20160774 김창우

인공지능프로젝트 포토폴리오

INDEX

AI의 시작

앨런 튜링

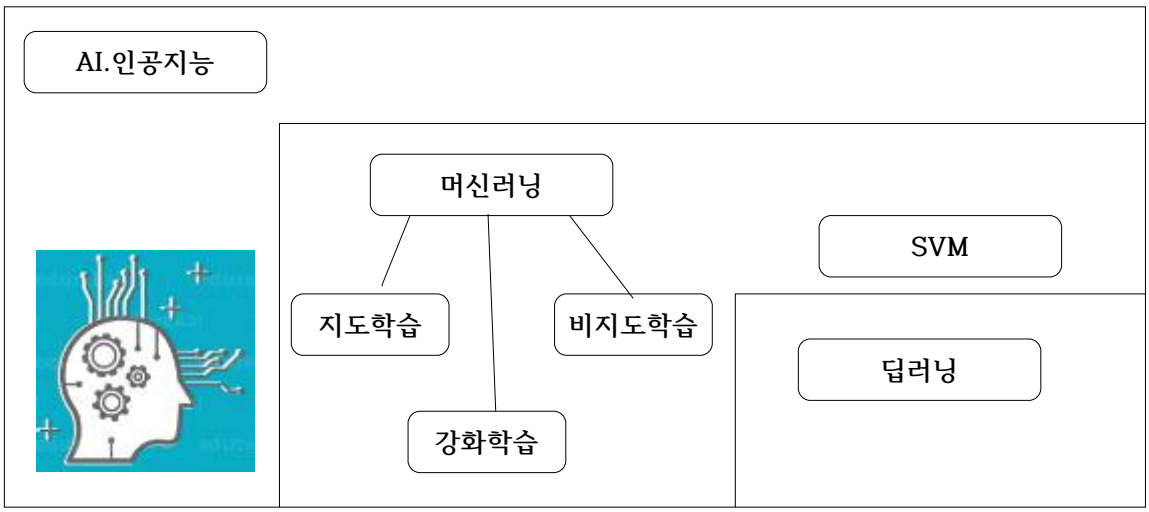
1950년에 논문 <Computing machinery and intelligence>은 생각하는 기계의 구현 가능성에 대한 내용과 인공지능 실험, 튜링 테스트 텍스트로 주고받는 대화에서 기계가 사람인지 기계인지 구별할 수 없을 정도로 대화를 잘 이끌어 간다면, 이것은 기계가 "생각" 하고 있다고 말할 충분한 근거를 담은 논문을 발표하였다.

AI와 딥러닝의 역사는 1940년대부터 시작되어서 두 번의 혹한기를 보냈다 첫 번째로는 (1969~1980)에 마빈 민스키의 인공 신경망인 퍼셉트론에 대한 비판으로 촉발되었고 두 번째는 (1987~1993)에 규칙 기반의 전문가시스템의 의구심과 한계가 발생하여 암흑기를 맞이하였다.

처음 인공지능의 사용은 1956년 다트머스대 학술대회에서 세계 최초의 AI 프로그램인 논리 연산기를 발표하였다.
요즘 시대에는 2000년대 이전의 딥러닝의 문제가 해결되고 있는 중인데 데이터가 부족한 예전과 다르게 빅데이터가 출현하여 해결하고 처리 계산속도가 증가하고 알고리즘도 출현하여 문제들을 해결하여 인기가 많아지는 요즘이다

인공지능이란?(AI: Artificial Intelligence)

컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 기술이라 정의 내릴 수 있다.



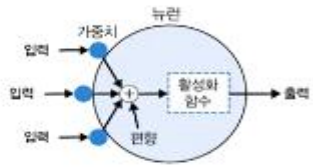
머신러닝(machine learning)
기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야이고 많은 알고리즘이 존재하는데 그 중에 SVM(Support Vector Machine)
수학적인 방식의 학습 알고리즘을 사용하고 또 딥러닝이 존재한다
주어진 데이터를 기반으로 기계가 스스로 데이터를 반복적으로 학습하고 성능을 향상시키거나 최적의 해답을 찾기 위한 학습 지능 방법이다.

지도학습(Supervised learning)
올바른 입력과 출력의 쌍으로 구성된 정답의 훈련데이터(labeled data)로부터 입출력 간의 함수를 학습시키는 방법
-k-최근접 이웃 (k-Nearest Neigbors)
-선형 회귀 (Linear Regression)
-로지스틱 회귀 (Logistic Regression)
-서포트 벡터 머신 (Support Vector Machines (SVM))
-결정 트리 (Decision Tree)와 랜덤 포레스트(Random Forests)

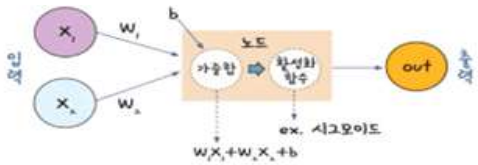
비지도(자율)학습(unsupervised learning)
정답이 없는 훈련 데이터 (unlabeled data)를 사용하여 데이터 내에 숨어있는 어떤 관계를 찾아내는 방법
-clustering

강화학습(reinforcement learning)
잘한 행동에 대해 보상을 주고 잘못한 행동에 대해 벌을 주는 경험을 통해 지식을 학습하는 방법
-딥마닝의 알파고
-자동 게임분야

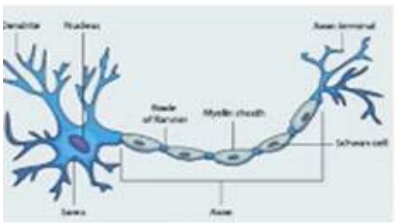
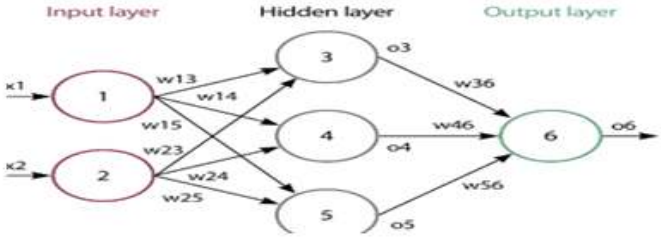
퍼셉트론(perceptron)
코넬대 교수 프랭크 로젠블랫이 세계 최초의 인공신경망과 데이터를 정확하게 구분하도록 시스템을 학습시켜 원하는 결과를 얻어냄
현재 발전해 여러 분야에서 활용되는데 항공기 드론의 자율비행과 필체 인식 음성인식, 언어 번역이 가능하다.



