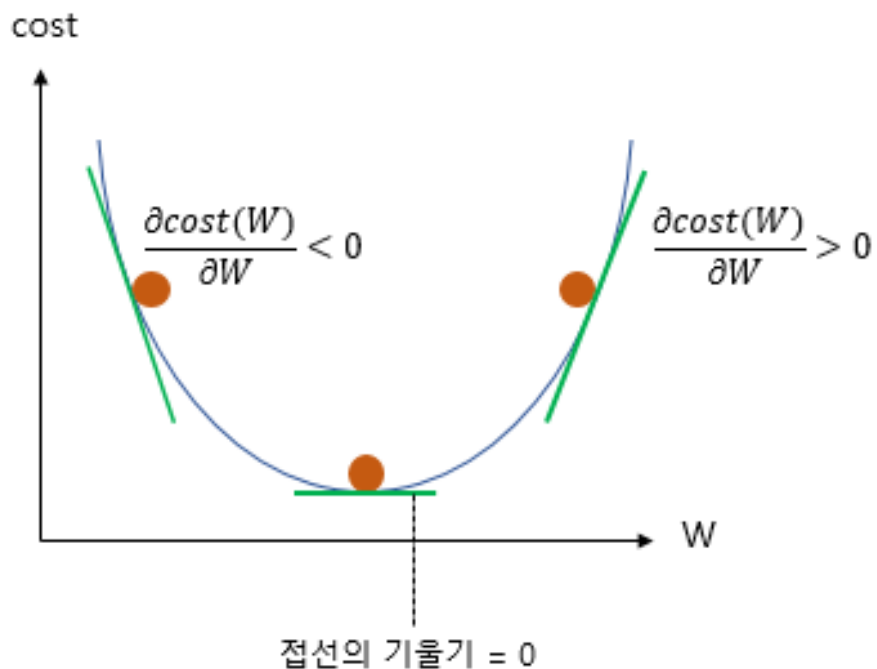


- 옵티마이저(Optimizer) 또는 최적화 알고리즘

✓머신 러닝 학습은 결국 비용 함수를 최소화하는 가중치, 편향을 찾기 위한 작업을 수행

$$w_{\text{updated}} \approx w - \alpha \cdot \frac{\partial \text{cost}}{\partial w}$$

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



# 선형회귀 실습

Lab1: 케라스로 구현하는 선형 회귀

- 케라스로 모델을 만드는 기본적인 형식
- `model = keras.models.Sequential()`
  - ✓ Sequential로 모델을 이라는 이름을 만들고
- `model.add(keras.layers.Dense(1, input_dim=1))`
  - ✓ add를 통해서 필요한 사항을 추가해 나간다.
  - ✓ 첫 인자 1은 출력의 차원
  - ✓ 두 번째 인자 input\_dim은 입력의 차원을 정의

```
[2] import tensorflow as tf
import numpy as np
print(tf.__version__)
```

2.8.0

```
[3] from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers
```

```
[4] X=np.array([1,2,3,4,5,6,7,8,9])
# 공부하는 시간
y=np.array([12,25,50,42,61, 67, 79, 85, 90])
# 각 공부하는 시간에 맵핑되는 성적
```

```
[5] model = Sequential()
```

```
model.add(Dense(1, input_dim=1, activation='linear'))
```

```
[6] # sgd는 경사 하강법을 의미.  
# 학습률(learning rate, lr)은 0.01.  
sgd = optimizers.SGD(lr=0.01)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102:  
super(SGD, self).__init__(name, **kwargs)
```



## 회귀에서

- 손실함수는 오차 MSE를 주로 사용한다.

```
[7] # 손실 함수(Loss function)은 평균제곱오차 mse를 사용합니다.  
model.compile(optimizer=sgd , loss='mse', metrics=['mse'])
```

