REPORT

지능형 컴퓨팅과정 포트폴리오

담당교수 : 강환수 교수님

학과 : 컴퓨터정보공학과

학번 : 20161852

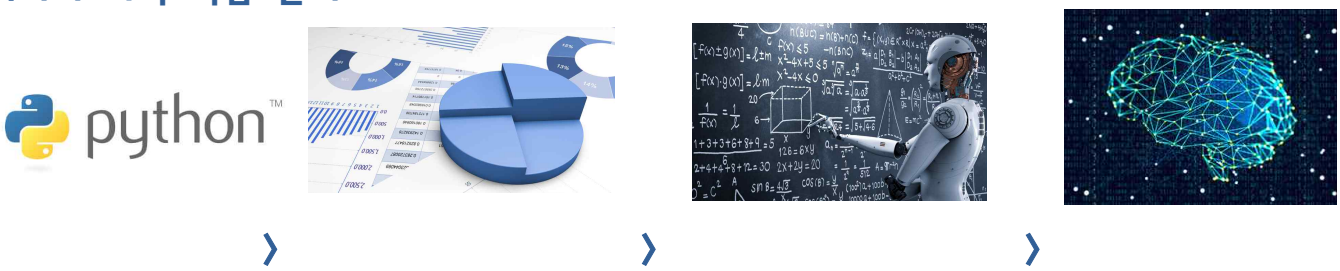
이름 : 권경환

제출일: 2020. 11. 20

【 목 차 】

[AI의 시작]

데이터 과학 학습 순서



파이썬의 장점

-배우기 쉬워서 학습용으로 좋다.
-읽고 쓰기 쉽다.
-간결하고 실사용률이 높다
-확장성이 뛰어나다.

앨런튜링

- 인공지능의 시작은 **앨런튜링**이었으며, **튜링테스트**를 통해 기계가 스스로 저장공간에 저장된 기호들을 읽어들여 처리하고, 그 상태에 따라 다른 상태로 전이가 가능하도록 한다면, 어떠한 연산이던지 스스로 처리 가능하다는 것을 이론적으로 증명한 것이다.

[AI의 역사]

AI의 첫 번째 암흑기 1969~1980

마빈 민스키의 인공신경망의 퍼셉트론에 대한 비판으로 촉발

AI의 두 번째 암흑기 1987~1993

규칙 기반의 전문가시스템의 의구심과 한계

2010년 이후

여러 문제들을 전문가들이 해결하고 새로운 방안을 제시하면서 **최고의 전성기**를 누림

< 인공지능 기술 발전 전망 >

인공지능이란?

- 인간의 경험과 지식을 바탕으로 문제를 해결하는 능력, 시각 및 음성 인식의 지각 능력, 자연 언어 이해능력, 자율적으로 움직이는 능력 등을 컴퓨터나 전자 기술로 실현하는 것을 목적으로 하는 기술 영역을 말함.

인공지능에는 다양한 연구분야가 있는데 머신러닝, 딥러닝, SVM 등이 있다.

머신러닝

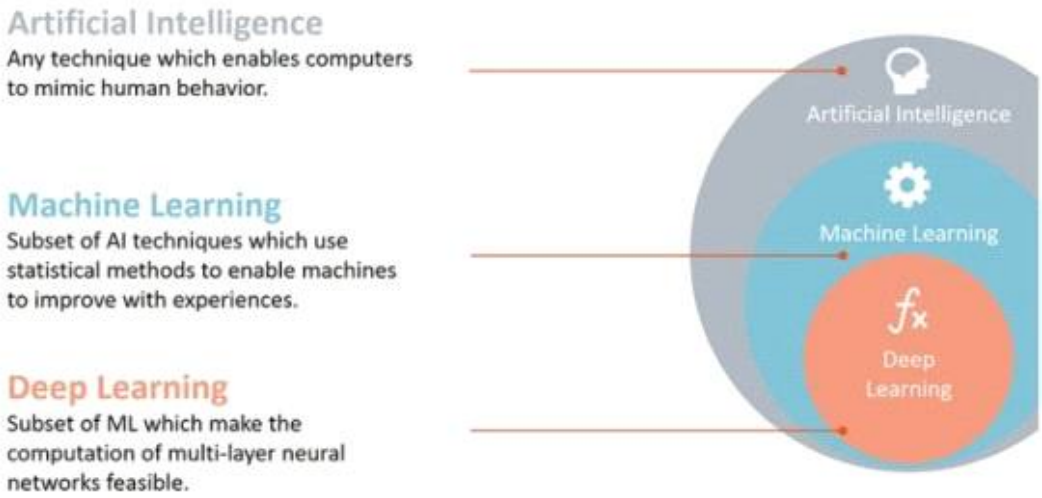
- 인공지능의 연구분야중 하나로 기계가 스스로 학습할 수 있도록 하는 분야이다.

