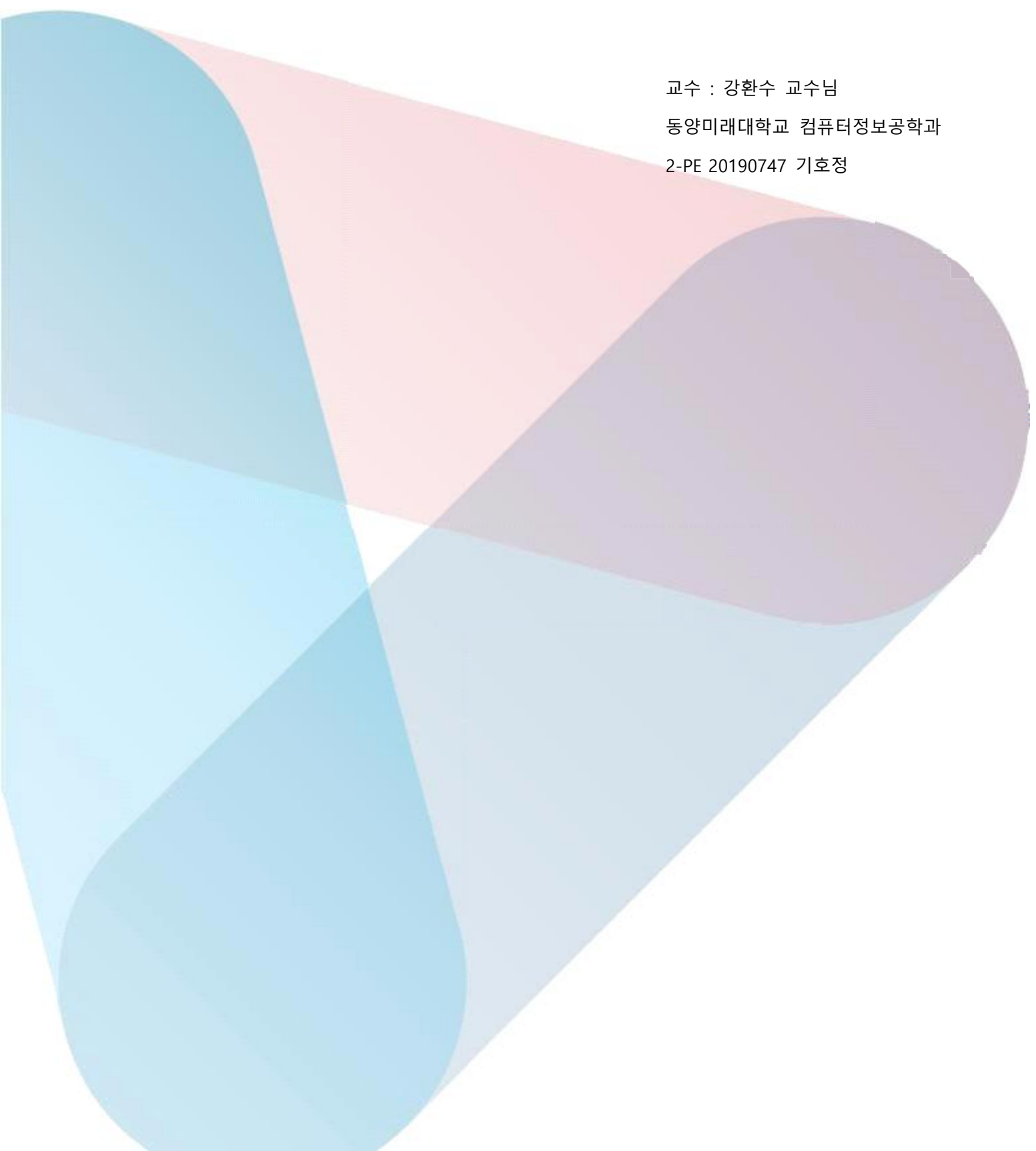


인공지능

교수 : 강환수 교수님

동양미래대학교 컴퓨터정보공학과

2-PE 20190747 기호정



강의계획서

Part 1

- 1.1 머신러닝과 딥러닝
- 1.2 딥러닝 라이브러리

Part 2

- 2.1 텐서플로 코딩
- 2.2 텐서플로 난수

Part 3

- 3.1 손글씨 글자 이미지 MNIST 분류하기
- 3.2 MNIST 손글씨 데이터 로드
- 3.3 MNIST 데이터 딥러닝 모델 적용 예측

Part 4

- 4.1 퍼셉트론
- 4.2 게이트

Part 5

5.1. 회귀와 분류

5.2. 선형 회귀(linear regresson)

5.3. 최적화 과정

Part 6

6.1. 케라스 모델 미사용 텐서플로 프로그래밍

6.2. 예측하기

6.3. seaborn

Part 7

7.1. 이항 분류 및 다항 분류



PART 1

1.1 머신러닝과 딥러닝

1.1.1 AI의 시작

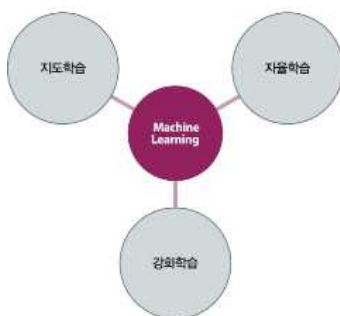
1950년, 앨런 튜링은 논문 <Computing machinery and intelligence>에서 생각하는 기계에 대한 내용을 발표했다. 인공지능 실험인 '튜링 테스트'에서 텍스트로 주고받는 대화에서 기계가 사람인지 기계인지 구별할 수 없을 정도로 대화를 잘 이끌어 간다면, 이것은 기계가 '생각'하고 있다고 말할 충분한 근거가 된다는 논문을 뒷받침했다.

1.1.2 AI에 대한 관심

AI(인공지능)는 이제 개발자를 대상으로 하는 전문서와 웹 사이트뿐만 아니라, 일반인을 대상으로 하는 뉴스와 신문에서도 자주 언급되는 주제이다. AI는 이미 두 번이나 많은 사람의 주목을 받았으며, 현재 세 번째 붐이 일고 있다. 그리고 그 중심에는 '머신러닝(기계학습)과 딥러닝(심층학습)'이 자리잡고 있다.

이를 활용하면 인간으로부터 주어진 데이터를 기반으로 기계인 컴퓨터가 스스로 학습하고, 지식을 습득할 수 있다. 과거의 인공지능과 다르게 현재의 인공지능은 인간이 생각하지 못한 패턴도 학습할 수 있으므로, 범용성이 높고 다양한 분야에 활용할 수 있다. 그중에서도 특히 '딥러닝'이라고 불리는 영역이 주목을 모으고 있으며, 딥러닝은 이미지 인식, 음성 인식, 기계 번역 등의 분야에서 큰 성과를 내고 있다.

1.1.3 기계학습이란?



기계학습이란 '기계에게 데이터를 학습시키고, 데이터에 포함된 규칙성 또는 패턴을 발견하도록 하는 처리'를 나타낸다. 데이터가 10개 정도밖에 없으면 인간이 규칙과 패턴을 어느 정도 발견할 수 있지만, 현실의 데이터는 너무 규모가 크고 구조가 복잡해서 인간이 처리할 수 있는 한계를 넘는다. 그래서 이러한 처리를 기계에게 맡기는 것이다.

기계학습은 다음과 같이 데이터를 처리한다. 기계는 일단 입력으로 '학습 데이터'를 받는다. 그리고 여기에서 '특징량'을 추출한다. 특징량이란 각각의 데이터가 가지고 있는 어떠한 특징을 수치화해서 나타낸 것이다. 이때 특징량 정의는 사람이 해야 한다.

지도학습(supervised learning)

기계학습은 크게 '지도학습'과 '비지도학습'으로 구분할 수 있다. 지도학습은 주로 '예측'에 사용되고, 비지도학습은 주로 '지식 발견'에 사용된다.

지도 학습에서는 학습 데이터로 정답 데이터(목적 변수)를 포함하고 있는 데이터를 입력한다. 그리고 정답 데이터를 제외한 입력 데이터(설명 변수)를 기반으로 얻은 출력 결과가 최 대한 정답 데이터에 가깝게 특징량을 추출해서 모델을 구축한다.

