클라우드 컴퓨팅과 AI서비스
(9주차)

융합학과 권오영

oykwon@koreatech.ac.kr

학습내용

- 구글티쳐블머신
- scikit learn (sklearn)
  - 기계학습패키지
- 신경망
- streamlit 활용

구글 티쳐블머신

Teachable Machine

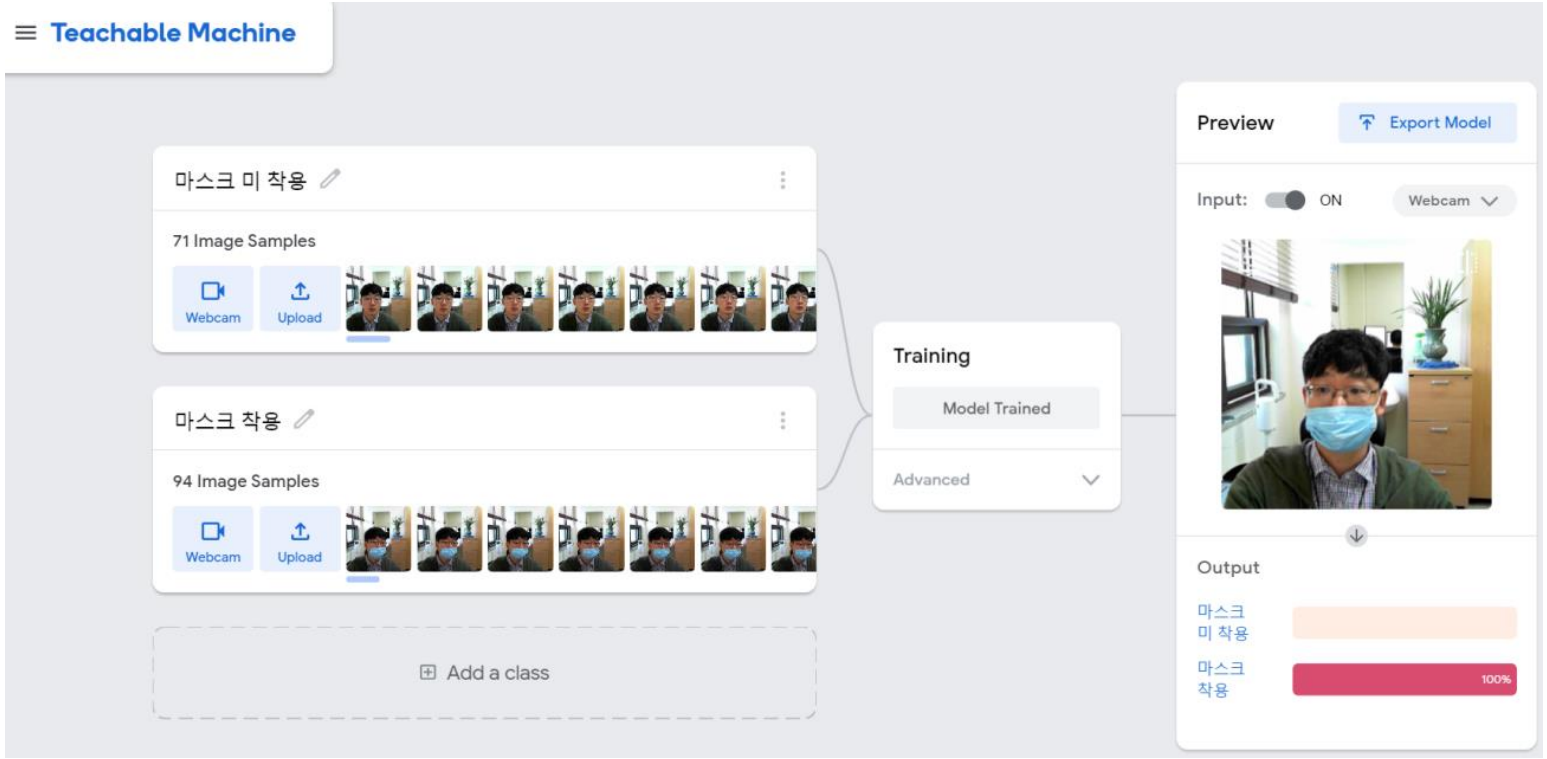
- ❖ 코딩없이 응용제작 (https://teachablemachine.withgoogle.com/)
- ❖ Teachable Machine을 이용한 인공지능 서비스 만들기 예제?
  (https://www.youtube.com/watch?v=UPgxnGC8oBU)
  - ✓ 2초 딜레이후에 6초간 동작인식시키고 -> 길게눌러 입력받기로 변환
  - ✓ 훈련을 시킨후에 (시간이 걸림 중간에 말을 하던지 생략)
  - ✓ Export 해서 다른 프로그램에 사용
  - ✓ 쉽게 만들 수 있음 2 ~ 3분 정도

- ❖ Youtube The coding train
  https://www.youtube.com/user/shiffman
  에서 teachable machine 을 검색

- ❖ 이미지, 소리, 포즈 인식

Teachable machine 서비스 제작

- 각자 아이디어를 내어서 서비스 만들어 보기
- 예시) 마스크 착용 여부 판단



KOREATECH 한국기술교육대학교

SCIKIT-LEARN

Scikit-learn (sklearn) 소개

- ❖ 기계학습 라이브러리
  - classification, regression, clustering, 차원축소 등 지원
- ❖ Numpy, Scipy, Matplotlib 를 활용하여 구성