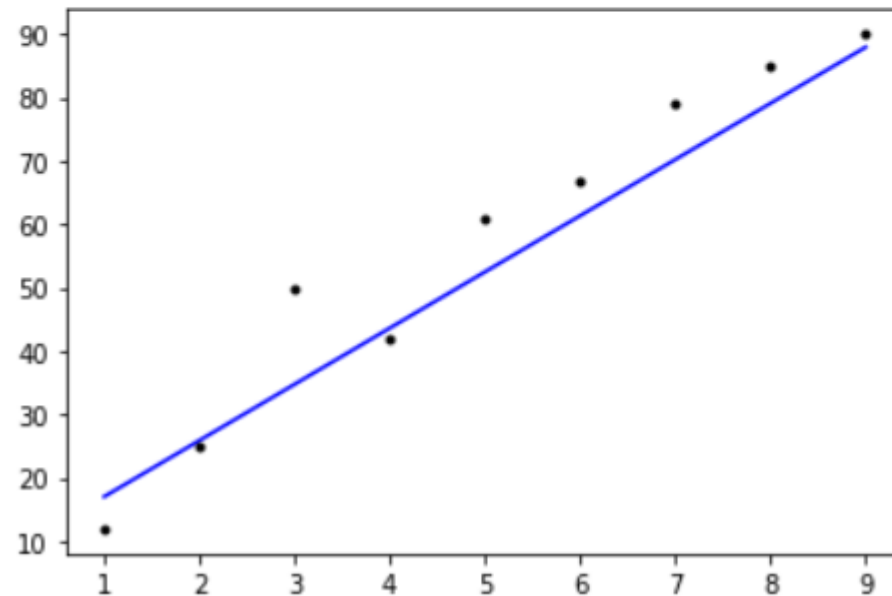
```
[8] # 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 300번 시도합니다.  
history=model.fit(X,y, batch_size=1, epochs=30, shuffle=False)
```

```
%matplotlib inline
```