SVM

- 수학적인 방식의 학습 알고리즘

딥러닝

- 다중 계층의 신경망 모델을 사용하는 머신러닝의 일종이다.



인공지능이 가장 큰 범주이고, 머신러닝이 그 다음으로 크며, 딥러닝이 가장 작습니다.

[머신러닝]

많은 데이터가 유입되면 컴퓨터는 더 많이 학습을 하고, 시간이 계속 흐르다보면 작업을 수행하는 능력과 정확도가 향상된다.

머신러닝은 지도학습, 자율학습(비지도학습), 강화학습으로 분류 된다.

지도학습

– 훈련 데이터(Training Data)로부터 하나의 함수를 유추해내기 위한 기계 학습(Machine Learning)의 한 방법이다.

훈련 데이터는 일반적으로 입력 객체에 대한 속성을 벡터 형태로 포함하고 있으며 각각의 벡터에 대해 원하는 결과가 무엇인지 표시되어 있다.

지도학습의 예시

– k-최근접이웃, 선형회귀, 로지스틱 회귀, 서포트 벡터머신, 결정트리, 랜덤 포레스트

비지도학습

– 정답이 없는 훈련데이터를 사용하여 데이터 내에 숨어있는 어떤 관계를 찾아내는 방법

비지도학습의 예시

– 군집, 시각화, 차원축소, 연관규칙학습

강화학습

– 잘한 행동에 대해 보상을 주고 잘못한 행동에 대해 벌을 주는 경험을 통해 지식을 학습하는 방법

강화학습의 예시

– 딥마닝의 알파고, 자동게임분야

머신러닝과 딥러닝의 차이점

	머신러닝	딥러닝
데이터 의존성	중소형 데이터 세트에서 탁월한 성능	큰 데이터 세트에서 뛰어난 성능
하드웨어 의존성	저가형 머신에서 작업하십시오.	GPU가 있는 강력한 기계가 필요합니다. DL은 상당한 양의 행렬 곱셈을 수행합니다.
기능 고학	데이터를 나타내는 기능을 이해해야 함.	데이터를 나타내는 최고의 기능을 이해할 필요가 없습니다.
실행시간	몇 분에서 몇시간	최대 몇 주. 신경망은 상당한 수의 가중치를 계산해야 합니다.

[딥러닝]

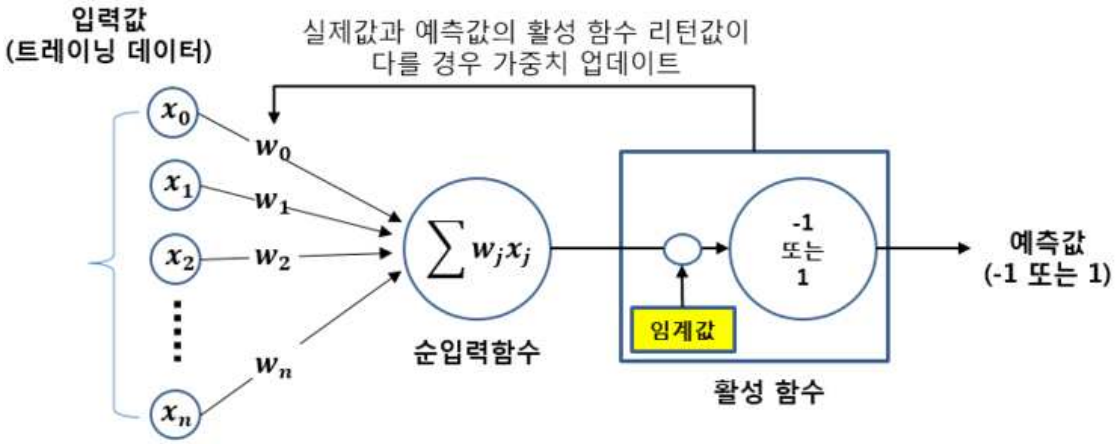
퍼셉트론(인공신경망)

– 세계 최초의 인공신경망을 제안

1957년 코넬대 교수, 심리학자인 프랭크 로젠블랫

– 신경망에서는 방대한 양의 데이터를 신경망으로 유입

데이터를 정확하게 구분하도록 시스템을 학습시켜 원하는 결과를 얻어냄.

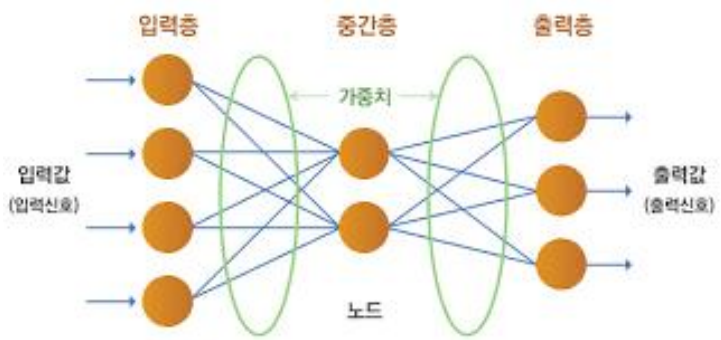


특징

–뇌를 구성하는 신경세포 뉴런의 동작원리에 기초한 기술

인간의 신경세포인 뉴런을 모방하여 만든 가상의 신경을 인공신경망이라고 한다.