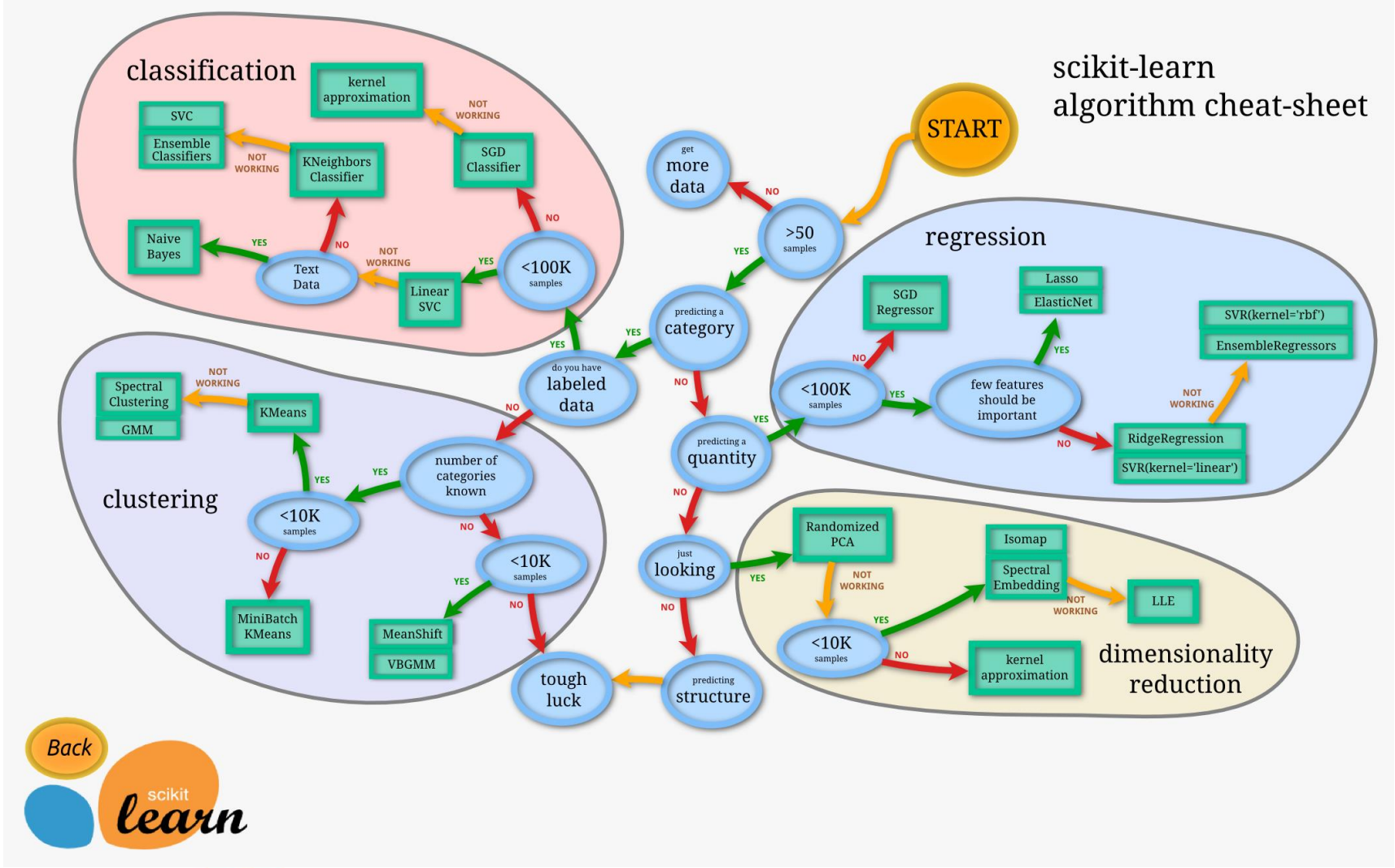
- ❖ 설치 pip install –U scikit-learn
- ❖ 데이터 모델링에 중점을 두고 라이브러리 구성

- ❖ Supervised Learning Algorithms
  - Linear Regression, Support Vector Machine(SVM), Decision Tree 등
- ❖ Unsupervised Learning Algorithms
  - clustering, factor analysis, PCA(Principal Component Analysis)

Modelling

- 데이터 준비
  (데이터 전처리)
- 데이터 로딩
- 데이터 분할
  (train, test;
  train, test, validation)
- 모델 선정 및 학습
  (estimator)
- 활용 (예측에 사용)

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html



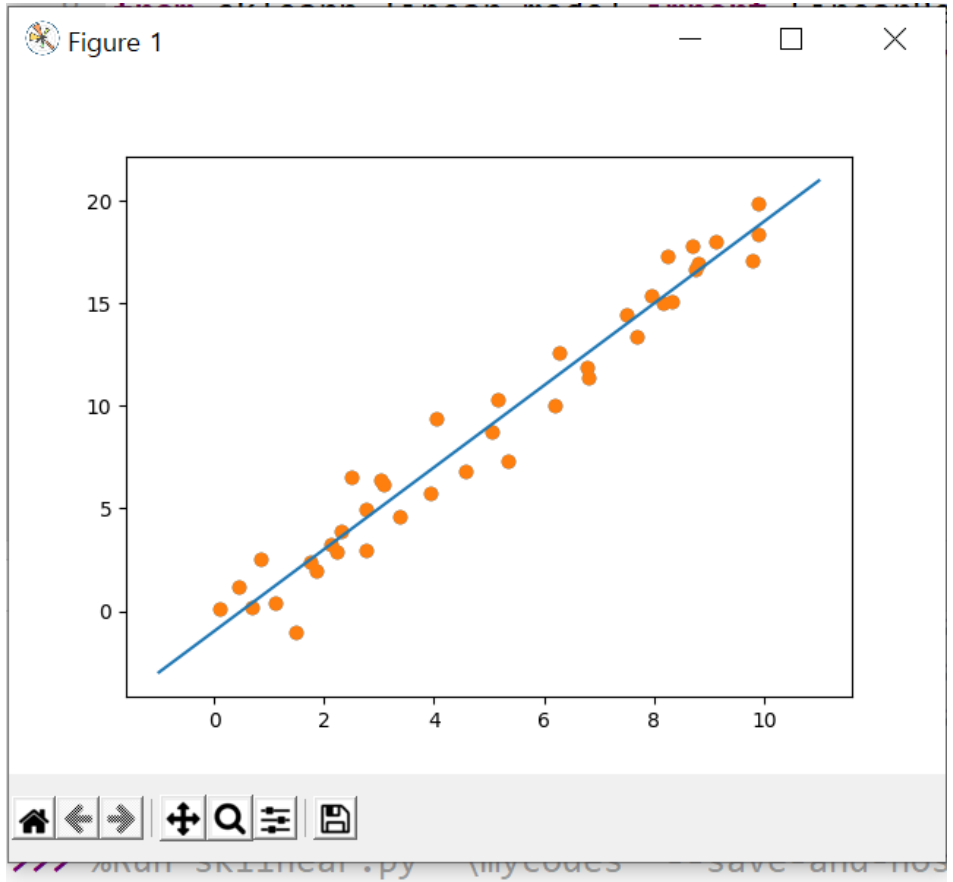
KOREATECH 한국기술교육대학교

Steps in using Estimator API

- estimator: 데이터로부터 학습하는 객체
- **Step 1: Choose a class of model**
  - It can be done by importing the appropriate Estimator class from Scikit-learn.
- **Step 2: Choose model hyperparameters**
  - It can be done by instantiating the class with desired values.
- **Step 3: Arranging the data**
  - to arrange the data into features matrix (X) and target vector(y).
- **Step 4: Model Fitting**
  - to fit the model to your data. (calling **fit()** method)
- **Step 5: Applying the model**
  - apply it to new data.
  - for supervised learning, use **predict()** method
  - for unsupervised learning, use **predict()** or **transform()**