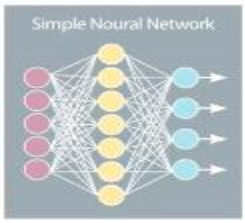
인공신경망(ANN;Artificial Nerual Network)
인간의 뇌를 구성하는 신경세포 뉴런의 동작원리에 기초한 기술이며 뇌와 유사한 방식으로 입력되는 정보를 학습하고 판별한다.

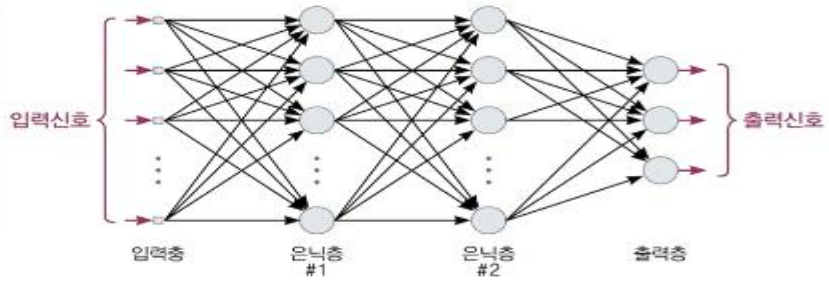


MLP(Multi Layer Perceptron)
입력층 중간의 은닉층 출력층으로 구성되어 있으며 여러 개의 층으로 연결하여 하나의 신경망과 하나의 신호를 출력한다

심층신경망(deep neural network)
다중 계층인 심층신경망을 사용하고 학습 성능을 높이는 고유 특징들만 스스로 추출하여 학습하는 알고리즘과 입력 값에 대해 여러 단계의 심층 신경망을 거쳐 자율적으로 사고 및 결론을 도출하는 방법





딥러닝의 활용
다중 계층의 신경망 구조로 이루어진 딥마인드의 알파고와
인간과 대화하는 지능형 에이전트와 실시간 채팅이 가능한 챗봇 드론의 자율비행이나 자동차의 자율주행 분야 주식이나 펀드,환율,일기예보 등의 예측 분야에서 활용

그래픽 처리 장치 GPU(Graphics Processing Unit)
그래픽 연산 처리를 하는 전용 프로세서
GPGPU(General Purpose Graphic Processing Unit)
일반 CPU 프로세서를 돕는 보조프로세서 GPU 딥러닝 심층신경망에서 빅데이터를 처리하기 위해 대량의 행렬과 벡터를 사용

딥러닝 다중 계층의 신경망 모델을 사용하는 머신러닝들은 특징과 데이터가 많을수록 딥러닝에 적합하다.

캐라스

독자적인 고수준 라이브러리 엔진으로 텐서플로,씨아노,CNTK을 사용하고 동일한 코드로 CPU와 GPU에서 실행 가능하고 쉬운 API를 가지고 있어 딥러닝 모델의 프로토타입을 빠르게 생성 가능하다.

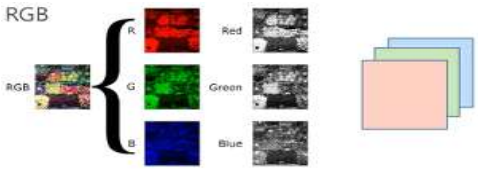
텐서플로(TensorFlow)

구글에서 만든 라이브러리로 연구 및 프로덕션용 오픈소스 딥러닝으로 자주 사용되고 쉽게 구현할 수 있도록 데스크톱, 모바일, 웹, 클라우드 개발용 API를 제공한다.
Python, Java, Go 등 다양한 언어를 지원해주고 이중 파이썬을 최우선으로 지원해준다.

텐서(Tensor)

모든 데이터이며 딥러닝에서 데이터를 표현하는 방식
- 0-D 텐서:스칼라 차원이 없는 텐서 ex) 10
- 1-D 텐서:벡터 1차원 텐서 ex) [10, 20, 30]
- 2-D 텐서:행렬 등 2차원 텐서

텐서의 차원을 텐서의 rank(순위) ex) RGB R(ed), G(reen), B(lue) 각 3개의 채널 마다 2차원 행렬로 표현하는데 이를 텐서로 표현함
[[[1,2, 3] [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]



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

텐서플로(TensorFlow)계산 과정

모두 그래프라고 부르는 객체 내에 저장되어 실행되고 그래프를 계산하려면 외부 컴퓨터에 그래프 정보를 전달하고 결과값을 받아야 함

Session

세션은 텐서플로 계산 과정에서 통신과정을 담당하고 생성 사용 종료 과정이 필요하다.