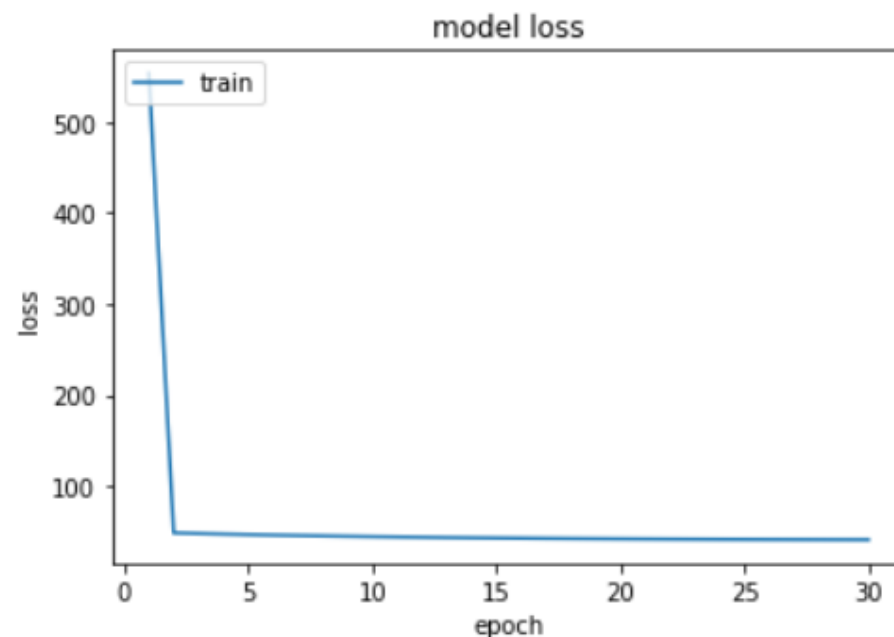
```
import matplotlib.pyplot as plt
```

```
plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x7fda5a760a90>,  
 <matplotlib.lines.Line2D at 0x7fdad0150490>]
```



```
▶ epochs = range(1, len(history.history['mse']) + 1)
plt.plot(epochs, history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



# 비선형회귀 실습

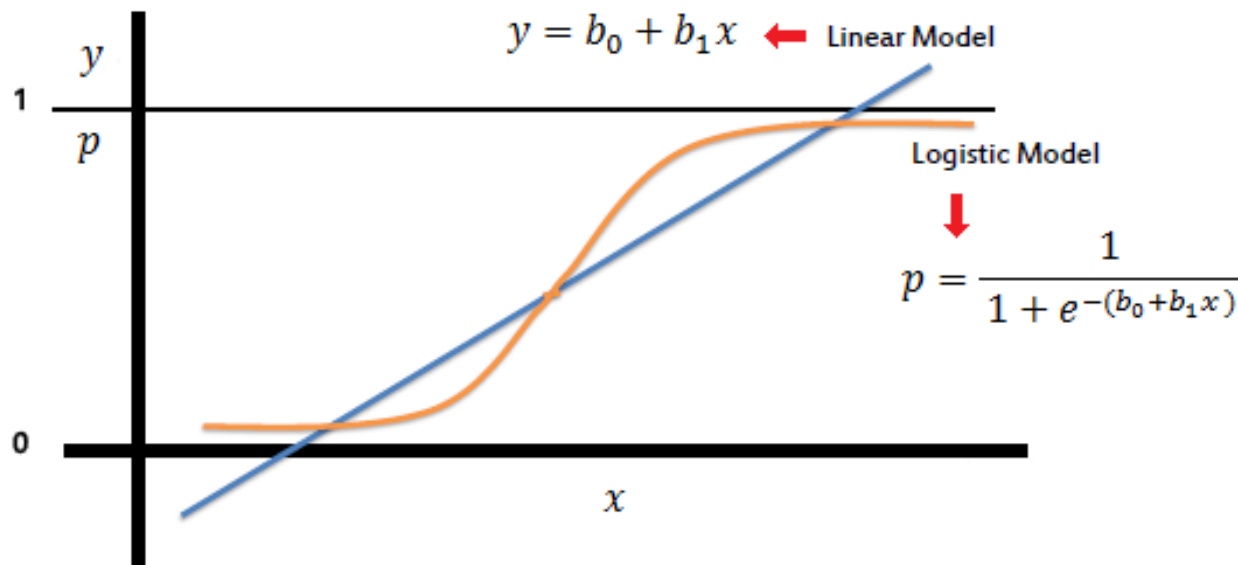
케라스로 구현하는 비선형 회귀

## 로지스틱 회귀 개요

- ✓ 일상 속 많은 문제 중에서는 두 개의 선택지 중에서 정답을 고르는 문제가 많다.
- ✓ 예를 들어 시험을 봤는데 이 시험 점수가 합격인지 불합격인지가 궁금할 수도 있고,
- ✓ 어떤 메일을 받았을 때 이게 정상 메일인지 스팸 메일인지를 분류하는 문제

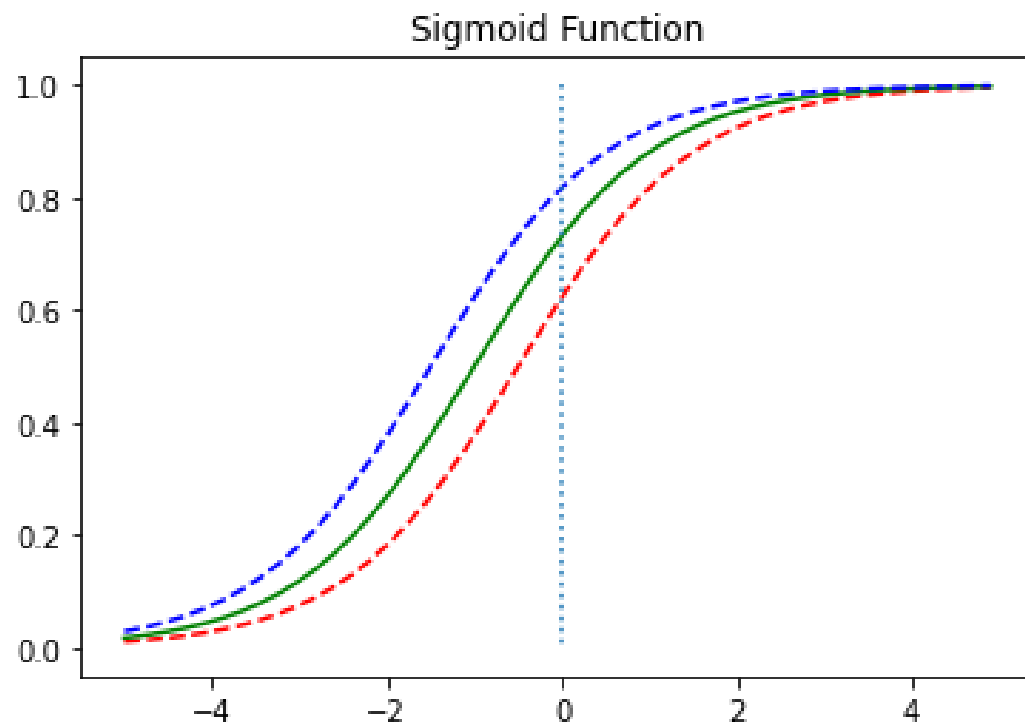
## 둘 중 하나를 결정하는 문제를 이진 분류(Binary Classification)라고 합니다.

- ✓ 이런 문제를 풀기 위한 알고리즘으로 로지스틱 회귀(Logistic Regression)가 있다.



## 시그모이드 함수(Sigmoid function)

$$H(X) = \frac{1}{1+e^{-(Wx+b)}} = \text{sigmoid}(Wx+b) = \sigma(Wx+b)$$



# x + 0.5

# x + 1.5

e=2.718281... 자연 상수