Supervised Learning Example

❖ simple linear regression

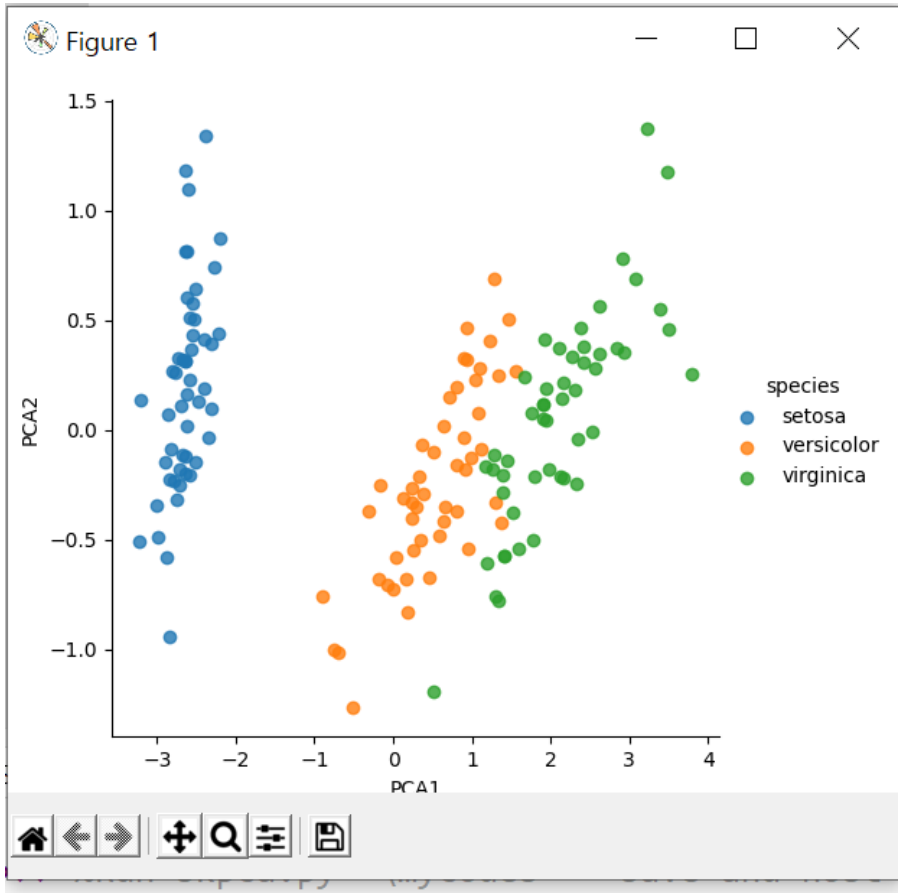



```
LinearRegression()
[1.99839352]
-0.989545945777504
```

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
rng = np.random.RandomState(35)
x = 10*rng.rand(40)
y = 2*x-1+rng.randn(40)
plt.scatter(x,y)
plt.show()
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
X = x[:, np.newaxis]
print(model.fit(X, y))
print(model.coef_)
print(model.intercept_)
xfit = np.linspace(-1, 11)
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.show()
```

Unsupervised Learning Example

❖ 차원축소 방법




```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
iris = sns.load_dataset('iris')
X_iris = iris.drop('species', axis = 1)
print(X_iris.shape)
y_iris = iris['species']
print(y_iris.shape)
from sklearn.decomposition import PCA
model = PCA(n_components=2)
print(model.fit(X_iris))
X_2D = model.transform(X_iris)
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False)
plt.show()
```

모델링과정

Modelling

❖ Dataset Loading

- Features: 입력 데이터
  - ✓ Feature matrix: It is the collection of features, in case there are more than one.
  - ✓ Feature Names: It is the list of all the names of the features.
- Response: 출력
  - ✓ Response Vector: It is used to represent response column. (We have just one response column.)
  - ✓ Target Names: It represent the possible values taken by a response vector

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("₩nFirst 10 rows of X:₩n", X[:10])
```

```
[[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]]
```

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
```

Modelling

❖ 데이터 셋의 분할

- training set (70%) : testing set (30%)
- 150 * 0.7 = 105

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data    # feature matrix
y = iris.target   # response vector
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(105, 4)
(45, 4)
(105,)
(45,)
```

Modeling

❖ Train the model

- scikit-learn에서 제공하는 ML 알고리즘을 활용하여 학습 (예. KNN: K nearest neighbors)

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
classifier_knn = KNeighborsClassifier(n_neighbors=3)
classifier_knn.fit(X_train, y_train)
y_pred = classifier_knn.predict(X_test)
# Finding accuracy by comparing actual response values(y_test)with predicted response value(y_pred)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
# Providing sample data and the model will make prediction out of that data
sample = [[5, 5, 3, 2], [2, 4, 3, 5]]
preds = classifier_knn.predict(sample)
pred_species = [iris.target_names[p] for p in preds]
print("Predictions:", pred_species)
```

```
Accuracy: 0.9833333333333333
Predictions: ['versicolor', 'virginica']
```

정확도 = (올바르게 예측한 샘플수)/(전체 샘플수)
= (TP+TN)/(TP+TN+FP+FN)

Modelling (모델저장)

❖ Model Persistence

- 학습된 모델의 보관

❖ 모델 dump

```
from sklearn.externals import joblib
joblib.dump(classifier_knn, 'iris_classifier_knn.joblib')
```

❖ 저장된 모델의 load

```
joblib.load('iris_classifier_knn.joblib')
```

Modelling (전처리)

❖ 입력데이터의 전처리

- 획득한 raw data를 학습(인공지능모델)에 활용할 수 있도록 데이터 가공이 필요

❖ 이진화(Binarisation)

- 0.5 기준으로 이진화

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[2.1, -1.9, 5.5],
                       [-1.5, 2.4, 3.5],
                       [0.5, -7.9, 5.6],
                       [5.9, 2.3, -5.8]])
data_binarized = preprocessing.Binarizer(threshold=0.5).transform(input_data)
print("₩nBinarized data:₩n", data_binarized)
```

```
Binarized data:
[[ 1. 0. 1.]
[ 0. 1. 1.]
[ 0. 0. 1.]
[ 1. 1. 0.]]
```

Modelling (전처리)

❖ Mean removal

```
import numpy as np
from sklearn import preprocessing
input_data = np.array([[2.1, -1.9, 5.5],
                       [-1.5, 2.4, 3.5],
                       [0.5, -7.9, 5.6],
                       [5.9, 2.3, -5.8]])
#displaying the mean and the standard deviation of the input data
print("Mean =", input_data.mean(axis=0)) # 세로축
print("Stddeviation = ", input_data.std(axis=0))
#Removing the mean and the standard deviation of the input data
data_scaled = preprocessing.scale(input_data)
print(data_scaled)
print("Mean_removed =", data_scaled.mean(axis=0))
print("Stddeviation_removed =", data_scaled.std(axis=0))
```

```
Mean = [ 1.75  -1.275  2.2  ]
Stddeviation =  [2.71431391 4.20022321 4.69414529]
[[ 0.12894603 -0.14880162  0.70300338]
 [-1.19735598  0.8749535   0.27694073]
 [-0.46052153 -1.57729713  0.72430651]
 [ 1.52893149  0.85114524 -1.70425062]]
Mean_removed = [1.11022302e-16 0.00000000e+00 0.00000000e+00]
Stddeviation_removed = [1. 1. 1.]
```

Modelling (전처리)

❖ Scaling (0 ~ 1 사이 값으로 정리)

```
import numpy as np
from sklearn import preprocessing
input_data = np.array([[2.1, -1.9, 5.5],
                       [-1.5, 2.4, 3.5],
                       [0.5, -7.9, 5.6],
                       [5.9, 2.3, -5.8]])
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print ("₩nMin max scaled data:₩n", data_scaled_minmax)   # 컬럼기준으로 해석
```