-세션 생성 Session 객체 생성

```
sess = tf.Session()
```

-세션 사용 run 메서드에 그래프를 입력하면 출력,값을 계산하여 반환

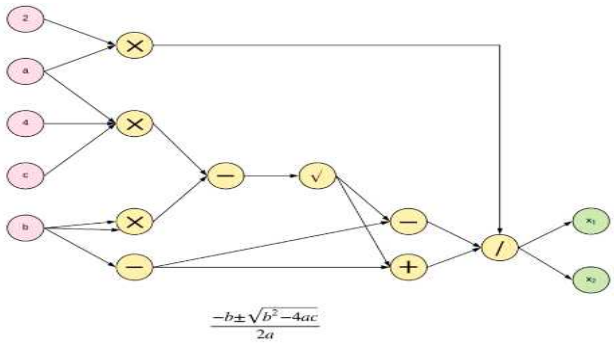
```
print(sess.run(x))
print(sess.run(y))
```

-세션 종료 close 메서드,with 문을 사용하면 명시적으로 호출 불필요

```
sess.close()
```

데이터 흐름 그래프(dataflow graph)

TensorFlow에서 계산으로 이루어 지고 텐서 형태의 데이터들이 딥러닝 모델을 구성하는 연산들의 그래프를 따라 흐르면서 연산이 일어남



Tensor + DataFlow

딥러닝에서 데이터를 의미하는 Tensor와 DataFlow Graph를 따라 연산이 수행되는 형태를 합쳐 TensorFlow란 이름이 탄생

매직 명령어로 tensorflow 불러오기

코랩 버전 사용방법

- %tensorflow_version 1.x

```
%tensorflow_version 1.x
```

- %tensorflow_version 2.x

텐서 2.0으로 실행

이미 1.x을 사용 중이라면 메뉴 런타임 다시 시작하고 이후 그대로 import

텐서 출력 메소드 numpy()

```
[1] import tensorflow as tf
    tf.__version__

'2.3.0'
```

```
[3] c = tf.constant('Hello, world!')
    print(c)
    print(c.numpy())

tf.Tensor(b'Hello, world!', shape=(), dtype=string)
b'Hello, world!'
```

```
[2] a = tf.constant(5)
    b = tf.constant(3)
    print(a+b)

tf.Tensor(8, shape=(), dtype=int32)
```

```
[12] a = tf.constant([1, 2, 3])
     a.shape

TensorShape([3])
```

벡터를 의미하고 크기를 나타냄

```
[13] a = tf.constant([[1, 2, 3], [4, 5, 6]])
     a.shape

TensorShape([2, 3])
```

```
[[1, 2, 3],
 [4, 5, 6]]]
```

2행 3열 (2x3) 결과를 나타냄

```
[14] a = tf.constant([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
     a.shape

TensorShape([2, 2, 3])
```

맨 밖에 차수가 2개, 2행 3열 결과를 나타냄

1차원 배열 텐서

```
[14] # 1차원 배열 텐서
     t = tf.constant([1, 2, 3])
     t
```

```
<tf.Tensor: shape=(3,), dtype=int32, numpy=array([1, 2, 3], dtype=int32)>
```

원소의수, 즉 3개원소가 존재하는 튜플 shape=(3,)

```
[15] x = tf.constant([1, 2, 3])
     y = tf.constant([5, 6, 7])

     print((x+y).numpy())
```

```
[ 6  8 10]
```

각각의 원소를 위치에 맞게 더한 값

```
[16] a = tf.constant([5], dtype=tf.float32)
     b = tf.constant([10], dtype=tf.float32)
     c = tf.constant([2], dtype=tf.float32)
     print(a.numpy())

     d = a * b + c

     print(d)
     print(d.numpy())
```

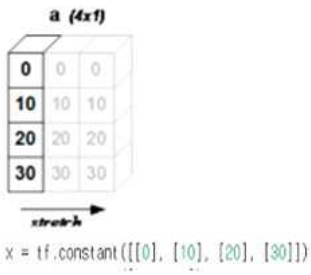
```
[5.]
tf.Tensor([52.], shape=(1,), dtype=float32)
[52.]
```

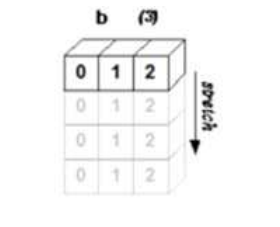
d의값 numpy()=5

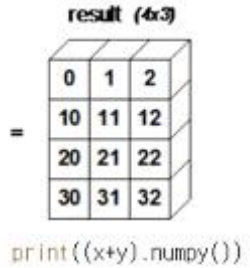
a*b+c=52 값 1차원 배열이라 shape(1,)

numpy()호출=52

브로드캐스팅 코드



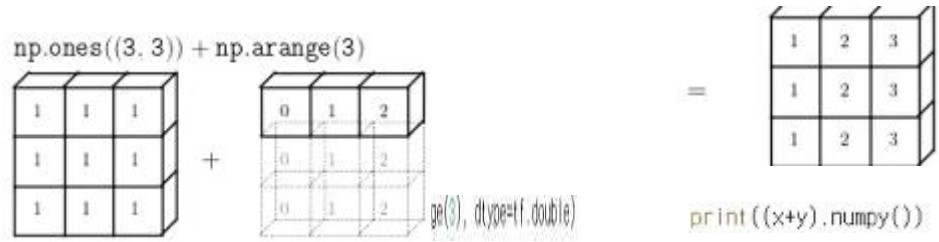




원소와 원소끼리 더하기

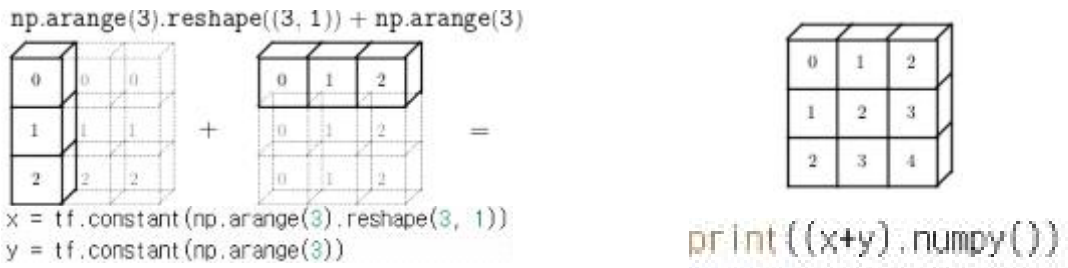

```
[5 6 7]
```

[0, 1, 2]를 [5] 한 번씩 덧셈




```
x = tf.constant((np.ones((3, 3))))
[[1. 2. 3.]
 [1. 2. 3.]
 [1. 2. 3.]]
```

np.ones((3,3)이랑 arange(3)을 한 번씩 덧셈




```
[[0 1 2]
 [1 2 3]
 [2 3 4]]
```

np.arange(3)은 [0, 1, 2]인데 reshape(3,1)로 인해 3행 1열로 바꾸고 arange(3)을 한 번씩 덧셈

행렬 곱셈

tf.matmul() 2차원 행렬 곱셈

```
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.]]
```