- 목적 함수(objective function)

$$J(W) = \frac{1}{n} \sum_{i=1}^n \text{cost} (H(x^{(i)}), y^{(i)})$$

$$H(X) = \frac{1}{1+e^{-(Wx+b)}} = \sigma(Wx+b)$$

- 크로스 엔트로피 함수 (Cross Entropy)

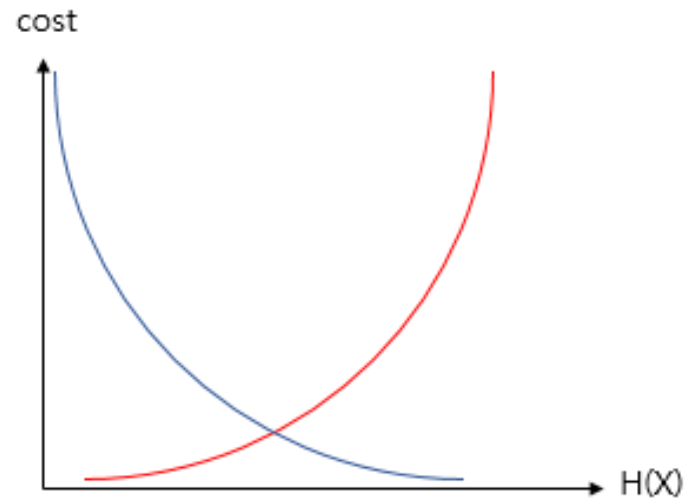
✓ 로지스틱 회귀에서 찾아낸 비용함수를 말한다

$$\text{if } y = 1 \rightarrow \text{cost} (H(x), y) = -\log(H(x))$$

$$\text{if } y = 0 \rightarrow \text{cost} (H(x), y) = -\log(1 - H(x))$$

$$J(W) = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log H(x^{(i)}) + (1 - y^{(i)}) \log(1 - H(x^{(i)}))]$$

$$\text{cost} (H(x), y) = -[y \log H(x) + (1 - y) \log(1 - H(x))]$$

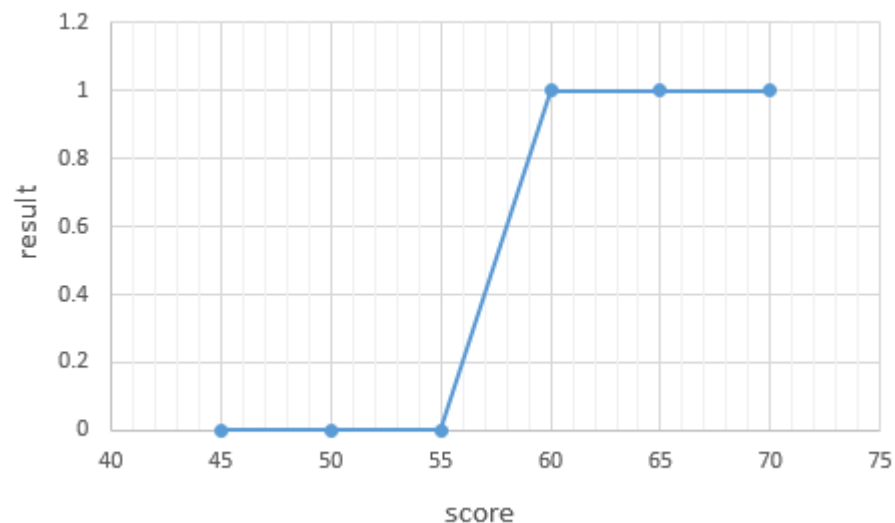


- 예제: 학생들이 시험 성적에 따라서 합격, 불합격이 기재된 데이터가 있다고 가정

✓ 시험 성적이  $x$ 라면, 합불 결과는  $y$ 입니다.

✓ 이 시험의 커트라인은 공개되지 않았는데 이 데이터로부터 특정 점수를 얻었을 때의 합격, 불합격 여부를 판정

score( $x$ )	result( $y$ )
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격