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

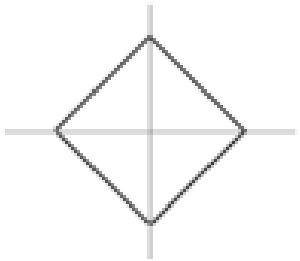
```
Min max scaled data:
[[ 0.48648649  0.58252427    0.99122807]
 [ 0.          1.            0.81578947]
 [ 0.27027027  0.            1. ]
 [ 1.          0.99029126    0. ]]
```

Modelling (전처리)

❖ Normalizaiton (정규화)

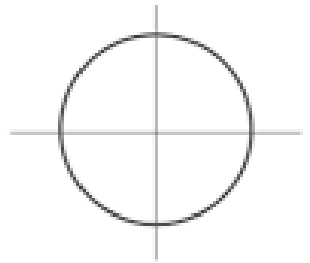
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



```
import numpy as np
from sklearn import preprocessing
input_data = np.array([[2.1, -1.9, 5.5],
                       [-1.5, 2.4, 3.5],
                       [0.5, -7.9, 5.6],
                       [5.9, 2.3, -5.8]])
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
print("₩nL1 normalized data:₩n", data_normalized_l1)    # 가로축을 기준으로 값을 정렬
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("₩nL1 normalized data:₩n", data_normalized_l2)    # 예) l1 = 2.1/(2.1+1.9+5.5)
                                                        #     l2 = 2.1/sqrt(2.1*2.1 + 1.9*1.9 + 5.5*5.5)
```

```
L1 normalized data:
[[ 0.22105263 -0.2  0.57894737]
[-0.2027027  0.32432432 0.47297297]
[ 0.03571429 -0.56428571 0.4 ]
[ 0.42142857 0.16428571 -0.41428571]]
```

```
L2 normalized data:
[[ 0.33946114 -0.30713151 0.88906489]
[-0.33325106 0.53320169 0.7775858 ]
[ 0.05156558 -0.81473612 0.57753446]
[ 0.68706914 0.26784051 -0.6754239 ]]
```

MNIST (숫자인식)

MNIST

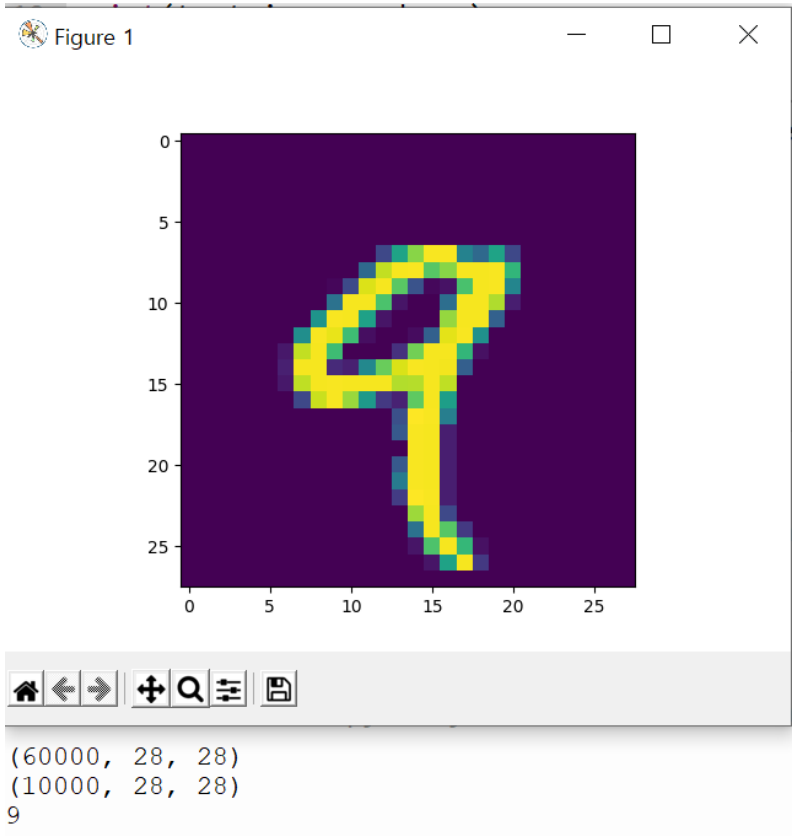
- The MNIST dataset is a well-known dataset consisting of 28x28 grayscale images. For each image, we know the corresponding digits (from 0 to 9).
  It is available here: http://yann.lecun.com/exdb/mnist/index.html

```
import mnist     # need to install
import matplotlib.pyplot as plt

# Load dataset
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
print(train_images.shape)
print(test_images.shape)

# Pick the fifth image from the dataset (it's a 9)
image, label = train_images[4], train_labels[4]
print(label)

plt.imshow(image)
plt.show()
```



MNIST

❖ KNN

```
import mnist
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Load dataset
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
# preprocessing
train_images = train_images.reshape(-1, 28*28)
test_images = test_images.reshape(-1, 28*28)

clf = KNeighborsClassifier()
#clf.fit(train_images, train_labels)
clf.fit(train_images[:10000], train_labels[:10000])

# Test on the next 100 images:
test_x = test_images[:100]
expected = test_labels[:100].tolist()
print("Compute predictions")
predicted = clf.predict(test_x)
print("Accuracy: ", accuracy_score(expected, predicted))
```

MNIST

❖ Random Forest

```
import mnist
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Load dataset
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
# preprocessing
train_images = train_images.reshape(-1, 28*28)
test_images = test_images.reshape(-1, 28*28)

clf = RandomForestClassifier(n_estimators=100)
clf.fit(train_images[:10000], train_labels[:10000])
# Test on the next 1000 images:
test_x = train_images[10000:11000]
expected = train_labels[10000:11000].tolist()
print("Compute predictions")
predicted = clf.predict(test_x)
print("Accuracy: ", accuracy_score(expected, predicted))
```

MNIST

- Linear Support Vector Classification

```
import mnist
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
# Load dataset
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
# preprocessing
train_images = train_images.reshape(-1, 28*28)
test_images = test_images.reshape(-1, 28*28)

clf = LinearSVC()
clf.fit(train_images[:10000], train_labels[:10000])
# Test on the next 1000 images:
test_x = train_images[10000:11000]
expected = train_labels[10000:11000].tolist()
print("Compute predictions")
predicted = clf.predict(test_x)
print("Accuracy: ", accuracy_score(expected, predicted))
```

베이즈정리

베이즈정리

- 확률을 지식 또는 믿음의 정도를 나타내는 양이라는 관점에서 접근
- 주어진 데이터를 바탕으로 확률을 변경할 수 있다.
- 베이즈 정리

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

  - P(A) 사전확률(prior)로써 사건 B가 발생하기 전에 사건 A의 확률
  - P(A|B) 사후확률(posterior): 사건 B가 발생하여 갱신된 사건 A의 확률
  - P(B|A) 가능도(likelihood; 우도), 사건 A가 발생한 경우 사건 B의 확률
  - P(B) 증거(evidence)