예를 들어 생산 기계의 가동 로그를 기반으로 다음에 어떤 기계가 고장 날지 예측해본다고 하자. 높은 정밀도로 예측할 수 있다면 빠르게 유지 보수해서 생산 효율을 높일 수 있을 것이다.

학습방법 =>

k-최근접 이웃(k-Nearest Neighbors)

선형 회귀(Linear Regression)

로지스틱 회귀(Logistic Regression)

서포트 벡터 머신(Support Vector Machines(SVM))

결정 트리(Decision Tree)와 랜덤 포레스트(Random Forests)

비지도 학습(unsupervised learning)

비지도 학습에서는 정답 데이터를 포함하지 않는(목적 변수가 없는) 입력 데이터를 사용한다. 따라서 입력 데이터에서 특징량을 추출해서 모델을 추출한다.

예를 들어 고객의 속성 정보를 기반으로 취향이 비슷한 고객을 묶는 경우 등을 생각할 수 있다. 고객을 명확하게 그룹으로 나눈다면 각 그룹에 속한 고객에게 적합한 상품을 추천해서 매출 증가를 기대할 수 있다.

학습방법 =>

군집화(Clustering) : k-평균, 계층 군집 분석, 기댓값 최대화

시각화(Visualization)와 차원 축소(Dimensionality reduction) : 주성분 분석, 커널,
지역적 선형 임베딩

연관 규칙 학습(Association rule learning) : 어프라이어리, 이클렛

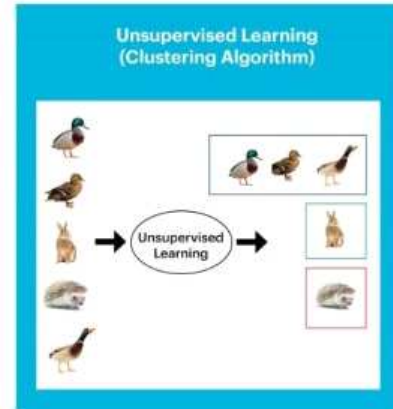
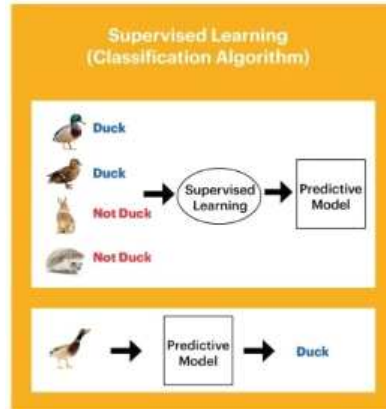
강화 학습(reinforcement learning)

잘한 행동에 대해 보상을 주고 잘못된 행동에 대해서는 벌을 주는 경험을 통해 지식을 학습하는 방법이다.

학습방법 =>

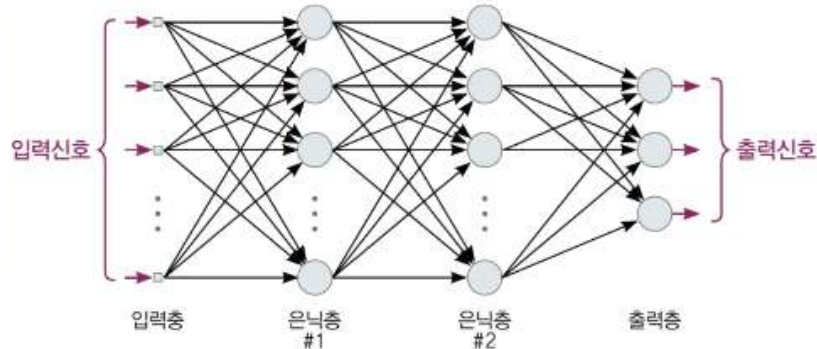
딥마닝의 알파고

자동 게임분야



1.1.4 딥러닝이란

딥러닝은 중간 레이어가 여러 개로 구성된 신경망을 사용해 구축하므로 '심층 신경망'(Deep Neural Network : DNN)이라고 부르기도 한다. 딥러닝은 2006년 토론토 대학의 제프리 힌튼 박사가 고안했다. 딥러닝이 세상에 널리 알려진 것은 2012년에 구글이 '대규모 비지도 학습 이미지 데이터'를 기반으로 기계가 고양이를 자동으로 인식할 수 있다'라고 발표하면서부터이다. 추가로 2015년에 구글의 알파고가 이세돌 선수와의 바둑에서 승리하면서 사회에 큰 충격을 주어 딥러닝이 사회에 전반적으로 알려졌다.



딥러닝은 신경망에서 발전된 것으로, '기계학습방법의 하나'이다. 하지만 '기존의 기계학습과는 완전히 다른 학습방법'이라고 보는 시각도 있다. 이는 특징량 추출에 사람이 개입할 필요가 없기 때문이다. 기존의 기계학습은 특징량을 사람이 직접 정의해야 하지만, 딥러닝은 **기계 학습하면서 특징량을 자동으로 추출한다**.

딥러닝은 특히 이미지, 음성, 언어 등의 '비구조화 데이터'에서 특징량을 추출하고, 정밀도가 높은 모델을 구축하는 데 뛰어나다. 이러한 데이터는 설명 변수의 수가 많아서, 사람이 직접 특징량을 추출하기는 굉장히 어렵다. 따라서 기존의 기계학습방법으로 정밀한 모델을 구축하려면 이미지, 신호 처리, 자연어 처리와 관련된 전문적인 지식이 필요하다. 하지만 딥러닝을 사용하면 기계가 자동으로 특징량을 추출해주므로 전문 지식이 많지 않아도 어느 정도 정밀한 모델을 구축할 수 있다.

1.2 딥러닝 라이브러리

1.2.1 텐서플로

텐서플로(TensorFlow)는 딥러닝을 포함해 다양한 머신러닝을 구현할 때 사용되는 라이브러리이며, 2015년 11월에 구글이 공개했다. 아파치 2.0 라이선스를 따르므로 상업적인 용도와 비상업적인 용도 관계없이 자유롭게 사용할 수 있다.

텐서플로의 전신은 디스트빌리프라고 불리는 플랫폼이었다. 이는 2012년에 구글이 발표한 고양이 이미지 인식에 사용됐다. 여기에 빠른 속도 향상을 위한 스케일 아웃 등을 할 수 있게 개선된 것이 바로 텐서플로이다.

구글은 텐서플로를 라이브러리로 제공할 뿐만 아니라, 구글에서 직접 서비스를 만들 때도 사용하고 있다. 예를 들어 구글은 이미지 인식, 음성 인식, 번역 등에 모두 텐서플로를 활용하고 있다. 이처럼 텐서플로는 실제로 애플리케이션 구현에도 사용되고 있으므로 개인적인 용도의 사용부터 비즈니스 용도의 사용까지 모두 커버할 수 있는 굉장히 강력한 라이브러리라고 할 수 있다.

1.2.2 텐서

Tensor(텐서) : 모든 데이터

- 0-D 텐서 : 스칼라(차원이 없는 텐서)
- 1-D 텐서 : 벡터(1차원 텐서)
- 2-D 텐서 : 행렬 등(2차원 텐서) -> 하나의 채널에 2차원 행렬로 표현(회색조 이미지)
- n-D 텐서 : 텐서(n차원 행렬) -> RGB 각 3개의 채널마다 2차원 행렬로 표현(RGB 이미지)



PART 2

2.1 텐서플로 코딩

2.1.1 행렬 곱(내적)

- Numpy
 - `np.dot(a, b)`
 - `a.dot(b)`
- Tf
 - `tf.matmul`

$$\begin{matrix} \text{A} & \text{B} & \text{A} * \text{B} \end{matrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 1 \cdot 6 + 2 \cdot 5 + 3 \cdot 4 & 1 \cdot 3 + 2 \cdot 2 + 3 \cdot 1 \\ 4 \cdot 6 + 5 \cdot 5 + 6 \cdot 4 & 4 \cdot 3 + 5 \cdot 2 + 6 \cdot 1 \end{pmatrix}$$

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$C_{ij} = \sum_k A_{ik} B_{kj} = A_{ik} B_{kj}$$

```
[9] # Matrix multiplications 1
matrix1 = tf.constant([[1., 2.], [3., 4.]])
matrix2 = tf.constant([[2., 0.], [1., 2.]])
```

```
gop = tf.matmul(matrix1, matrix2)
print(gop.numpy())
```

```
↳ [[ 4.  4.]
    [10.  8.]]
```