```
[1] import numpy as np
```

```
[2] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import optimizers
```

```
[3] X=np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50])
    y=np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

```
[4] model=Sequential()
```

```
[5] model.add(Dense(1, input_dim=1, activation='sigmoid'))
```

```
[6] sgd=optimizers.SGD(lr=0.01)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102: UserWarning: The  
super(SGD, self).__init__(name, **kwargs)
```

- 회귀의 손실함수는 오차(MSE)를 사용해야 하지만, 로지스틱은 특별히 Accuracy를 자주 쓴다.

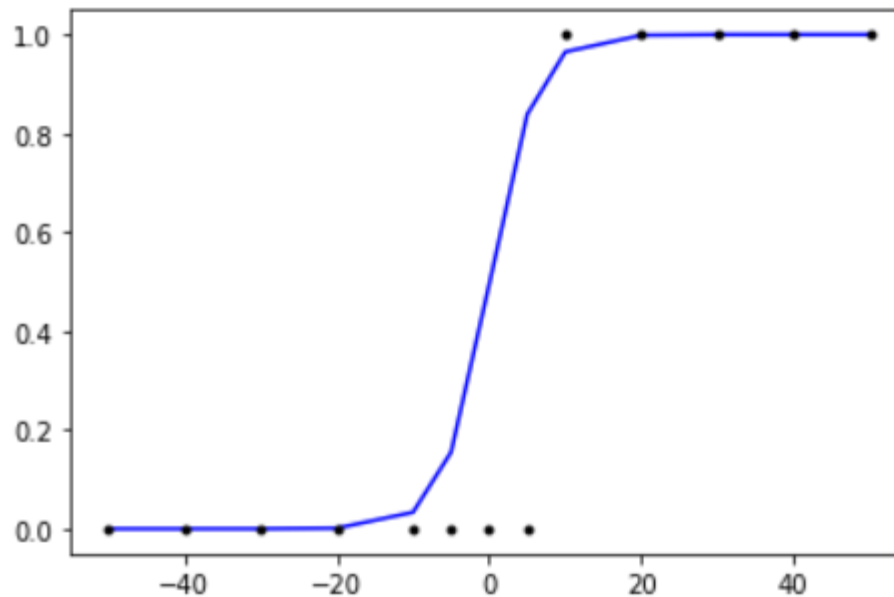
```
[7] model.compile(optimizer=sgd ,  
                  loss='binary_crossentropy',metrics=['binary_accuracy'])
```

▶ # 옵티마이저는 경사하강법 sgd를 사용합니다.  
# 손실 함수(Loss function)는 binary\_crossentropy를 사용합니다.  
history = model.fit(X,y, epochs=20, shuffle=False)

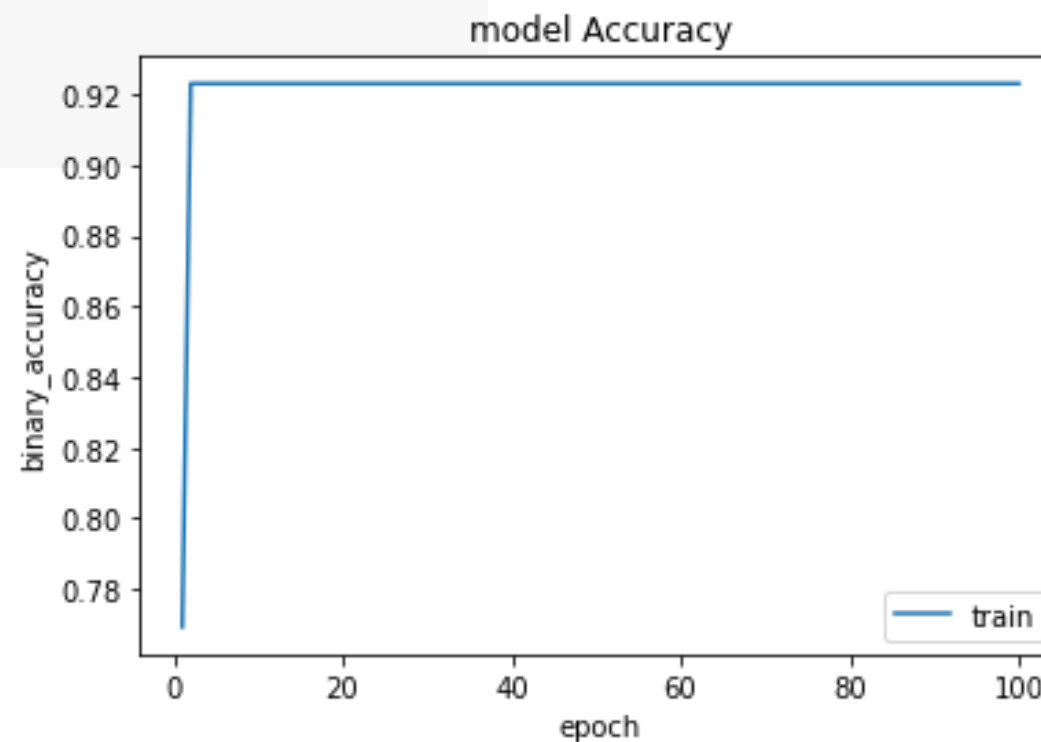
```
[9] import matplotlib.pyplot as plt
    %matplotlib inline
```

```
▶ plt.plot(X, model.predict(X), 'b', X, y, 'k.')
```

```
☞ [<matplotlib.lines.Line2D at 0x7f1c7c723950>,
    <matplotlib.lines.Line2D at 0x7f1c7af7da10>]
```