1*2+2*1 1*0+2*2

3*2+4*1 3*0+4*2

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

```
[[ 2.  4.]
 [ 7. 10.]]
```

2*1+0*3 2*2+0*4

1*1+2*3 1*2+2*4

tf.rank

행렬의 차수 반환

```
my_image = tf.zeros([2, 5, 5, 3])
my_image.shape
```

```
TensorShape([2, 5, 5, 3])
```

zero가 들어간 행렬을 만듬, 원소의 수: 2x5x5x3

```
tf.rank(my_image)
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=4>
```

4차원 행렬인 이미지의 차수인 4를 가져옴 numpy=4

Shape과 reshape

```
rank_three_tensor = tf.ones([3, 4, 5])
rank_three_tensor.shape
```

```
TensorShape([3, 4, 5])
```

ones로 인해 다 1을 갖으며 3차원이고 3,4,5의 모양을 갖는다.

```
rank_three_tensor.numpy()
```

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

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

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

내용물을 호출하면 실수로 호출되는데 4행 5열이 3개라는 결과가 나타난다

```
matrix = tf.reshape(rank_three_tensor, [6, 10])
matrix
```

```
<tf.Tensor: shape=(6, 10), dtype=float32, numpy=
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]], dtype=float32)>
```

reshape로 인해 6행 10열로 모양을 바꾸는 결과 발생

```
matrix8 = tf.reshape(matrix, [3,  1])
matrix8
```

기존 내용을 3x20 행렬로 형태 변경

1은 처음 그기를 계산하여 자동으로 결정하라는 의미

matrix를 3x20=60 열이 20개로 만들어짐

```
<tf.Tensor: shape=(3, 20), dtype=float32, numpy=
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1.]], dtype=float32)>
```

텐서플로 난수

균등분포 난수

-tf.random.uniform([1], 0, 1) 배열 [시작, 끝]

```
rand = tf.random.uniform([1],0,1)
print(rand)
```

```
tf.Tensor([0.5543064], shape=(1,), dtype=float32)
```

맨앞에 1은 몇 개와 구조를 만들지 형태를 정하고 0은 시작 1은 끝

```
rand = tf.random.uniform([5, 4],0,1)
print(rand)
```

```
tf.Tensor(
[[0.43681145 0.84187937 0.9562702  0.7846168 ]
 [0.6079582  0.95665395 0.9038415  0.19482386]
 [0.51012075 0.8609252  0.9433547  0.9636686 ]
 [0.2134043  0.9559026  0.5170028  0.4017253 ]
 [0.0141474  0.15945261 0.23697984 0.7221806 ]], shape=(5, 4), dtype=float32)
```

5행 4열을 만들고 0~1 사이에서 실수를 만들어지는 결과

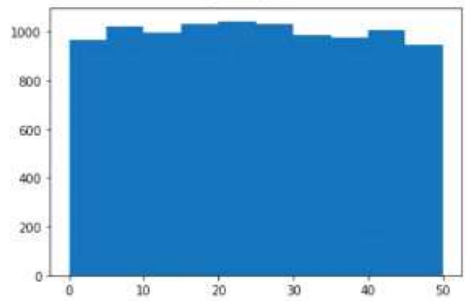
균등 분포 1000개 그리기

```
import matplotlib.pyplot as plt
```
plt를 사용하여 균등 분포 생성

```
rand = tf.random.uniform([10000],0,50)
```
난수를 0~50사이에서 생성

```
plt.hist(rand, bins=10)
```
막대그래프를 그리는 hist, bins는 막대그래프의 개수

```
(array([ 965., 1020.,  994., 1032., 1043., 1030.,  987.,  976., 1008.,
         945.]),
 array([6.0786738e-04, 4.9998469e+00, 9.9990854e+00, 1.4998324e+01,
        1.9997562e+01, 2.4996801e+01, 2.9996040e+01, 3.4995281e+01,
        3.9994518e+01, 4.4993759e+01, 4.9992996e+01], dtype=float32),
 <a list of 10 Patch objects>)
```



정규분포 난수

-tf.random.uniform([4], 0, 1)

```
rand = tf.random.normal([4],0,1)
print(rand)
```

4개의 크기만큼 만들고 0은 평균 1은 표준편차를 선언

```
tf.Tensor([-0.5962639   0.47093895  1.9455601  -0.42773333], shape=(4,), dtype=float32)
```