- 확장

$$\begin{aligned} P(A_1|B) &= \frac{P(B|A_1)P(A_1)}{P(B)} \\ &= \frac{P(B|A_1)P(A_1)}{\sum_i P(A_i, B)} \qquad A_i \cap A_j = \emptyset \\ &= \frac{P(B|A_1)P(A_1)}{\sum_i P(B|A_i)P(A_i)} \qquad A_1 \cup A_2 \cup \cdots = \Omega \end{aligned}$$

베이즈 정리

- ❖ 검사 시약 문제: 특정 질병을 검사하는 시약으로 특정 질병에 걸리 환자를 대상으로 시약 검사를 하면 99%의 확률로 양성반응을 보인다.
- ❖ 양성반응을 보였지만 실제 병에 걸렸을 확률은?
  - 병에 걸리는 경우 : 사건 D
  - 양성 반응을 보이는 경우 : 사건 S
  - 병에 걸린 사람이 양성 반응을 보이는 경우 : 조건부 사건 $S|D$
  - 양성 반응을 보이는 사람이 병에 걸려 있을 경우 : 조건부 사건 $D|S$
  - 문제
  - $P(S|D) = 0.99$가 주어졌을 때, $P(D|S)$를 구하라.

$$P(D|S) = \frac{P(S|D)P(D)}{P(S)}$$

베이즈정리

❖ 베이즈 정리를 이용하려면 추가 정보가 필요하다.

- 특정질병은 희귀병으로 전체인구의 0.2%가만 걸렸다. P(D) = 0.002
- 시약검사를 했을때 잘못된 양성반응이 나오는 확률이 5%이다. P(S|(1-D)) = 0.05

$$
\begin{aligned}
P(D|S) &= \frac{P(S|D)P(D)}{P(S)} \\
&= \frac{P(S|D)P(D)}{P(S,D) + P(S,D^C)} \\
&= \frac{P(S|D)P(D)}{P(S|D)P(D) + P(S|D^C)P(D^C)} \\
&= \frac{P(S|D)P(D)}{P(S|D)P(D) + P(S|D^C)(1 - P(D))} \\
&= \frac{0.99 \cdot 0.002}{0.99 \cdot 0.002 + 0.05 \cdot (1 - 0.002)} \\
&= 0.038
\end{aligned}
$$

베이즈정리확장

$$P(A|B,C) = \frac{P(B|A,C)P(A|C)}{P(B|C)}$$

$$P(A|B,C,D) = \frac{P(D|A,B,C)P(A|B,C)}{P(D|B,C)}$$

$$P(A,B|C,D) = \frac{P(D|A,B,C)P(A,B|C)}{P(D|C)}$$

베이즈분류모형

❖ 주어진 데이터를 가지고 가능도를 추정 $P(x \mid y=k) = P(x_1, \ldots, x_D \mid y=k)$

- 입력데이터의 차원이 높아지면 가능도 추정이 어려워짐

❖ 모든 차원의 개별 독립변수가 서로 조건부 독립이라면 (navie assumption)

$$P(x_1, \ldots, x_D \mid y=k) = \prod_{d=1}^{D} P(x_d \mid y=k)$$

$$P(y=k \mid x) = \frac{P(x_1, \ldots, x_D \mid y=k)P(y=k)}{P(x)}$$

$$= \frac{\left(\prod_{d=1}^{D} P(x_d \mid y=k)\right) P(y=k)}{P(x)}$$

❖ 가능도를 정규분포로 가정 $P(x_d \mid y=k) = \frac{1}{\sqrt{2\pi\sigma_{d,k}^2}} \exp\left(-\frac{(x_d - \mu_{d,k})^2}{2\sigma_{d,k}^2}\right)$

나이브베이즈 모형

❖ 정규분포 나이브베이즈 모형

- 주어진 데이터가 정규분포라고 생각하고, 데이터에 기반한 평균, 분산을 추정

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
X1 = iris.data
y1 = iris.target

from sklearn.naive_bayes import GaussianNB
model1 = GaussianNB().fit(X1, y1)
print(model1.class_prior_)
print(model1.theta_)
print(model1.sigma_)
y1_pred = model1.predict(X1)