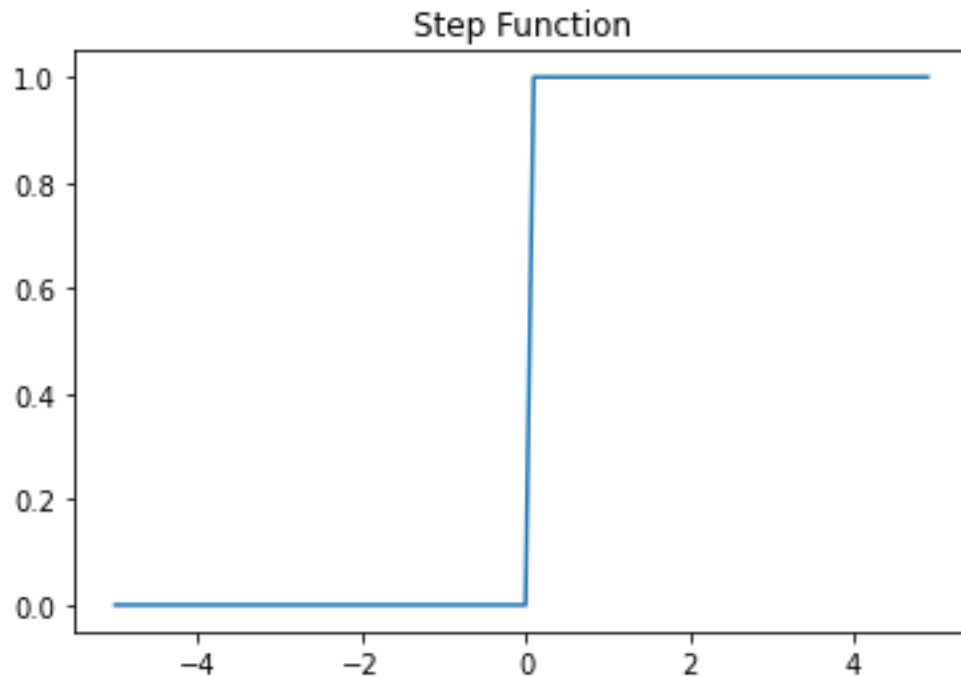
```
▶ epochs = range(1,
                  len(history.history['binary_accuracy']) + 1)
plt.plot(epochs, history.history['binary_accuracy'])
plt.title('model Accuracy')
plt.ylabel('binary_accuracy')
plt.xlabel('epoch')
plt.legend(['train'])
plt.show()
```