4개중에서 0~1사이의 평균을 구함

```
rand = tf.random.normal([2, 4],0,2)
print(rand)
```

2행 4열로 만들고 평균은 0 표준편차 2로 만들어 0~2사이에 값을 구함

정규 분포 1000개 그리기

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

plt를 사용하여 정규 분포 생성

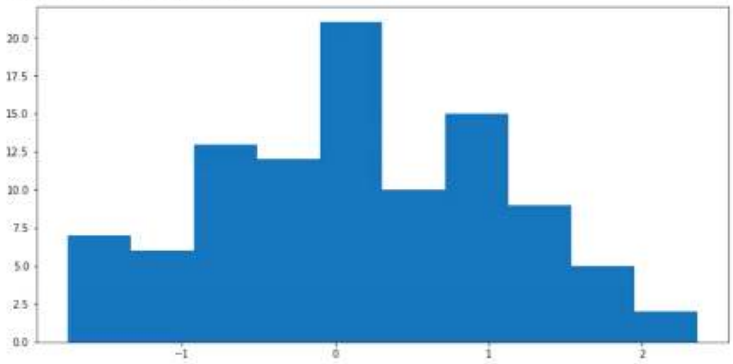
```
rand = tf.random.normal([100], 0, 1)
```

normal로 100개의 크기를 구하고 0~1사이 평균을 구함

```
plt.hist(rand, bins=10)
```

막대그래프를 그리는 hist,bins는 막대그래프의 개수

```
(array([ 7.,  6., 13., 12., 21., 10., 15.,  9.,  5.,  2.]),
 array([-1.7424843 , -1.332153  , -0.921821  , -0.511489  , -0.10115702,
         0.30917495,  0.7195069 ,  1.129839  ,  1.5401709 ,  1.9505029 ,
         2.3608348 ], dtype=float32),
 <a list of 10 Patch objects>)
```



균등분포와 정규분포의 비교

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

plt를 이용하여 생성

```
rand1 = tf.random.normal([1000],0, 1)
```

normal을 이용하여 0~1사이 1000개의 정규분포를 생성

```
rand2 = tf.random.uniform([2000], 0, 1)
```

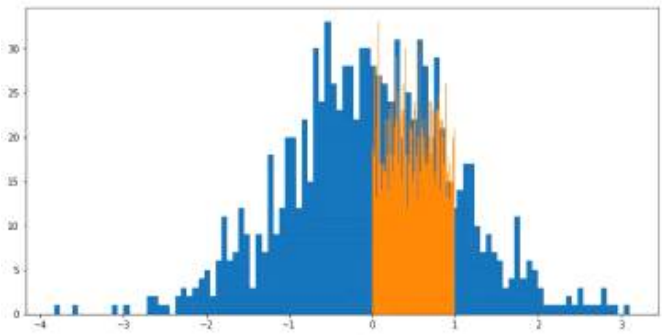
uniform을 이용하여 0~1사이 2000개의 균등분포를 생성

```
plt.hist(rand1, bins=100)
```

정규분포의 막대그래프 개수를 100개 생성

```
plt.hist(rand2, bins=100)
```

균등분포의 막대그래프 개수를 100개 생성



shuffle

-tf.random.shuffle(a)

```
import numpy as np
```

np로 객체 선언

```
a = np.arange(10)
```

0에서 10까지

```
print(a)
tf.random.shuffle(a)
```

0에서 10까지 랜덤한 숫자

```
[0 1 2 3 4 5 6 7 8 9]
<tf.Tensor: shape=(10,), dtype=int64, numpy=array([7, 9, 1, 4, 3, 5, 8, 6, 2, 0])>
```

```
a = np.arange(20).reshape(4, 5)
a
```

20개의 숫자중에서 4행 5열을 만듬

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

```
tf.random.shuffle(a)
```

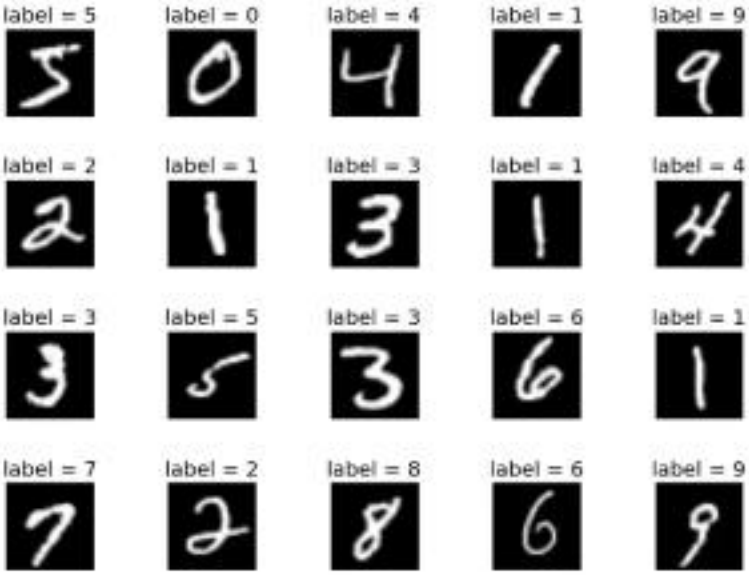
위에 받아온 값을 shuffle을 통해 행을 섞는 결과

MNIST 데이터셋

MNIST(Modified National Institute of Standards an Technology)

딥러닝 손글씨 인식에 사용되는 데이터셋

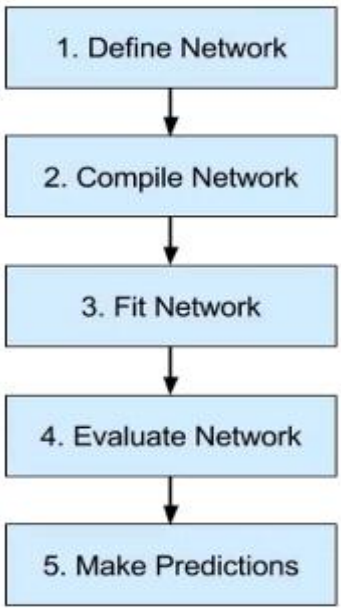
미국 국립 표준 기술원에서 손으로 쓴 자릿수에 대한 데이터 집합을 만듬 픽셀 이미지와 레이블의 쌍으로 구성되어있고 숫자의 범위는 0에서 9까지 총 10개의 패턴을 의미하고 필기는 크기가 28x28 픽셀인 회색조 이미지가 보여진다.
이 두 가지의 데이터를 가져오는 방법으로는
-tensorflow.keras.datasets.mnist
-tensorflow.examples.tutorials.mnist 가 존재한다.