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y1, y1_pred))
from sklearn.metrics import classification_report
print(classification_report(y1, y1_pred))
```

신경망(NEURAL NETWORK)

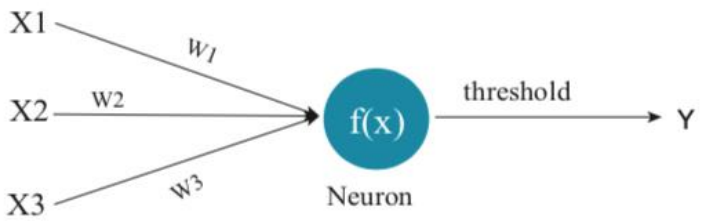
Deep Learning

❖ Deep Learning 이란

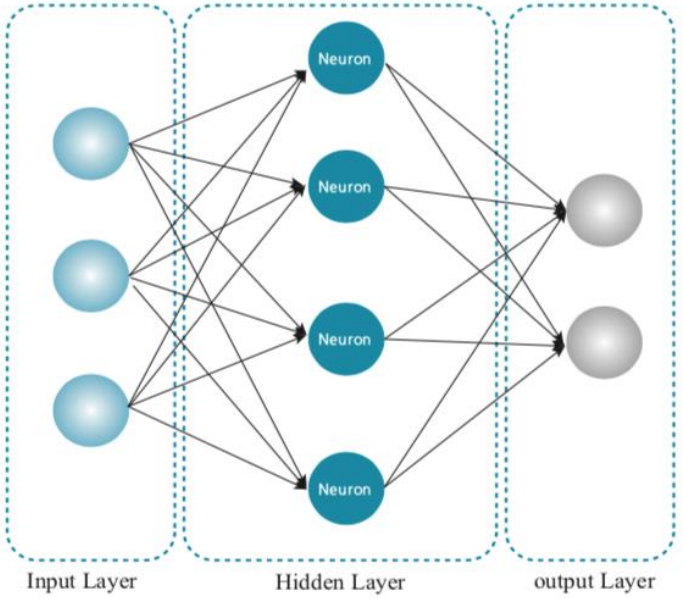
- 여러 층을 가진 인공 신경망(Artificial Neural Network)을 사용하여 머신러닝 학습을 수행
- 딥러닝은 기계가 자동으로 학습하려는 데이터에서 특징을 추출하여 학습

❖ 인공신경망(Artificial Neural Network)

- 인공 신경망은 인간의 신경세포 뉴런(Neuron)과 같은 서로 연결된 뉴런은 서로의 입력신호와 출력 신호를 이용하여 동작함
- 뉴런과 신경망 연결 구조



뉴런 동작



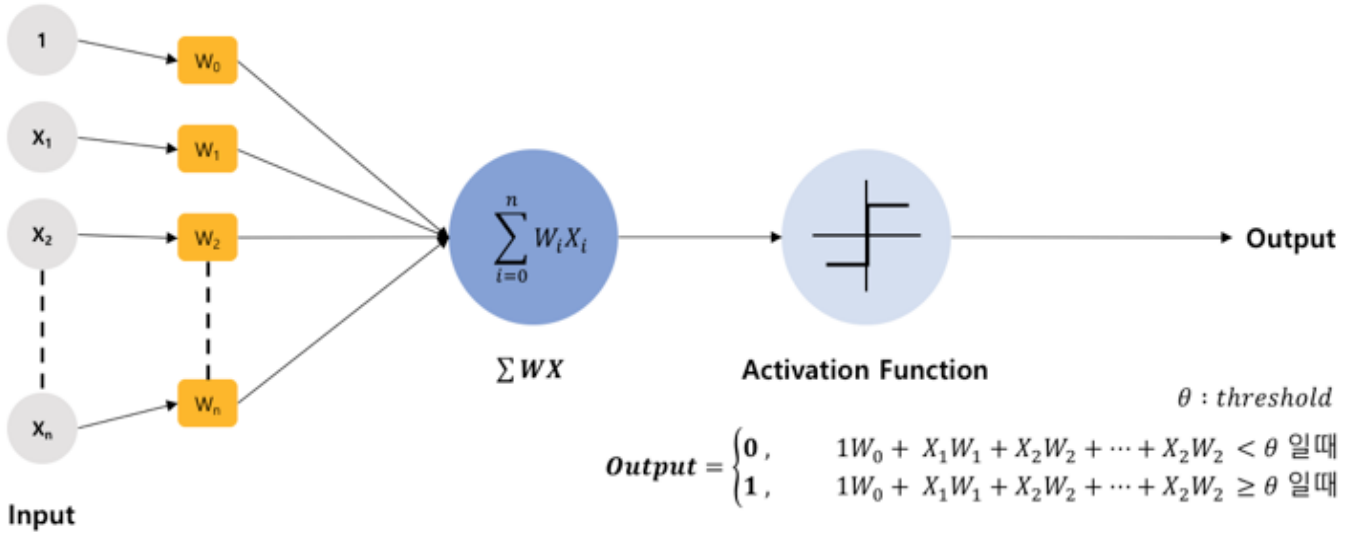
신경망 구성

KOREATECH 한국기술교육대학교

Perceptron

❖ 퍼셉트론(Perceptron)

- 가장 간단한 인공 신경망 구조
- 다수의 신호(Input)를 입력받아서 하나의 신호(Output)를 출력
- 퍼셉트론 동작 순서
  - ✓ 각각의 입력 신호에 부여된 W(Weight)와 계산
  - ✓ 계산 결과의 총합이 활성화 함수(Activation Function)로 입력
  - ✓ 활성화 함수에서는 정해진 임계값(threshold)을 넘었을때 1을 출력 넘지 못한 경우 0 혹은 -1 을 출력
- W값이 크면 해당 신호는 중요한 신호라고 판단하게 됨
- 일반적으로 퍼셉트론에서 사용되는 활성화 함수는 헤비사이드 계단함수(Heaviside Step Function)이 사용됨



Perceptron

❖ 퍼셉트론 결과값에서 임계값

- 활성화 함수에서 사용하는 임계값(threshold)은 θ 로 표현
- $1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 < \theta$ 수식에서 θ 를 -b(bias, 편향)로 치환하여 수식 을 변경

$$Output = \begin{cases} 0, & 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 < \theta \\ 1, & 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 \geq \theta \end{cases} \Rightarrow Output = \begin{cases} 0, & b + 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 < 0 \\ 1, & b + 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 \geq 0 \end{cases}$$

- 편향(bias)는 학습 데이터(입력신호)와 가중치(Weight)의 계산에 의한 값이 넘어야 할 값
- 편향보다 높으면 1 혹은 0으로 분류되는 기준이 높아지기 때문에 분류할때 엄격하게 분류하게됨
- 편향값이 높을 수록 학습 모델은 간단해지는 경향을 보이고 Underfitting(과소적합)이 될 수 있음
- 편향값이 낮을 수록 학습 모델은 복잡해지는 경향을 보이고 Overfitting(과적합)이 될 수 있음