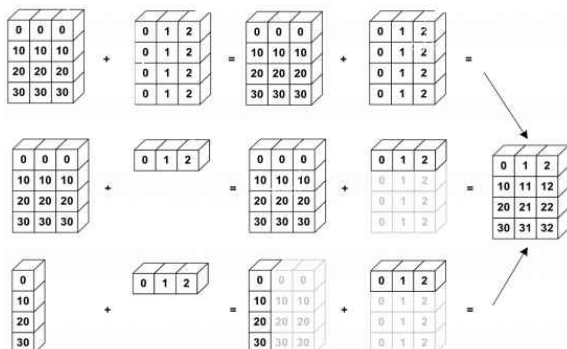
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 10 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 7 & 10 \end{bmatrix}$$

```
[10] # Matrix multiplications 2
gop = tf.matmul(matrix2, matrix1)
print(gop.numpy())
```

```
↳ [[ 2.  4.]
    [ 7. 10.]]
```

2.1.2 브로드캐스팅



=> 가지고 있는 값을 이용하여 Shape을 맞추

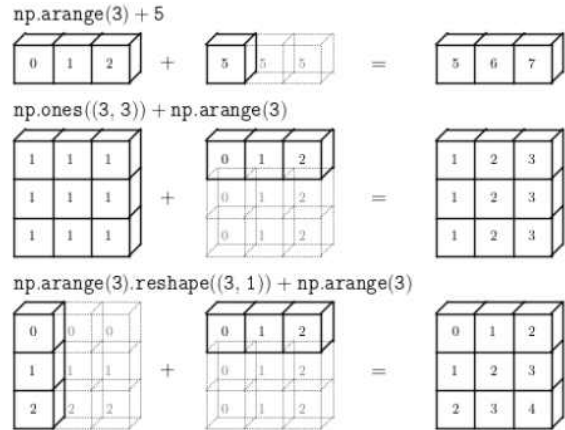
```
[45] x = tf.constant((np.arange(3)))
y = tf.constant([5], dtype=tf.int64)
print((x+y).numpy())

x = tf.constant((np.ones((3, 3))))
y = tf.constant(np.arange(3), dtype=tf.double)
print((x+y).numpy())

x = tf.constant(np.arange(3).reshape(3, 1))
y = tf.constant(np.arange(3))
print((x+y).numpy())
```

↳

```
[5 6 7]
[[1. 2. 3.]
 [1. 2. 3.]
 [1. 2. 3.]]
[[0 1 2]
 [1 2 3]
 [2 3 4]]
```



2.1.3 텐서플로 연산

- tf.add() => 더하기
- tf.multiply() => 곱하기
- tf.pow() => 제곱
- tf.reduce_mean() => 평균
- tf.reduce_sum() => 합

```
[46] a = 2
b = 3
c = tf.add(a, b)
print(c.numpy())
```

↳ 5

```
[47] x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
pow_op = tf.pow(add_op, mul_op)

print(pow_op.numpy())
```

↳ 15625

```
[48] a = tf.constant(2.)
b = tf.constant(3.)
c = tf.constant(5.)

# Some more operations.
mean = tf.reduce_mean([a, b, c])
sum = tf.reduce_sum([a, b, c])

print("mean = ", mean.numpy())
print("sum = ", sum.numpy())
```

↳ mean = 3.3333333
sum = 10.0

2.2 텐서플로 난수

2.2.1 텐서플로의 난수 활용

- `tf.random.uniform([1], 0, 1)` => 균등분포

* ([배열], 시작, 끝)

```
[11] 1 rand = tf.random.uniform([1000], 0, 10)
      2 print(rand[:10])
```

```
tf.Tensor(
[5.1413307 1.548909 8.911686 9.880335 5.5388713 5.6710424 6.80269
 1.9444573 7.549943 6.573516 ], shape=(10,), dtype=float32)
```

- `tf.random.normal([4], 0, 1)` => 정규분포

* 크기, 평균, 표준편차

```
[54] 1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
      2 rand = tf.random.normal([2, 4], 0, 2)
      3 print(rand)
```

```
tf.Tensor(
[[-2.145662  0.64699423  2.0760484 -1.4640687 ]
 [ 1.3588632 -0.9740333  1.4347676 -1.3747462 ]], shape=(2, 4), dtype=float32)
```

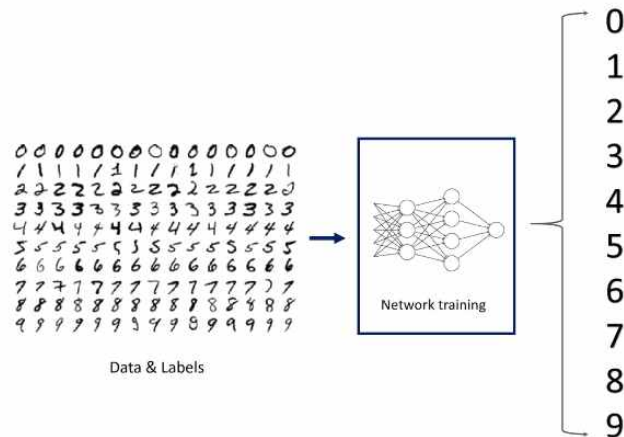


PART 3

3.1 손글씨 글자 이미지 MNIST 분류하기

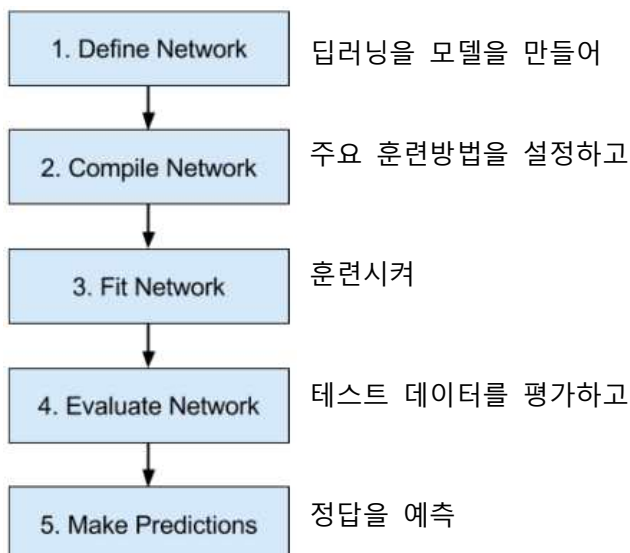
3.1.1 MNIST 데이터셋

MNIST는 손글씨 숫자 0~9가 적혀 있는 흑백 이미지 데이터 세트이다. 일반적으로 이미지 분류와 관련된 내용을 공부할 때 널리 사용된다. 각 이미지의 크기는 28 X 28 픽셀이다.



제공되는 이미지는 JPEG 등의 일반적인 이미지 형식이 아니라, 픽셀값을 행렬로 표현한 형식으로 제공된다. 데이터에는 '28 X 28 = 784개의 픽셀값'과 함께 '0~9 중에 어떤 숫자의 이미지인지 정답 데이터'가 포함되어 있다. '필기 숫자 이미지'와 정답인 '레이블'의 쌍으로 구성된 MNIST는 회색조 이미지이다.