케라스 딥러닝 구현

5개의 과정

- 딥러닝 모델을 만드는 define
- 주요 훈련방법을 설정 compile
- 훈련 fit
- 테스트 데이터를 평가 evaluate
- 정답을 예측 predict



MNIST 손글씨 데이터 로드 코드

-훈련 데이터 손글씨와 정답
x_train, y_train 6만개로 구성
-테스트 데이터 손글씨와 정답
x_test, y_test 1만개로 구성

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
```

mnist 변수를 선언하여 불러옴

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

MNIST 데이터셋을 훈련과 테스트 데이터로 로드함

MNIST 손글씨 데이터 구조 확인

```
1 x_train.shape
```

```
(60000, 28, 28)
```

학습 문제의 크기를 요청 픽셀은 28x28

```
1 y_train.shape
```

```
(60000,)
```

정답은 60000개가있는 정수 벡터 값

```
x_train[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
```

x_train[0]은 60000개중 맨 첫 번째 28x28픽셀을 보여줌

```
1 y_train[0]
```

```
5
```

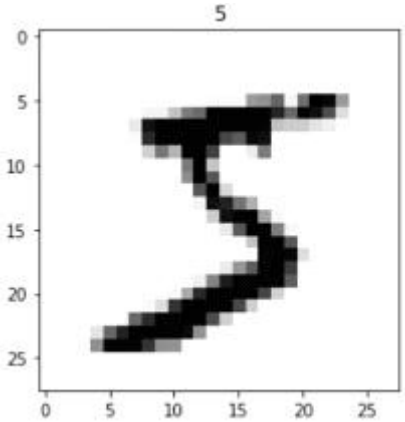
첫 번째 정답 5

```
n = 0
ttl = str(y_train[n])
```

ttl = y_train[n]값을 5를 대입

```
plt.figure(figsize=(6, 4))
plt.title(ttl)
plt.imshow(x_train[n], cmap='Greys')
```

plt.figure 사이즈를 6,4 로 만들고 타이틀값을 5로 대입 색상은 흑백



훈련용 데이터 60000개 중에서 임의 손글씨 출력

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

0~59999중의 임의 수 20개 선택

```
from random import sample
nrows, ncols = 4, 5 #출력 가로 세로 수
```

sample함수를 사용하여 4행 5열로 출력

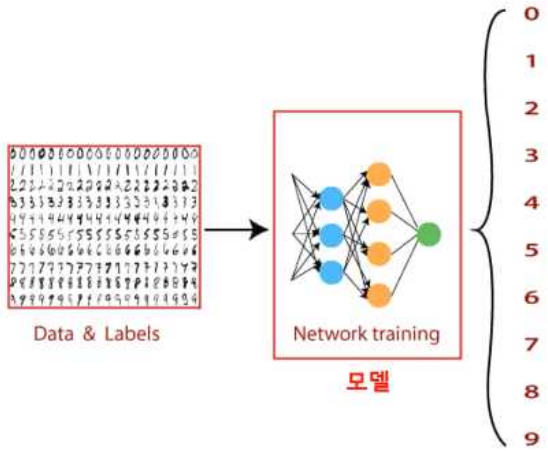
```
idx = sorted(sample(range(len(x_train)), nrows * ncols))
print(idx)
```

```
[577, 4733, 6096, 7075, 15445, 15592, 22448, 22721, 23361, … ]
```

오름차순으로 정렬하여 20개를 출력

MNIST 데이터 셋을 위한 딥러닝

0에서 9까지의 분류 클래스 10개



딥러닝 과정

- 필요 모듈 임포트
- 훈련과 정답 데이터 지정
  (데이터 전처리(옵션))
- 모델구성
- 학습에 필요한 최적화 방법과 손실 함수 등 설정
  (구성된 모델 요약(옵션))
- 생성된 모델로 훈련 데이터 학습
- 테스트 데이터로 성능 평가
  (테스트 데이터 또는 다른 데이터로 결과 예측(옵션))

데이터셋

-훈련용과 테스트용

Train data set, Test data set x(입력문제), y(정답, 레이블)

모델

- 딥러닝 핵심 신경망, 여러 층 구성

Dense():완전연결층 Flatten():1차원 배열로 평탄화

학습 방법의 여러 요소들

- 옵티마이저 최적화 방법 손실 함수

딥러닝 훈련 Epochs, 총 훈련 횟수 훈련 데이터를 모두 훈련 것 1에폭

1.훈련과 정답 데이터 지정

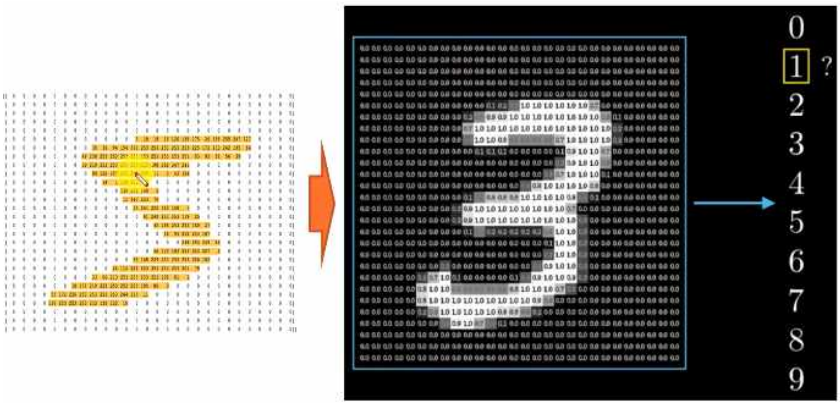
MNIST 데이터셋을 로드하여 준비

```
import tensoflow as tf

mnist = tf.keras.datasets.mnist

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

데이터 셋을 로드하여 준비하는 과정

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

한 비트의 값(x_train)을 0~255로 실수가 되도록 나눔



정규화 결과로 픽셀 값은 0에서 1사이의 값이나옴

2.모델 구성

(신경망 구성)