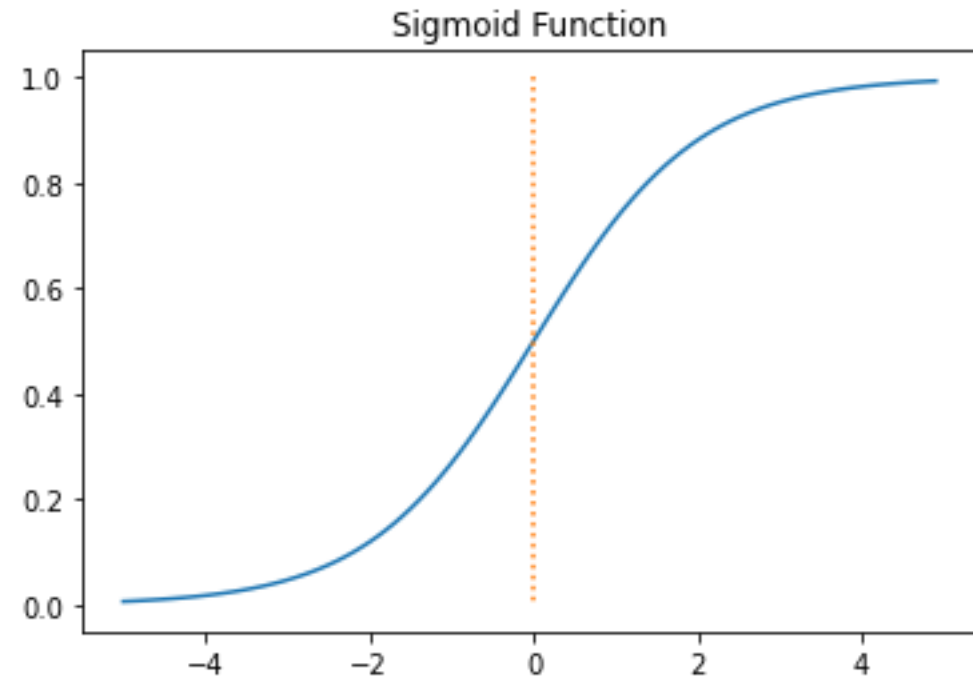
# 활성함수 실습

Softmax, Tanh(x), RuLU 등 만들어 보자

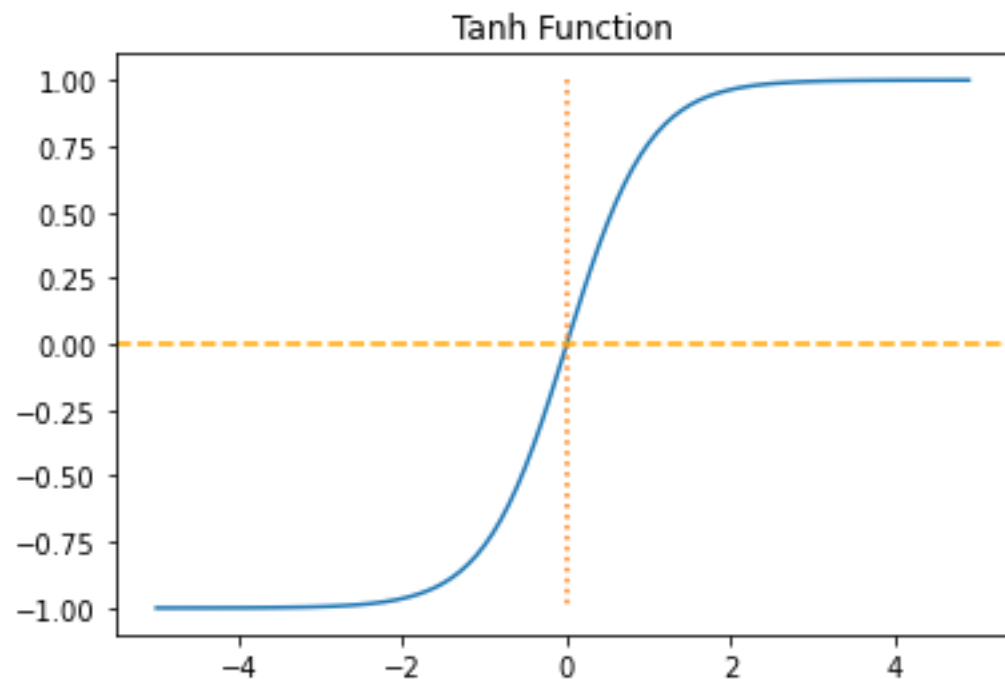
```
def step(x):  
    return np.array(x > 0, dtype=np.int)
```



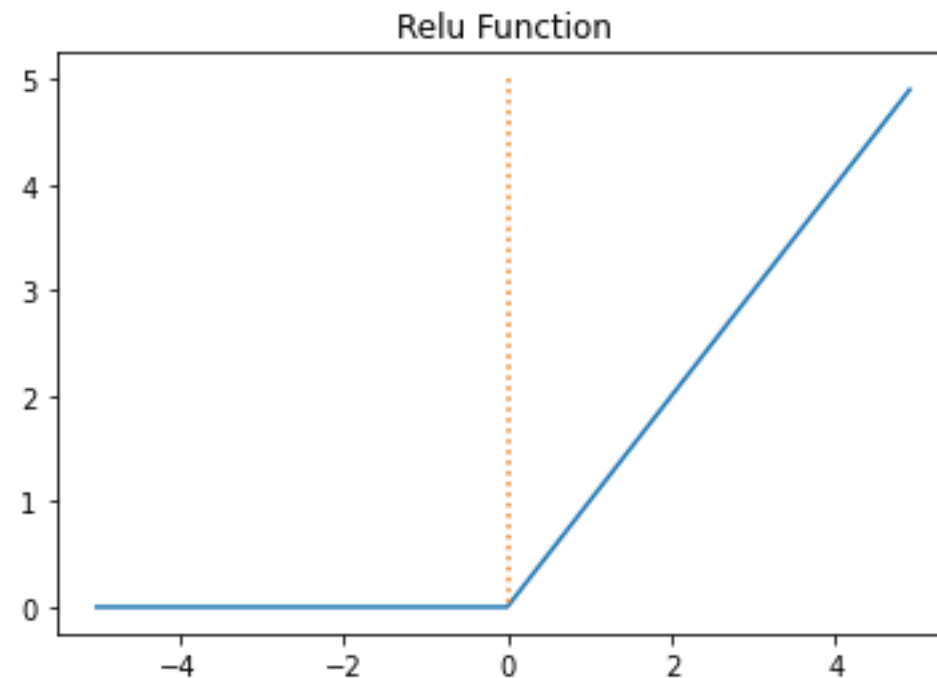
```
def sigmoid(x):  
    return 1/(1+np.exp(-x))
```