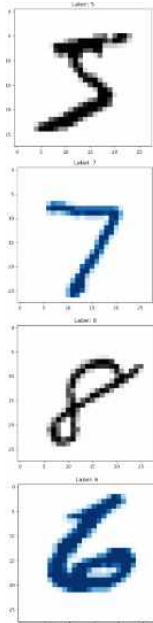
3.1.2 케라스 딥러닝 구현과정



3.2 MNIST 손글씨 데이터 로드

3.2.1 행렬 곱(내적)

- 훈련 데이터 손글씨와 정답(x_train, y_train) => 6만개
- 테스트 데이터 손글씨와 정답(x_test, y_test) => 1만개



MNIST 데이터(훈련, 테스트)의 내부 첫 내용을 그려보자.
import matplotlib.pyplot as plt

```
tmp = "Label: " + str(y_train[0])  
plt.title(tmp)  
plt.imshow(x_train[0], cmap="Greys")  
plt.show()
```

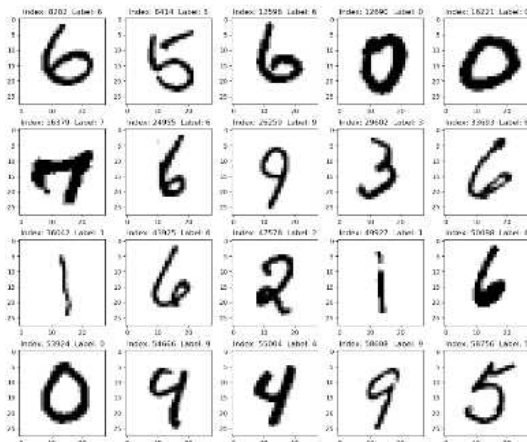
```
tmp = "Label: " + str(y_test[0])  
plt.title(tmp)  
plt.imshow(x_test[0], cmap='Blues')  
plt.show()
```

MNIST 데이터(훈련, 테스트)의 내부 마지막 내용을 그려보자.

```
idx = len(x_train) - 1  
tmp = "Label: " + str(y_train[idx])  
plt.title(tmp)  
plt.imshow(x_train[idx], cmap="Greys")  
plt.show()
```

```
idx = len(x_test) - 1  
tmp = "Label: " + str(y_test[idx])  
plt.title(tmp)  
plt.imshow(x_test[idx], cmap='Blues')  
plt.show()
```

3.2.2 랜덤하게 20개



#랜덤하게 20개의 훈련용 자료를 그려 보자.

```
from random import sample
```

```
nrows, ncols = 4, 5 #출력 가로 세로 수
```

```
# 출력할 첨자 선정
```

```
idx = sorted(sample(range(len(x_train)), nrows * ncols))
```

```
#print(idx)
```

```
count = 0
```

```
plt.figure(figsize=(12, 10))
```

```
for n in idx:
```

```
    count += 1
```

```
    plt.subplot(nrows, ncols, count)
```

```
    tmp = "Index: " + str(n) + " Label: " +
```

```
    str(y_train[n])
```

```
    plt.title(tmp)
```

```
    plt.imshow(x_train[n], cmap='Greys')
```

```
plt.tight_layout()
```

```
plt.show()
```


3.3 MNIST 데이터 딥러닝 모델 적용 예측

3.3.1 MNIST 데이터 셋을 위한 딥러닝

- 0 ~ 9까지의 분류(classification)
 - 클래스 10개

3.3.2 딥러닝 과정

- 모델 구성(개발)
 - 블랙 박스
- 모델 훈련 : train
 - 경사하강법(내리막 경사 따라 가기)
 - 손실 함수(Loss Function)
 - 모니터링 지표(metrics)
- 예측 : inference, prediction

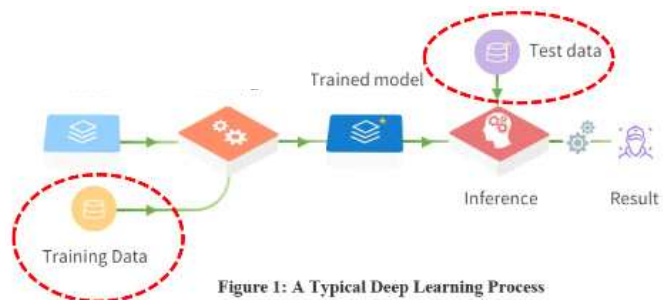


Figure 1: A Typical Deep Learning Process

3.3.3 주요 용어

- 데이터셋
 - Train data set, Test data set
 - x(입력), y(정답)
- 모델
 - Dense() : 완전연결층
 - Flatten() : 평탄화
- 학습 방법의 여러 요소들
 - optimizer : 최적화 방법(경사하강법)
 - loss function : 손실 함수
- 딥러닝 훈련
 - Epochs : 총 훈련 횟수

3.3.4 MNIST 데이터셋

- 전처리 : 0~1 사이값으로 변경

```
import tensorflow as tf

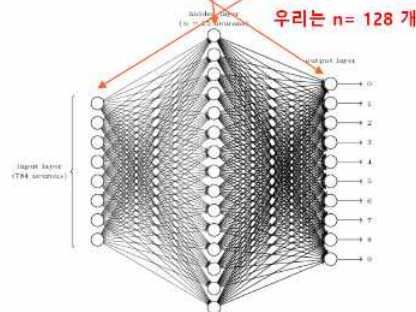
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 샘플 값을 정수 (0~255)에서 부동소수 (0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0
```

- 모델 구성 : 신경망 구성

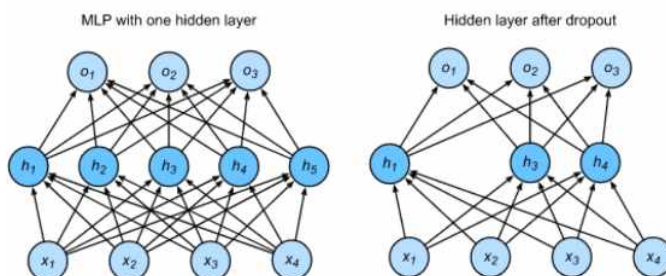
```
# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



- 평탄화 : 1차원 형태로 정렬

```
# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
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')
])
```

- 드롭아웃 : 훈련 중에 중간을 끊는 작업



3.3.5 MNIST 딥러닝 구현 전 소스

```
import tensorflow as tf

# mnist 모듈 준비
mnist = tf.keras.datasets.mnist

# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0

# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

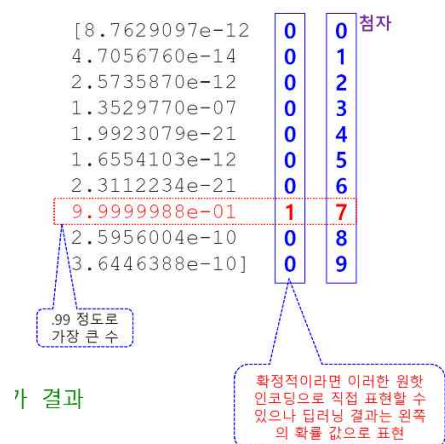
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 요약 표시
model.summary()

# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)

# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

3.3.6 원핫인코딩




- 10개의 수를 더하면 1
- 0 ~ 9까지의 확률값

3.3.7 배열에서 가장 큰 값의 첨자 구하기

- 메소드 `np.argmax()`

```
[46] 1 import numpy as np
      2
      3 #####
      4 # 원한 인코딩과 argmax 학습
      5 print(np.argmax([5, 4, 10, 1, 2]))
      6 print(np.argmax([3, 1, 4, 9, 6, 7, 2]))
      7 print(np.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```

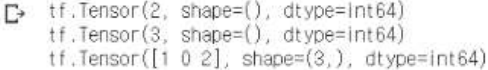


`axis = 1`은 각 배열마다 하나씩 뽑겠다는 의미로 위의 예제에서는 0.8, 0.7, 0.7을 의미하는데 `argmax`는 첨자를 뜻하므로 각각의 첨자가 출력된다.

- 메소드 `tf.argmax()`

```
[11] import numpy as np

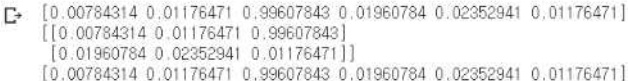
#####
# 원한 인코딩과 argmax 학습
print(tf.argmax([5, 4, 10, 1, 2]))
print(tf.argmax([3, 1, 4, 9, 6, 7, 2]))
print(tf.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```



3.3.8 MNIST 손글씨 예측과 결과 확인

- 활성화 함수 `softmax()`
 - `tf.keras.layers.Dense(128, activation='relu')`
 - `softmax`의 결과값은 확률값임
- 평탄화 메소드 `flatten`
 - `ary.flatten()`

```
[50] 1 #####
      2 # 간단한 자료 처리
      3 import numpy as np
      4
      5 x = np.array([2, 3, 254, 5, 6, 3])
      6 x = x / 255.0
      7 print(x)
      8
      9 x = x.reshape(2, 3)
     10 print(x)
     11
     12 x = x.flatten()
     13 print(x)
```



- 드롭아웃
 - 층에서 결과 값을 일정 비율로 제거하는 방법
 - 과적합 문제 해결
 - 일반적으로 훈련단계에서 적용하고, 테스트 단계에서는 어떤 유닛도 드롭하지 않음
 - 지정한 비율로 0 지정