❖ W 역할 : 입력 신호가 결과 출력에 주는 영향을 조절

❖ b 역할 : 얼마나 쉽게 활성화(결과를 1로 출력)되는지를 조절

❖ 다층 퍼셉트론(Multi Layer Perceptron, MLP - 다수의 퍼셉트론 사용하는 신경망)을 활용하여 어려운 문제 혹은 비선형적 문제를 해결 할 수 있음

Activation Function

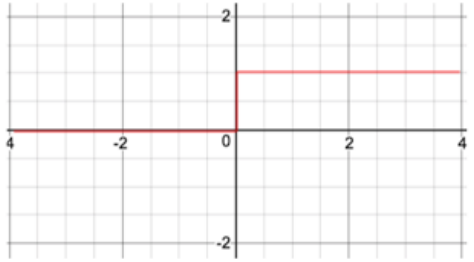
❖ Activation Function(활성화 함수)

- threshold(임계값)을 이용하여 출력값을 결정하는 함수
- 출력값에 따라서 다음 단계(뉴런) 의 입력값의 상태를 결정하게 됨

❖ 종류

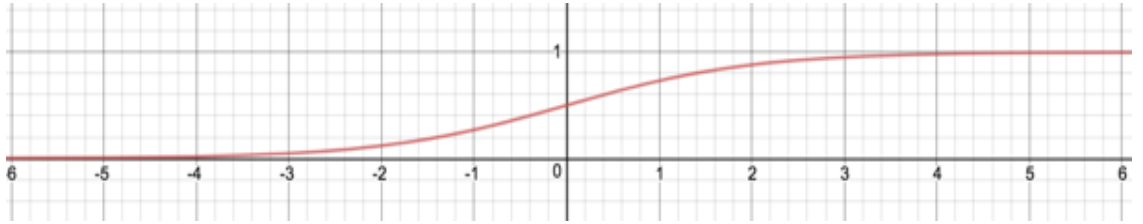
- Step Function
  - ✓ 가장 기본이 되는 활성화 함수로 계단 형태를 가지고 있음
  - ✓ 0을 기준으로 0 혹은 1을 출력

$$Output = \begin{cases} 0, & x \le 0 \\ 1, & x > 0 \end{cases}$$



- Sigmoid Function
  - ✓ 0과 1 사이의 연속적인 출력값을 가질 수 있도록 하는 비선형 함수
  - ✓ 신경망 초기에는 많이 사용되었지만 Gradient Vanishing 현상이 발생하여 최적화가 안되는 현상이 발생하여 최근에는 많이 사용하지 않음

$$P = \frac{1}{1 + e^{-X}}$$

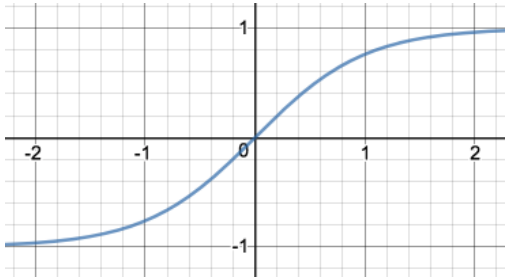


Activation Function

❖ 종류

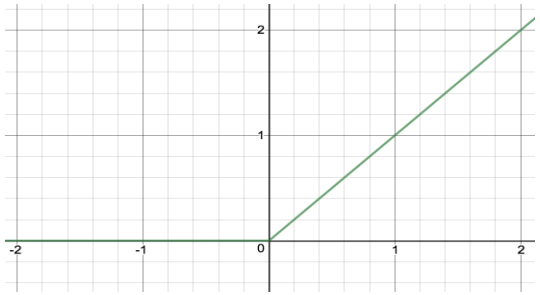
- Hyperbolic Tangent Function, tanh
  - ✓ 함수의 중심값을 0으로 옮겨 출력값의 범위는 -1~1 사이의 연속적인 출력값가지는 비선형 함수
  - ✓ Sigmoid Function 보다 최적화가 빠르지만 Gradient Vanishing 현상이 발생함

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



- ReLU(Rectified Linear Unit) Function
  - ✓ 최근 많이 사용되는 활성화 함수로 x가 0보다 크면 기울기가 1인 직선을 가짐
  - ✓ Sigmoid, tanh Function보다 학습이 빠르며 구현이 쉬움
  - ✓ x 가 0보다 작은 값들에 대해서는 미분시 기울기가 0이기 때문에 뉴런이 활성화가 되지 않음

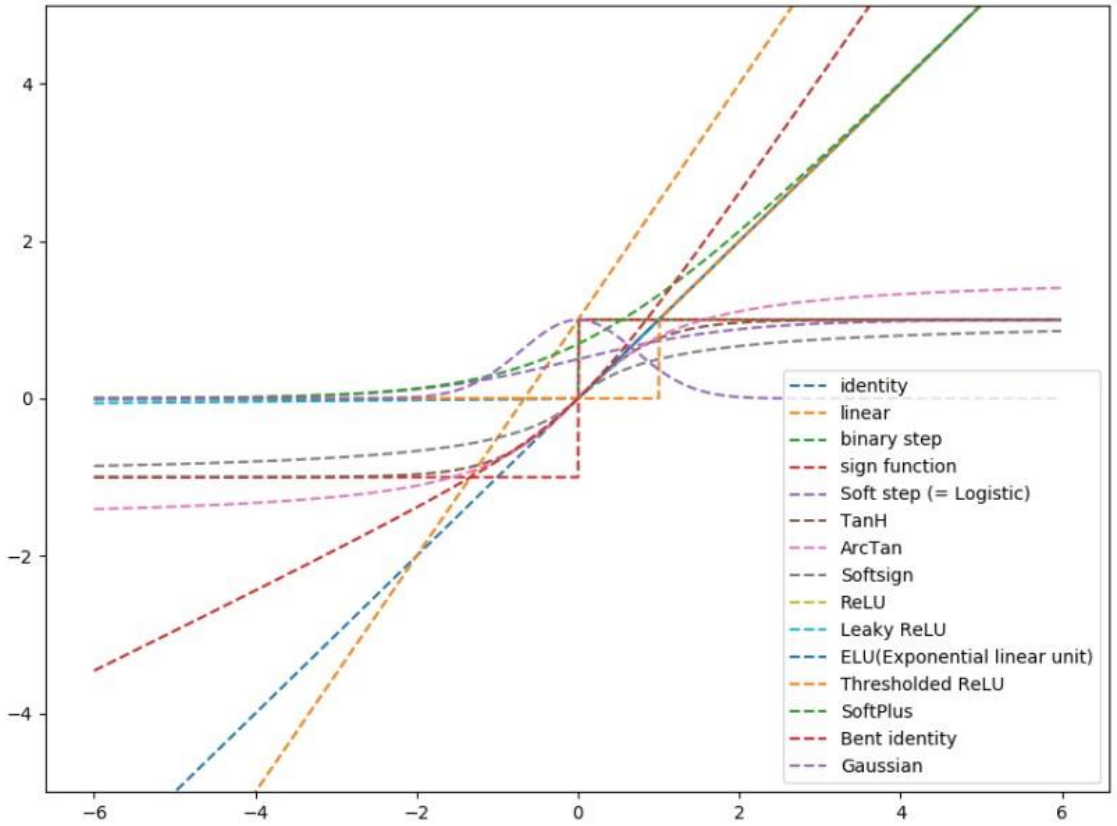
$$f(x) = \max(0, x)$$



Activation Function

❖ 종류

- 이외의 Activation Function 그래프



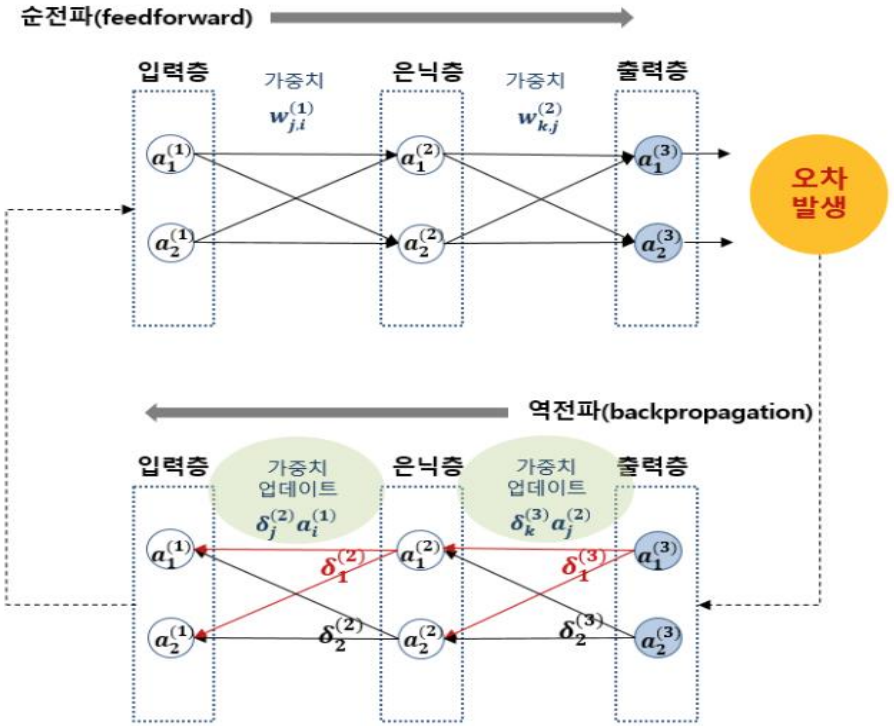
출처 : https://mblogthumb-phinf.pstatic.net/MjAxNzA2MDNfMTQ2/MDAxNDk2NDU0NjE5OTY1.KDNgrWWc2BlWJzitH-7kd6HkA_7tR-uBhSA1SBNhBdgg.-G6q8LTex-T7CvoRCSkuCfULFEFoGSjHa6TxkA7Qm58g.JPEG.wideeyed/%25EC%25A0%2584%25EC%25B2%25B4%25EA%25B7%25B8%25EB%259E%2598%25ED%2594%2584.jpg?type=w800

KOREATECH 한국기술교육대학교

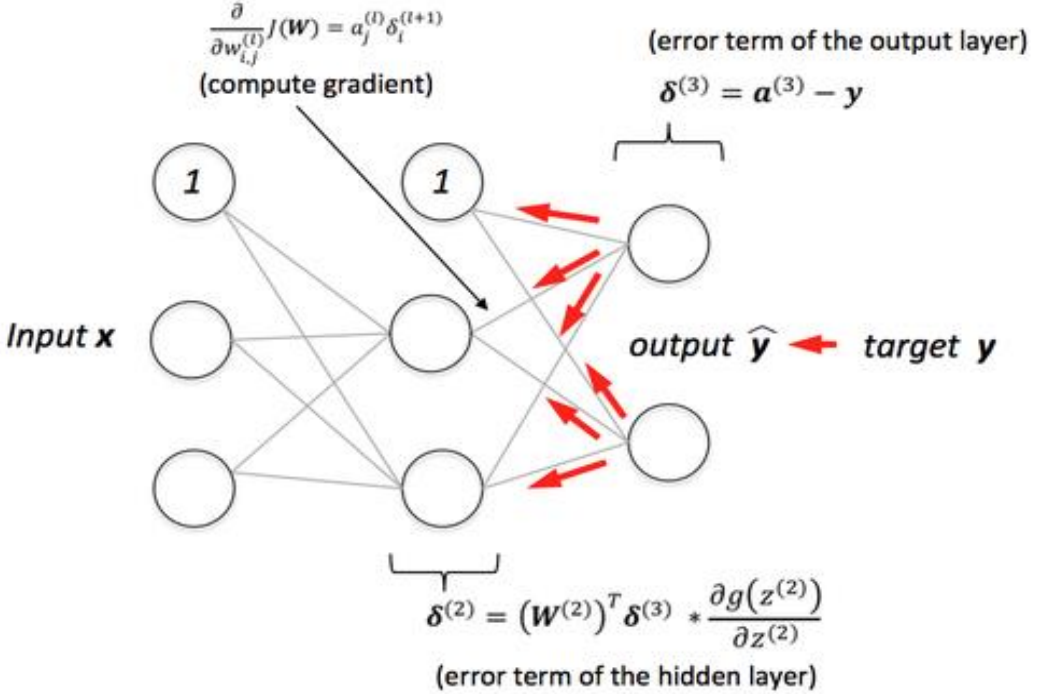