`y = np.tanh(x)`



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

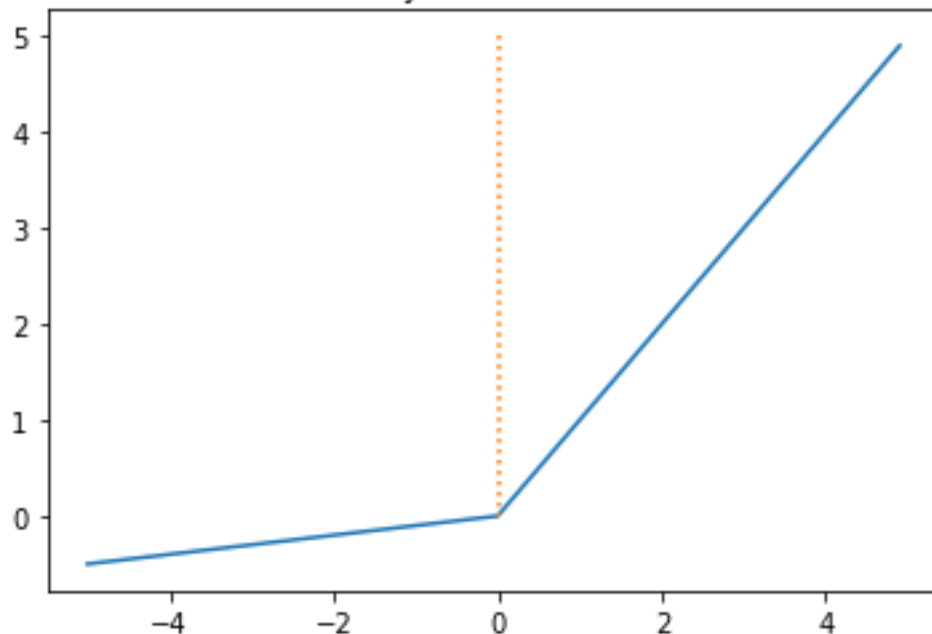


```
def leaky_relu(x):  
    return np.maximum(a*x, x)
```

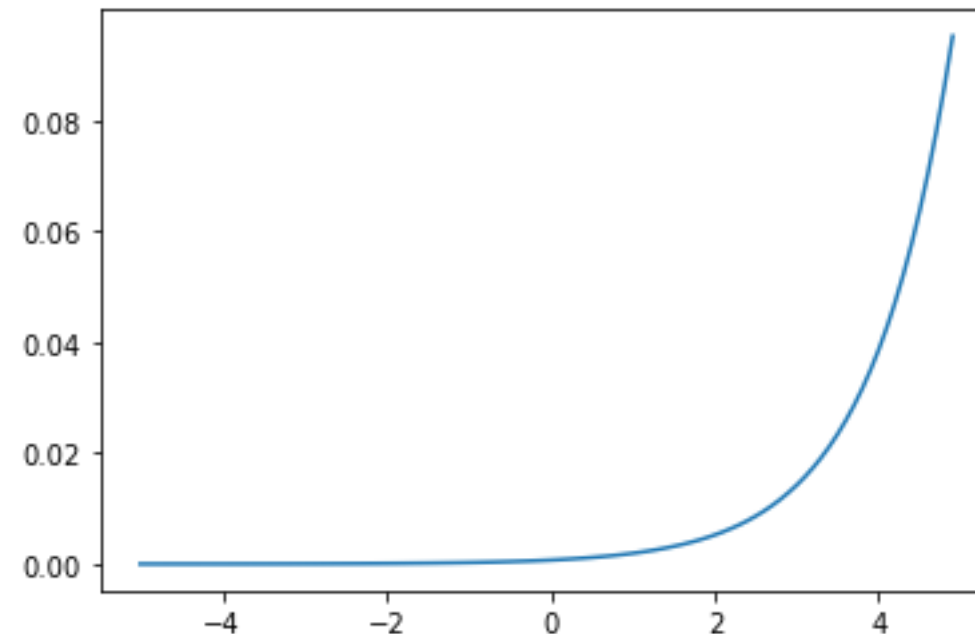
$y = \text{np.exp}(x) / \text{np.sum}(\text{np.exp}(x))$

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \quad \text{for } i = 1, 2, \dots, k$$

Leaky ReLU Function



Softmax Function





2022

Korea Institute of Science  
and Technology Information

TRUST  
KISTL