3.3.9 MNIST 손글씨 임의 20개 정답과 오답

- 임의의 20개 예측값과 정답
 - pred_result : 모델의 예측 결과, 확률값
 - pred_labels : 모델의 예측 결과, 정수
 - samples : 출력할 20개의 첨자 리스트

```
#####
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.
samples = sorted(sample(range(len(x_test)), rows * ncols)) # 출력할 첨자 선정

# 임의의 20개 그리기
count = 0
nrows, ncols = 5, 4
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()
#####
```

- pred_labels[n] == y_test[n]
 - => 예측이 맞는 경우

- 임의의 20개 예측값과 오답

```

from random import sample
import numpy as np

#####
# 예측 틀린 것 점자들 저장할 리스트
mispred = []
# 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)

for n in range(0, len(y_test)):
    if pred_labels[n] != y_test[n]:
        mispred.append(n)
print('정답이 틀린 수', len(mispred))

# 랜덤하게 틀린 것 20개의 첨자 리스트 생성
samples = sample(mispred, 20)
print(samples)

# 틀린 것 20개 그리기
count = 0
nrows, ncols = 5, 4
plt.figure(figsize=(12,10))
for n in samples:
    count += 1
    plt.subplot(nrows, ncols, count)
    plt.imshow(x_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
    tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])
    plt.title(tmp)

plt.tight_layout()
plt.show()
#####

```

- 틀린 첨자 저장 : mispred
- pred_labels[n] != y_test[n]
- => 예측이 틀린 조건

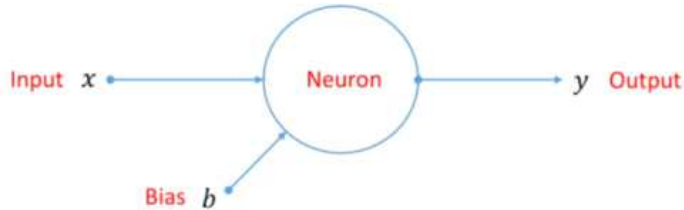


PART 4

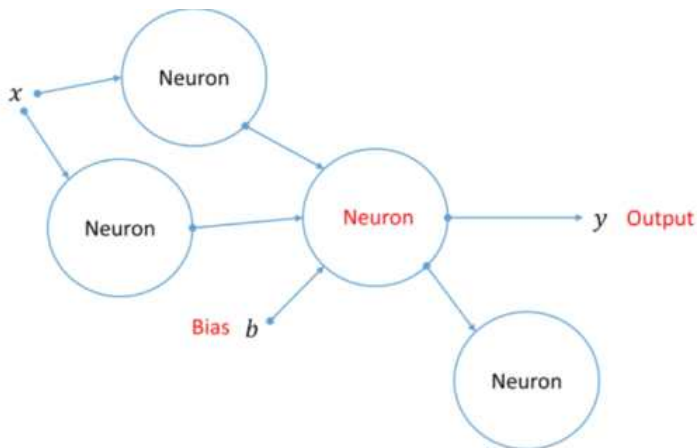
4.1 퍼셉트론

4.1.1 인공 신경망 퍼셉트론

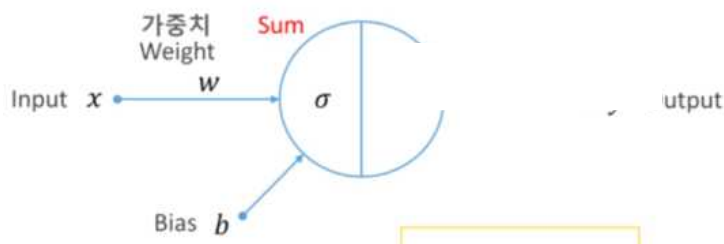
- 뉴런 : 입력, 편향



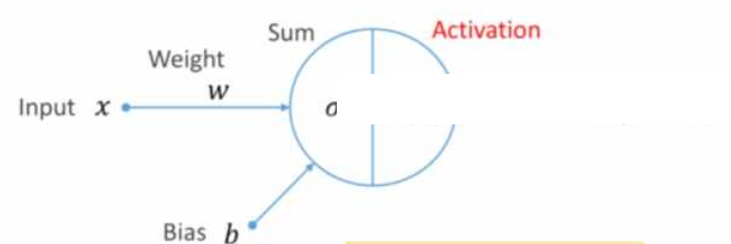
- 신경망 : 뉴런의 연결



- 편향 : 조정하여 출력을 맞춤
- 뉴런식 : $\sigma = w \cdot x + b$

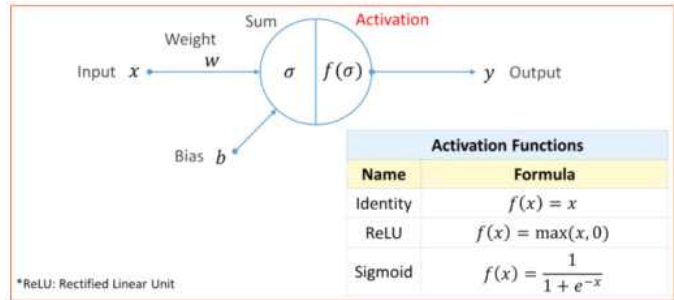


- 활성화 함수 : 뉴런의 출력값을 정하는 함수 $f(\sigma) = f(w \cdot x + b)$



4.1.2 활성화 함수 ReLU, sigmoid

- ReLU
 - 정류된 선형 함수로 $y=x$ 를 의미
 - $\max(x, 0)$ => 양수만 사용 가능
- Sigmoid
 - S자 형태의 곡선이라는 의미



- 활성화 함수와 편향
 - 결과가 임계 값 이상이면 활성화
 - 결과가 임계 값 미만이면 비활성화

$$f_w(x) = \left\{ \begin{array}{l} \sum w_i x_i \geq \theta \rightarrow \text{neuron fires} \\ \sum w_i x_i < \theta \rightarrow \text{neuron doesn't fire} \end{array} \right\}$$

- 행렬의 다른 표현
 - 입력을 오른쪽 행렬에 배치
 - 가중치는 왼쪽 행렬에 배치
 - 곱의 순서도 변환

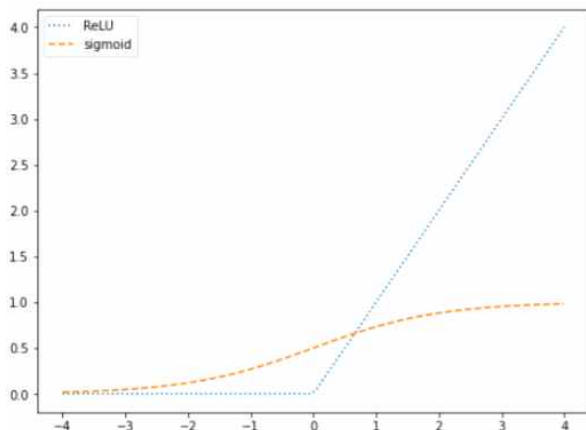
```
import numpy as np
import matplotlib.pyplot as plt

# ReLU(Rectified Linear Unit)
# (정류된 선형 유닛) 함수
def relu_func(x):
    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=(8, 6))
x = np.linspace(-4, 4, 100)
y = np.linspace(-0.2, 2, 100)

plt.plot(x, relu_func(x), linestyle=':', label="ReLU")
plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")
plt.legend(loc='upper left')
```



4.2 게이트

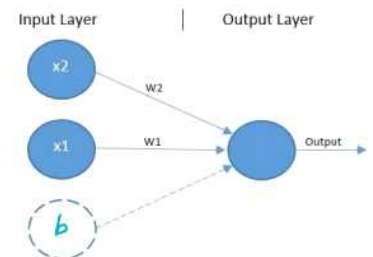
4.2.1 논리 게이트 AND OR XOR 신경망 구현

- AND 게이트 구현
 - 가중치 2개 + 편향 1개

```
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=(2,)),
])

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



- OR 게이트 구현

```
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,)),
])