Backpropagation

❖ Backpropagation 알고리즘을 이용한 모델 학습 과정

- 순전파 -> 역전파 -> 가중치업데이트 -> 순전파 -> 역전파 -> 가중치 업데이트 …. 과정을 반복하여 예측값과 결과값의 오차가 최소가 되는 W, b를 찾음



출처 : https://m.blog.naver.com/samsjang/221033626685?view=img_75



출처 : https://sebastianraschka.com/faq/docs/visual-backpropagation.html

- 역전파 데모 참고 자료 : https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/?hl=ko

Backpropagation

❖ 일반적인 비용함수 최적화

- Gradient descent 알고리즘을 이용하여 비용함수 미분을 통하여 오차가 최소가 되는 W(Weight), b(bias) 를 최적화함
- 순전파(Forward propagation) 과정(Input->Hidden->Output Layer)을 통하여 미분값을 업데이트

❖ Backpropagation 알고리즘 학습 과정

- 신경망의 W(가중치)를 적당한 값으로 초기화
- Input Layer에 학습데이터를 입력하여 순전파(Foward propagation) 과정을 통하여 비용함수의 미분값 연산 수행
- Output Layer의 출력한 예측값과 실제값의 오차를 계산
- 계산된 오차를 신경망의 각각의 뉴런들에 오차를 역전파(Backpropagate)하여 에러값을 이전 Layer로 전달
- 전달된 오차는 뉴런들의 W로 사용되며, 오차가 최소가 되는 W, b 를 최적화 함

❑ Forward propagation: Input Layer로 입력된 학습데이터로부터 예측값을 계산하고, 각 Ouput Layer 뉴런에서의 오차를 계산.
Input -> Hidden -> Output 으로 정보가 흘러가기 때문에 'Forward' propagation이라 함

❑ Backpropagation: Output Layer 뉴런에서 계산된 오차를 각 edge들의 weight를 사용해 바로 이전 Layer의 뉴런들이 얼마나 오차에 영향을 미쳤는지 계산.
Output -> Hidden Layer 으로 정보가 흘러가기 때문에 'Back' propagation이라 함

MNIST Neural Network 개요

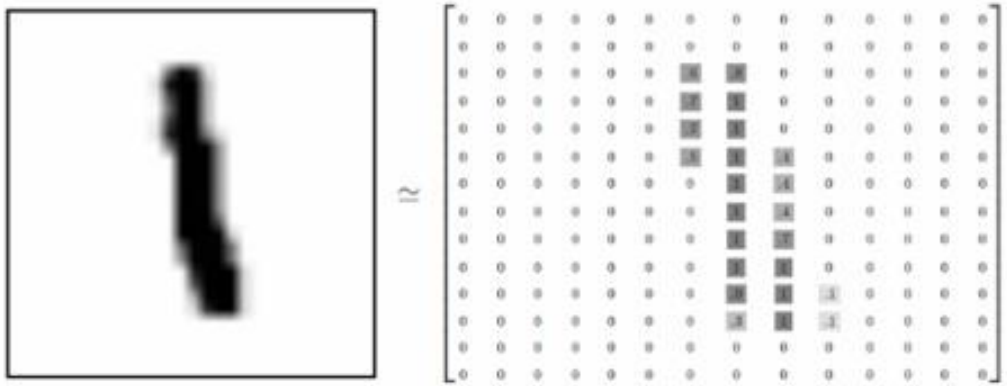
❖ MNIST

- MNIST(Modified National Institute of Standards and Technology database) 데이터세트
  - ✓ 손으로 쓴 숫자들로 이루어진 대형 데이터베이스
  - ✓ 다양한 화상 처리 시스템을 트레이닝 하기 위해 일반적으로 사용
  - ✓ 55,000개의 훈련데이터와 10,000개의 테스트 데이터 5,000개의 검증 데이터로 구성
  - ✓ 데이터 샘플 이미지

MNIST Neural Network 개요

❖ MNIST

- 손글씨 이미지를 픽셀 데이터로 변환하여 학습에 사용할 수 있도록 함



- 하나의 이미지는 28 x 28 픽셀로 구성되어 있으며 픽셀 데이터를 784(28*28)의 벡터로 변환하여 학습에 사용

- Scikit Learn의 MLP를 이용하여 인식기 구성해보자.

학습정리

- 구글티쳐블머신
  - no coding
- scikit learn (sklearn)
  - 기계학습패키지
  - MNIST 활용