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

층을 차례대로 쌓아 tf.keras.models.Sequential 모델 생성

```
tf.keras.layers.Flatten(input_shape=(28, 28)),
```

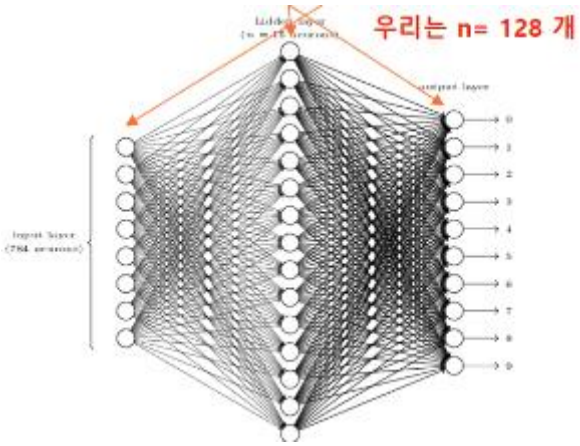
Flatten은 28x28로 평탄화 즉 1차원 배열로 만드는 과정

```
tf.keras.layers.Dense(128, activation='relu'),
```

Dense는 입력 중 히든층으로 만드는 과정 (히든층의 뉴런의수 128 (짐작) activation는 활성화 하는 함수

```
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10, activation='softmax')
```

마지막 단계인 Out put은 0~9까지의 클라스의 수 만큼 작성하고 softmax를 활성화



Neural Networks

- 입력층
- 중간층(은닉층)
- 출력층

단순 신경망 모델과 Dense()층

Dense()는 완전 연결층이며 중간 은닉층이 없는 구조이며 입력층과 출력층만 존재한다.

중간층(은닉층)이 없는 경우

ex) 입력 : 784

-이미지의 각 픽셀 값

ex) 출력 : 10

-각 위치의 값이 될 크기

```
[1.22, .67, .45, .46, .86
.87, .45, .65, 1.14, 2.56]
```

테스트 데이터의 첫 번째 손글씨 예측 결과 확인

model.predict(input)

-input 값 모델의 fit(), evaluate()에 입력과 같은 형태가 필요하고 28x28 이미지가 여러 개인 3차원

-첫 번째 손글씨만 알아보더라도 3차원 배열로 입력

x_test[:1], pred_result=model.predict(x_test[:1])

결과

- 정수: 손글씨 값의 정수

```
(1, 28, 28)
```

- 실제: (1, 10)의 이차원 배열

```
(1, 10)
```

- 결과: 10개의 0~1의 실수

```
[[8.7629097e-12 4.7056760e-14 2.5735870e-12 1.3529770e-07 1.9923079e-21
  1.6554103e-12 2.3112234e-21 9.9999988e-01 2.5956004e-10 3.6446388e-10]]
```

Tensorflow 메소드

tf.reduce_sum(), tf.argmax()

```
import numpy as np


one_pred = pred_result[0]
print(tf.reduce_sum(one_pred))
print(tf.reduce_sum(one_pred).numpy())
```

10개의 수를 더함 1.0

```
print(tf.argmax(one_pred).numpy())
```

가장 큰 수가 있는 첨자의 결과 7

배열에서 가장 큰 값의 첨자 구하기

메소드 np.argmax()

2차원에서 내부 행의 argmax 구하려면 axis=1로 선언

```
print(np.argmax([5, 4, 10, 1, 2]))
```

0, 1, 2, 3, 4

중에서 가장 큰 숫자를 구한 결과 2

```
print(np.argmax([3, 1, 4, 9, 6, 7, 2]))
```

0, 1, 2, 3, 4, 5, 6

중에서 가장 큰 숫자를 구한 결과 3

```
print(np.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```

0, 1, 2 0, 1, 2 0, 1, 2

중에서 가장 큰 숫자를 구한 결과 [1 0 2]

메소드 tf.argmax()

```
print(tf.argmax([5, 4, 10, 1, 2]))
```

0, 1, 2, 3, 4

중에서 가장 큰 숫자를 구한 결과 tf.Tensor(2, shape=(), dtype=int64)

```
print(tf.argmax([3, 1, 4, 9, 6, 7, 2]))
```

0, 1, 2, 3, 4, 5, 6

중에서 가장 큰 숫자를 구한 결과 tf.Tensor(3, shape=(), dtype=int64)

```
print(tf.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```

0, 1, 2 0, 1, 2 0, 1, 2

중에서 가장 큰 숫자를 구한 결과 tf.Tensor([1 0 2], shape=(3,), dtype=int64)

flatten 평탄화 메소드

ary.flatten()

```
import numpy as np

x = np.array([2, 3, 254, 5, 6, 3])
x = x / 255.0
```

np.array로 벡터를 만들어 x로 선언
선언한 x를 255로 나눔

```
print(x)
```

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

6개의 값들을 각각 255를 나눈 결과를 출력

```
x = x.reshape(2, 3)
```

x.reshape(2,3) x행렬을 2행 3렬로 바꾸는 명령어

```
print(x)
```

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

출력 시 2차원인 2행 3렬로 바뀐 결과 출력

```
x = x.flatten()
print(x)
```

인자가 없는 x를 벡터로 만들고 평탄화를 하는 과정

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

드롭아웃

2012년 토론토 대학의 힌튼 교수와 그의 제자들이 개발에 성공했고
훈련 단계보다 더 많은 유닛이 활성화되기 때문에 균형을 맞추기 위해 층의 출력값을 드롭아웃 비율만큼 0으로 만듦
훈련 단계에서만 적용하고 테스트 단계에서는 어떤 유닛도 드롭아웃 하지 않음
ex)[9.2, 0.5, 1.3, 0.8 1.1] 벡터를 출력하면 랜덤하게 0으로 바뀌어서 출력되는 결과 확인 가능

tf.keras.layers.Dropout(0,2)
확률 값은 0.2~0.5를 사용

```
data = np.arange(1, 11).reshape(5, 2).astype(np.float32)
print(data)
```

np.arange(1~10)에서 reshape(5,2)로 인해 실수로 바꾼 값을 출력

```
[[ 1.  2.]
 [ 3.  4.]
 [ 5.  6.]
 [ 7.  8.]
 [ 9. 10.]]
```

```
np.sum(data)
```

```
55.0
```