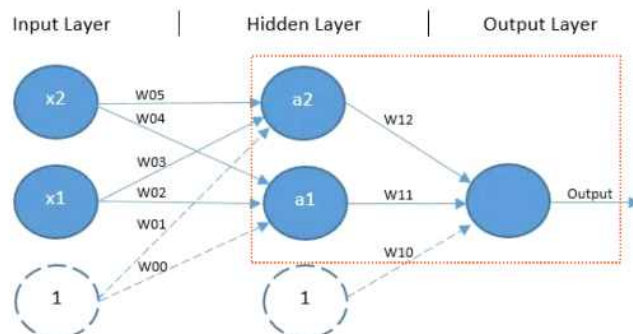
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()
```

- XOR 게이트 구현

```
import tensorflow as tf
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()
```



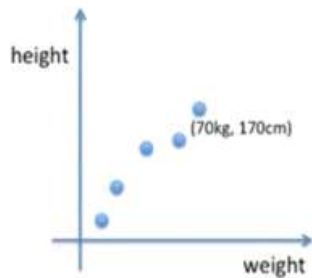


PART 5

5.1 회귀와 분류

5.1.1 회귀(regression)와 분류(classification)

- 회귀 모델
 - 연속적인 값을 예측
 - ex) 사용자가 이 광고를 클릭할 확률은?



- 분류 모델
 - 불연속적인 값을 예측
 - ex) 이 이미지는 강아지인가?



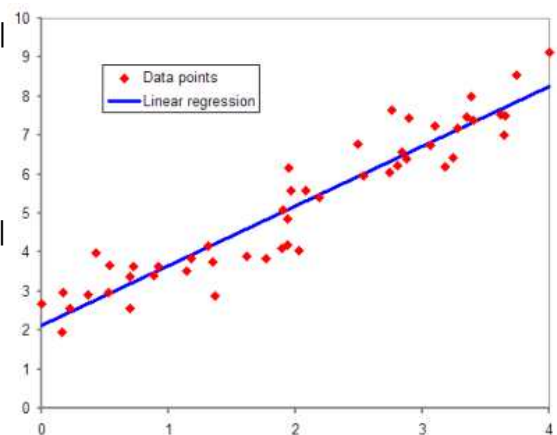
5.2 선형 회귀(linear regresson)

5.2.1 선형 회귀와 로지스틱 회귀

- 단순 선형 회귀 분석
 - 입력 : 특징이 하나
 - 출력 : 하나의 값
 - => ex) 공부 시간으로 시험점수 측정
- 다중 선형 회귀 분석
 - 입력 : 특징이 여러 개
 - 출력 : 하나의 값
 - => ex) 공부 시간, 이전 학기 시험점수, 지금까지의 학점으로 시험 점수 예측
- 로지스틱 회귀
 - 이진 분류 : 두 개로 분류
 - 입력 : 하나 또는 여러 개
 - 출력 : 0 아니면 1
 - => ex) 합격 혹은 불합격으로만 따짐

5.2.2 선형 회귀

- 데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 방법 => 가설
 - $Y = wX + b$ (가중치 w 와 편향 b)
- 공부 시간이 x 라면 점수는 y 이고, 알려진 데이터로부터 x 와 y 의 관계를 유추할 수 있다.
 - 학생이 6시간을 공부했을 때의 성적, 그리고 7시간, 8시간을 공부했을 때의 성적을 예측



5.2.3 손실 함수(Loss function)

- 머신 러닝은 W와 b를 찾기 위해서 손실 함수를 정의한다.
 - 실제 값과 가설로부터 얻은 예측값의 오차를 계산하는 식
 - 손실 함수값을 최소화하는 최적의 W와 b를 찾아내려고 노력
- 손실 함수
 - 목적함수, 비용 함수라고도 부름
 - 예측값의 오차를 줄이는 일에 최적화된 식
 - 평균 제곱 오차(MSE) 사용

5.2.4 손실 함수 : MSE

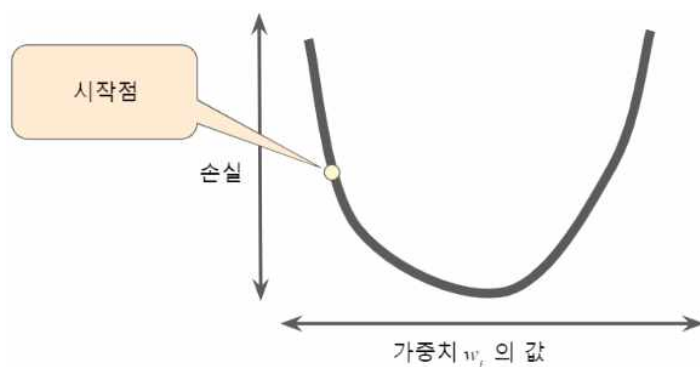
$$cost(W, b) = \frac{1}{n} \sum_i^n [y_i - H(x_i)]^2$$

- 평균 제곱 오차를 W와 b에 의한 비용 함수로 재정의
 - 모든 점들과의 오차가 클수록 평균 제곱 오차는 커지고, 오차가 작아질수록 평균 제곱 오차는 작아짐.
 - cost(W, b)를 최소가 되게 만드는 W와 b를 구하면 결과적으로 y와 x의 관계를 가장 잘 나타내는 직선을 그릴 수 있다.

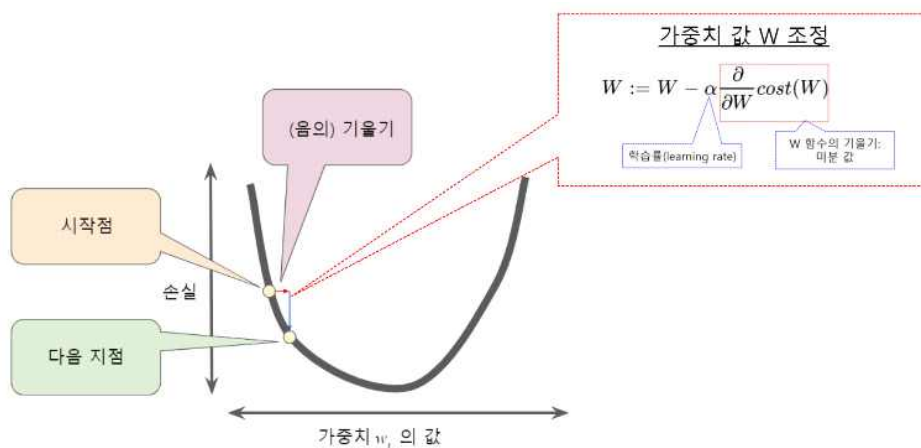
5.3 최적화 과정

5.3.1 옵티마이저(Optimizer) : 최적화 과정

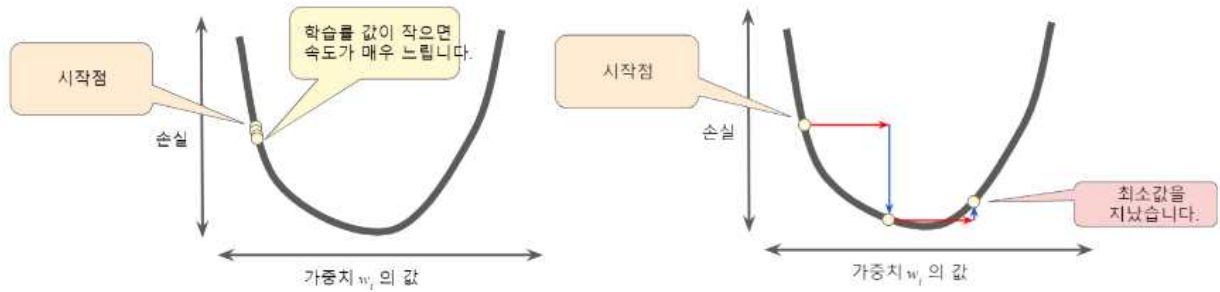
- 최적화 알고리즘
 - 적절한 W 와 b 를 찾아내는 과정 : 경사 하강법 (경사 따라 내려오기)
- 경사하강법은 시작점을 선택한 후 시작점에서 손실 곡선의 기울기를 계산한다.
- 시작점은 별로 중요하지 않아, 많은 알고리즘에서는 0으로 설정하거나 임의의 값을 선택한다.
- 단일 가중치에 대한 손실의 기울기는 미분값과 같다.



- 기울기가 0인 지점을 찾기 위해 기울기의 반대 방향으로 이동한다. 현재의 기울기가 음수이면 다음 가중치 값은 현재의 값보다 크게 조정한다.



- 다음 가중치 값을 결정할 때는 기울기에 학습률을 곱하여 다음 지점을 결정한다. 예를 들어 기울기가 -2.5 이고, 학습률이 0.01 이면 $w = w - (-2.5 * 0.01) = w + 0.025$ 가 된다. 그러므로 이전 지점으로부터 0.025 떨어진 지점을 다음 지점으로 결정한다.
- 학습률의 값은 너무 작게 설정하면 학습 시간이 너무 오래 걸리고, 너무 크게 설정하면 다음 지점이 곡선의 최저점을 무질서하게 이탈할 우려가 있다.



5.3.2 오차역전파

- 순전파
 - 입력층에서 출력층으로 계산하여 최종 오차를 계산
- 역전파
 - 오차 결과값을 통해 다시 역(input)방향으로 오차가 적어지도록 보내며 가중치를 수정
 - 처리 속도 증가



PART 6

6.1 케라스 모델 미사용 텐서플로 프로그래밍

6.1.1 변수 Variables

- 딥러닝 학습에서 최적화 과정
 - 모델의 매개변수(parameters) 즉, 가중치 및 편향을 조정하는 것
- 변수 `tf.Variable` : 프로그램에 의해 변화하는 공유된 지속 상태를 표현하는 가장 좋은 방법
 - 하나의 텐서를 표현
 - 텐서값은 텐서에 연산을 수행하여 변경 가능
 - 모델 파라미터를 저장하는데 사용
- 변수 생성
 - 변수 생성하려면 단순히 초기값을 설정