위에 나온 데이터 값들의 합을 출력

```
tf.random.set_seed(0)
```

난수를 만드는 seed값을 0으로 지정

```
layer = tf.keras.layers.Dropout(.3, input_shape=(2,))
```

Dropout을 만들고 30%를 필요하면 0으로 만들고 input_shape(2,)은 2행으로 만듬

```
outputs = layer(data, training=True)
```

outputs에는 layer(data training=true)를 주어서 Dropout을 적용

```
print(outputs)
```

```
tf.Tensor(
[[ 0.         0.       ]
 [ 4.285714   5.714286 ]
 [ 7.1428576  8.571428 ]
 [10.        11.428572 ]
 [12.857143   0.       ]], shape=(5, 2), dtype=float32)
```

결과를 출력해보면 10개의 원소중에 30%의 확률로 0으로 변환된값을 출력

```
np.sum(outputs)
```

```
60.0
```

위에 나온 데이터 값들의 합을 출력

임의의 20개 예측 값과 정답

- 리스트 pred_result : 모델의 예측 결과, 확률 값
- 리스트 pred-labels : 모델의 예측 결과, 정수
- 리스트 samples : 출력할 20개의 첨자 리스트

pred_labels[n] == y_test[n]
예측값이 맞는 경우에 y_test 훈련 데이터 정답

```
count = 0
plt.figure(figsize=(12,10))
```

plt.figure를 사용하여 figsize의 크기를 12,10으로 만듬

```
for n in samples:
    count += 1
```

난수를 선언하고 count +=1로 선언하여 계속해서 1부터 20까지 진행

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

subplot은 행과 열을 만들어 순서대로 들어갈 그림을 count값을 받아서 1부터20까지 1씩커지게 선언

```
cmap = 'Greys' if ( pred_labels[n] == y_test[n]) else 'Blues'
```

cmap는 if문을 사용하여 예측이 틀린 것은 파란색 맞는 것은 회색으로 출력

```
plt.imshow(x_test[n].reshape(28, 28), cmap=cmap, interpolation='nearest')
```

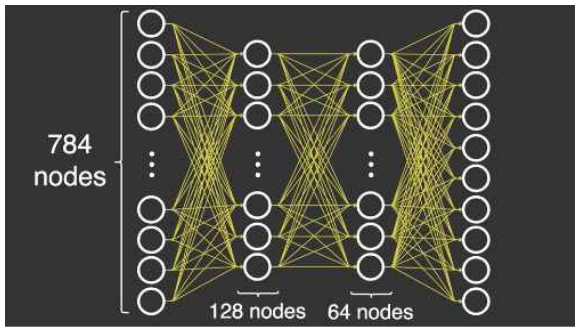
imshow를 이용하여 cmap을 동일하게 보여주고 x_test[n] 출력 시 모양이 바뀐 결과가 출력되면 reshape(28, 28)로 바꾸고 바뀌지 않을시 그대로 출력

```
tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])
```

Label 제목을 str(y_test[n]) 출력시 정답값 Prediction을 예측값을 작성

중간층을 늘리고 훈련 횟수를 증가

중간층 2개 출력층
128개 뉴런 64개 뉴런 10개 출력
훈련 횟수 20회
epochs=20




```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
```

models.Sequential로 모델을 생성하고 크기shape(28.28) Flatten으로 평탄화 과정

```
tf.keras.layers.Dense(128, activation='relu')
```

중간층인 Dense 128


```
tf.keras.layers.Dropout(.2),
```

Dense 128에서 나오는 output을 Dropout(.2)을 사용하여 20%만큼 0으로 만듬

```
tf.keras.layers.Dense(64, activation='relu'),
```

중간층인 Dense 64

```
tf.keras.layers.Dropout(.2),
```

Dense 64에서 나오는 output을 Dropout(.2)을 사용하여 20%만큼 0으로 만듬

```
tf.keras.layers.Dense(10, activation='softmax')
```

Dense에서 마지막층에 10개만듬 activation='softmax'로 작성

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

훈련에 사용할 옵티마이저와 손실 함수 출력 정보를 출력 하는 코드

```
model.summary()
```

모델 요약 표시 하는 코드

```
model.fit(x_train, y_train, epochs=5)
```

모델을 훈련 데이터로 총 5번 훈련 시키는 코드

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

모델을 테스트 데이터로 평가

포트폴리오를 작성하면서 느낀점

군 복무를 마치고 1년 동안 돈 벌다가 다시 학업을 진행하게 되었다. 3년 동안의 공백이 십사 리 채워지지 않을 거라는 생각을 하고 학교를 다니면서 열심히 배우려고 다짐한 1년이었다. 2학기 과목에 최근 들어 인기가 급상승하고 많이 사용되는 AI 인공 지능 프로그래밍이 존재했는데 마침 평소에 흥미를 갖고 배워보고 싶은 과목이라 바로 수강신청을 하고 배우려는 자세를 갖췄다.
그러나 난생처음 보고 낯선 느낌이 없지 않아 있어서 수업에 따라가기에는 어렵지만 마침 과제가 배운 내용들을 스스로 정리하고 검토하여 포트폴리오로 작성하는 과제라 이 시간이 너무 감사하고 복습하는 시간이 되어서 뜻깊은 시간이었고 결코 시간이 아깝지 않았다. 작성하면서 어려운 부분들도 많았지만 스스로 터득하고 알아갈 수 있어서 보람찬 기회였다.