```
# a와 b를 랜덤한 값으로 초기화합니다.
# a = tf.Variable(random.random())
# b = tf.Variable(random.random())
a = tf.Variable(tf.random.uniform([1], 0, 1))
b = tf.Variable(tf.random.uniform([1], 0, 1))
```

6.1.2 메소드 `minimize()`

- 첫 번째 인자 : 최소화할 손실 함수
- 두 번째 인자 `var_list` : 학습시킬 변수 리스트, 가중치와 편향
- 1000번의 학습을 거쳐 잔차의 평균을 최소화하는 적절한 값 `a, b`에 도달 기대

```
for i in range(1000):
    # 잔차의 평균을 최소화(minimize)합니다.
    optimizer.minimize(compute_loss, var_list=[a,b])
```

4.4 텐서플로를 이용해서 회귀선 구하기

```
import tensorflow as tf
import numpy as np
# import random
```

```
X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85, -0.41, -
0.27, 0.02, -0.76, 2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21.
94, 12.83, 15.51, 17.14, 14.42]
```

```
# a와 b를 랜덤한 값으로 초기화합니다.
# a = tf.Variable(random.random())
# b = tf.Variable(random.random())
a = tf.Variable(tf.random.uniform([1], 0, 1))
b = tf.Variable(tf.random.uniform([1], 0, 1))
```

```
# 잔차의 평균을 반환하는 함수입니다.
def compute_loss():
    y_pred = a * X + b
    loss = tf.reduce_mean((Y - y_pred) ** 2)
    return loss
```

```
optimizer = tf.keras.optimizers.Adam(lr=0.07)
for i in range(1000):
    # 잔차의 평균을 최소화(minimize)합니다.
    optimizer.minimize(compute_loss, var_list=[a,b])
```

```
if i % 100 == 99:
    print(i, 'a:', a.numpy(), 'b:', b.numpy(), 'loss:', compute_loss().numpy())
```

```
99 a: [0.09661794] b: [7.1642118] loss: 81.94983
199 a: [-0.13748273] b: [11.560307] loss: 26.625612
299 a: [-0.2691387] b: [14.037779] loss: 12.436548
399 a: [-0.32794657] b: [15.1445] loss: 10.055599
499 a: [-0.34860265] b: [15.533232] loss: 9.79928
599 a: [-0.35433018] b: [15.641014] loss: 9.781603
699 a: [-0.35558546] b: [15.664643] loss: 9.780827
799 a: [-0.35580176] b: [15.668713] loss: 9.780804
899 a: [-0.355831] b: [15.66926] loss: 9.780804
999 a: [-0.3558332] b: [15.669303] loss: 9.780804
```

6.2 예측하기

6.2.1 보스턴 주택 가격 예측

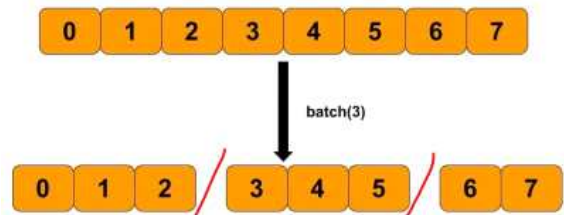
- 506개 타운의 주택 가격 중앙값, 천 달러 단위, 13가지 특성
- 학습 데이터 404개, 테스트 데이터 102개



```
[17] 1 # 4.11 데이터 불러오기
      2 from tensorflow.keras.datasets import boston_housing
      3 (train_X, train_Y), (test_X, test_Y) = boston_housing.load_data()
      4
      5 print(train_X.shape, test_X.shape)
      6 print(train_X[0])
      7 print(train_Y[0])
```

```
(404, 13) (102, 13)
[ 1.23247 0. 8.14 0. 0.538 6.142 91.7
 3.9769 4. 307. 21. 396.9 18.72 ]
15.2
```

- validation_split: 검증용 데이터의 비율
- 만약 2면 훈련:검증 == 80:20 비중으로 준비
- batch_size: 훈련에서 가중치와 편향의 패러미터를 수정하는 데이터 단위 수



- train_size: 훈련 데이터 수

```
# 4.14 회귀 모델 학습
history = model.fit(train_X, train_Y, epochs=25,
                    batch_size=32,
                    validation_split=0.25)
```

6.2.2 자료의 정규화

- 특성의 단위가 다름: 비율, 0/1, 양수 등
- 정규화 방법
 - 학습 데이터: $(\text{train_X} - \text{학습 데이터 평균}) / \text{학습 데이터 표준편차}$
=> 정규 분포를 가정
 - 테스트 데이터: $(\text{test_X} - \text{학습 데이터 평균}) / \text{학습 데이터 표준편차}$
=> 테스트 데이터가 정규 분포를 가정할 수 없다.

```
# 4.12 데이터 전처리 (정규화)
x_mean = train_X.mean(axis=0)
x_std = train_X.std(axis=0)
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])
```

6.2.3 자동으로 학습 중단

- 검증 손실(val_loss)이 적을수록 테스트 평가의 손실도 적음
 - 과적합에 의해 검증 손실이 증가하면 학습이 중단되도록 지정 : 함수 callbacks 사용

6.2.4 일찍 멈춤 기능

- `tf.keras.callbacks.EarlyStopping`
 - `monitor = 'val_loss'` : 지켜볼 기준값이 검증 손실
- `patience = 3`
 - 3회의 epochs를 실행하는 동안 기준값이 최고 기록을 갱신하지 못하면(더 낮아지지 않으면) 학습을 멈춤

```
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25,  
                    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3, monitor='val_loss')])
```

```
Epoch 9/25  
10/10 [=====] - 0s 4ms/step - loss: 0.2058 - val_loss: 0.1839  
Epoch 10/25  
10/10 [=====] - 0s 5ms/step - loss: 0.1620 - val_loss: 0.1748  
Epoch 11/25  
10/10 [=====] - 0s 5ms/step - loss: 0.1459 - val_loss: 0.1848  
Epoch 12/25  
10/10 [=====] - 0s 4ms/step - loss: 0.1124 - val_loss: 0.2126  
Epoch 13/25  
10/10 [=====] - 0s 4ms/step - loss: 0.1683 - val_loss: 0.4185
```

=> 10 에폭의 기록인 .1748 이후 13 에폭에서도 그 기록을 갱신하지 못했으므로 학습 중단

6.3 seaborn

6.3.1 seaborn 설치와 모듈 가져오기

- seaborn 설치
 - !pip install seaborn
 - !pip install -q seaborn
 - pip install seaborn
- 모듈 가져오기

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

6.3.2 Auto MPG 데이터셋

- 판다스를 사용하여 데이터 읽기

```
col_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight', 'Acceleration',
             'Model Year', 'Origin']
raw_data = pd.read_csv(dataset_path,
                       names=col_names, na_values = "?", comment='\t', sep=" ",
                       skipinitialspace=True)
dataset = raw_data.copy()
dataset.tail(10)
```

=> names는 열 이름, na_values는 값이 없거나 잘못 들어간 것들, sep=" " 전까지가 하나의 컬럼 데이터라는 것을 알려주고, skipinitialspace은 공백이 있으면 뛰어넘어가라는 것을 의미

6.3.3 데이터셋을 훈련 세트와 테스트 세트로 분할

```
# 데이터셋을 훈련 세트와 테스트 세트로 분할
# 전체 자료에서 80%를 훈련 데이터로 사용
train_dataset = dataset.sample(frac=0.8, random_state=0)
print(train_dataset)
# 전체 자료에서 나머지 20%를 테스트 데이터로 사용
test_dataset = dataset.drop(train_dataset.index)
print(test_dataset)
```

=> 80:20으로 분할

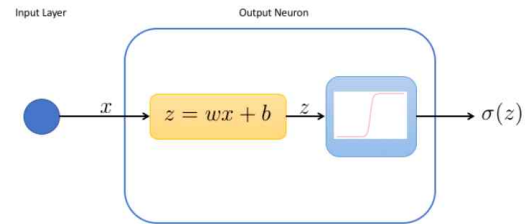


PART 7

7.1 이항 분류 및 다항 분류

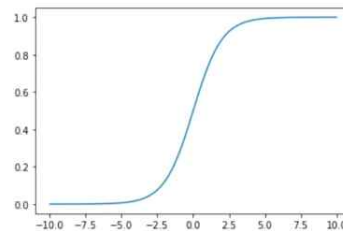
7.1.1 이진(이항) 분류

- 두 가지로 분류
 - PASS / FAIL
 - POSITIVE / NEGATIVE
 - 로지스틱 회귀로도 불림
 - 결과 기술 방식 : 4개의 결과
 - 시그모이드 함수 : 0~1 사이의 값 출력



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

x = np.linspace(-10, 10, 100)
y = 1 / ( 1 + np.exp(-x) )
plt.plot(x, y)
plt.show()
```



5.1 와인 데이터셋 불러오기

```
import pandas as pd
red = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-red.csv', sep=';')
white = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-white.csv', sep=';')
print(red.head())
print(white.head())
```

5.2 와인 데이터셋 합치기

```
red['type'] = 0
white['type'] = 1
print(red.head(2))
print(white.head(2))
```

=> 메소드 pd.concat으로 두 데이터 합침

```
wine = pd.concat([red, white])
print(wine.describe())
```

```
fixed acidity  volatile acidity  citric acid  ...  alcohol  quality  type
0             7.4              0.70        0.0  ...      9.4         5         0
1             7.8              0.88        0.0  ...      9.8         5         0

[2 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 rows x 13 columns]
fixed acidity  volatile acidity  ...  quality  type
count      6497.000000      6497.000000  ...  6497.000000  6497.000000
mean         7.215307         0.339666  ...    5.816378    0.753886
std          1.296434         0.164636  ...    0.879255    0.430779
min           3.800000         0.080000  ...    3.000000    0.000000
25%           6.400000         0.230000  ...    5.000000    1.000000
50%           7.000000         0.290000  ...    6.000000    1.000000
75%           7.700000         0.400000  ...    6.000000    1.000000
max          15.900000         1.580000  ...    9.000000    1.000000

[8 rows x 13 columns]
```


7.1.2 다중 분류

- 소프트맥스 함수
 - 분류의 마지막 활성화 함수로 사용 : 모든 y의 합은 1

7.1.3 crossentropy 손실 함수

- tf.keras.losses.categorical_crossentropy : 원핫 인코딩
 - y_true = [[0, 1, 0], [0, 0, 1]]
- tf.keras.losses.sparse_categorical_crossentropy : 일반
 - y_true = [[1], [2]]

```
import tensorflow as tf

y_true = [[1], [2]]
y_true = tf.one_hot(y_true, depth=3)
print(y_true)
y_true = tf.reshape(y_true, [-1, 3])
print(y_true)
y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]

loss = tf.keras.losses.categorical_crossentropy(y_true, y_pred)
loss.numpy()

tf.Tensor(
  [[[0. 1. 0.]]], shape=(2, 1, 3), dtype=float32)
tf.Tensor(
  [[0. 1. 0.]
   [0. 0. 1.]], shape=(2, 3), dtype=float32)
array([0.05129331, 2.3025851 ], dtype=float32)
```

=> 일반값을 원핫으로 변환

7.1.4 학습 데이터와 테스트 데이터 분리

```
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]
```

정답 제외 정답만

7.1.5 조건에 맞는 새로운 열 추가

- df.loc[data['컬럼']] 조건, '새로운 컬럼명' = '값'

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


7.1.6 Matplotlib

- subplot : 행, 열, 순번 순인데 순번이 0이 아닌 1부터 시작
- plt.subplot(2, 2, 1)

```
import numpy as np

x = np.arange(0, 5, 0.01)

plt.figure(figsize=(10, 12))

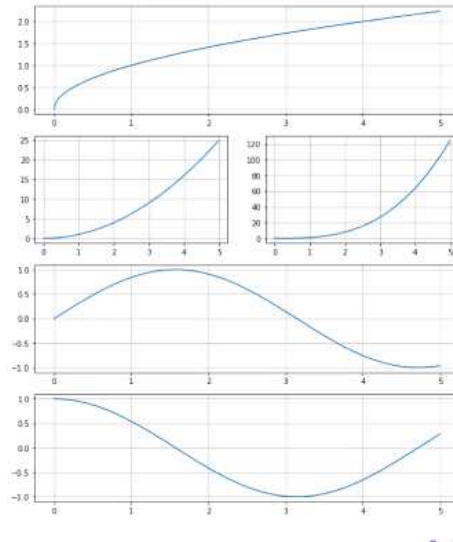
plt.subplot(411)
plt.plot(x, np.sqrt(x))
plt.grid()

plt.subplot(423)
plt.plot(x, x**2)
plt.grid()

plt.subplot(424)
plt.plot(x, x**3)
plt.grid()

plt.subplot(413)
plt.plot(x, np.sin(x))
plt.grid()

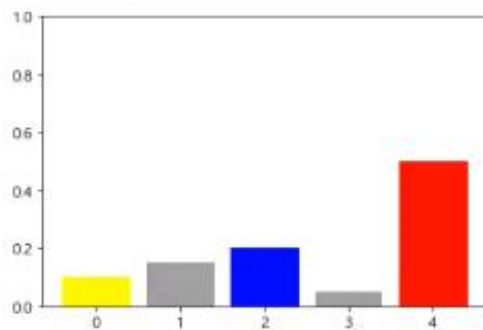
plt.subplot(414)
plt.plot(x, np.cos(x))
plt.grid()
```



7.1.7 막대 그래프

- plt.bar

```
1 predictions_array = [.1, .15, .2, .05, .5]
2 predicted_label = np.argmax(predictions_array) # 4
3 true_label = 2;
4
5 # 막대 그래프 그리기 (x, y, 막대 색상)
6 thisplot = plt.bar(range(5), predictions_array, color="#999999")
7 # 부분 막대의 색상 수정
8 thisplot[0].set_color('yellow')
9 thisplot[predicted_label].set_color('red')
10 thisplot[true_label].set_color('blue')
11
12 plt.ylim([0, 1])
13 plt.show()
```



마무리글...

수업을 들은 후 복습의 의미로 이번 레포트를 작성하게 되었다.

수업 교재뿐만 아니라 유튜브, 블로그 등을 통해 다양한 자료들을 통해 조사하고, 참고하여 정리하였다.

또한, 정리에 그치지 않고, 정리한 것을 다시 읽으며 밑줄을 치으로써 중요한 부분들이 더 쉽게 눈에 띄어서 더 많이 읽으며 복습할 수 있도록 했다.

물론 과제이긴 하지만 과제라는 생각은 잊고, 블로그들을 참고하며 작성하는 것부터 도움이 되었다.

남은 시간 동안 이해가 안 된 부분은 반복적으로 보며, 이해하고 다음 부분으로 넘어갈 수 있도록 할 것이다.

이번 포트폴리오를 계기로 머릿속에 엉켜있던 앞서 배웠던 내용들도 조금이나마 정리할 수 있었다.

참고 : Google