항공기, 드론, 자동차의 자율주행, 필체 인식, 음성인식 등 많은 분야에 이용되고 있음



인공신경망의 구조

(1) **입력층과 출력층**

다수의 신호를 입력 받아서 하나의 신호를 출력

(2) **은닉층**

여러개의 층으로 연결하여 하나의 신경망을 구성

딥러닝 구현을 위한 라이브러리

– 텐서플로, 케라스, 파이토치

가장 적합한 언어는 파이썬이다.

Keras: The Python Deep Learning library

케라스는 거의 모든 종류의 딥러닝 모델을 간편하게 만들고 훈련시킬 수 있는 파이썬을 위한 딥러닝 프레임워크입니다.

케라스의 특징

●동일한 코드로 CPU와 GPU에서 실행할 수 있습니다.

●사용하기 쉬운 API를 가지고 있어 딥러닝 모델의 프로토타입을 빠르게 만들 수 있습니다.

●(컴퓨터 비전을 위한) 합성곱 신경망, (시퀀스 처리를 위한) 순환 신경망을 지원하며 이 둘을 자유롭게 조합하여 사용할 수 있습니다.

●다중 입력이나 다중 출력 모델, 층의 공유, 모델 공유 등 어떤 네트워크 구조도 만들 수 있으며, 케라스는 기본적으로 어떤 딥러닝 모델에도 적합하다는 뜻입니다.

텐서플로는 구글에서 만든, 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공해주는 라이브러리다.

텐서플로의 특징

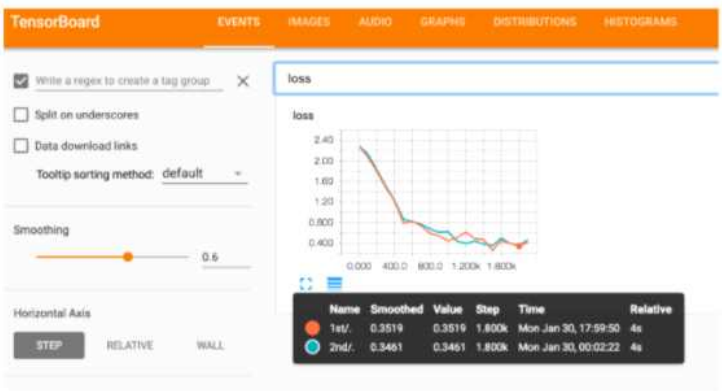
●데이터 플로우 그래프를 통한 풍부한 표현력

●코드 수정 없이 CPU/GPU 모드로 동작

●아이디어 테스트에서 서비스 단계까지 이용 가능

●계산 구조와 목표 함수만 정의하면 자동으로 미분 계산을 처리

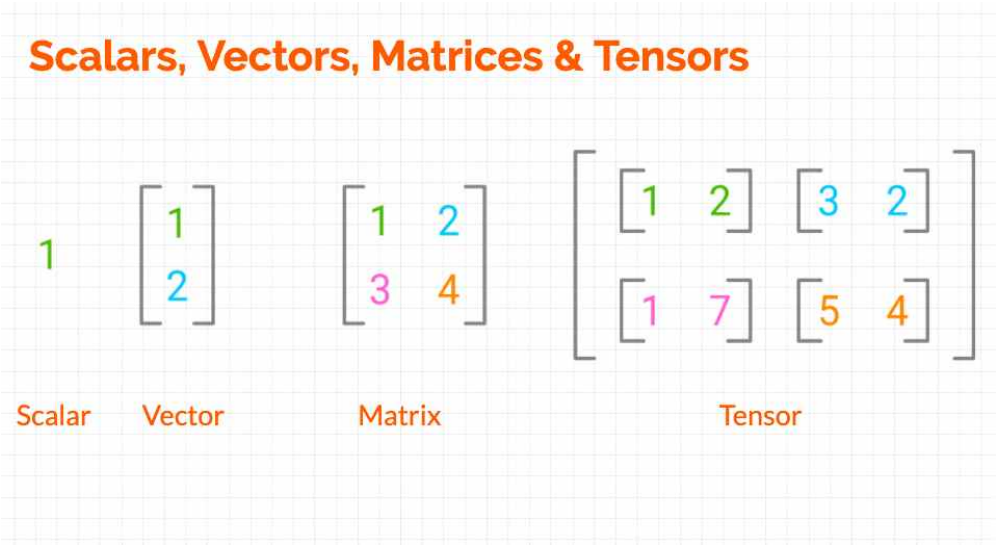
●Python/C++를 지원하며, SWIG를 통해 다양한 언어 지원 가능



●브라우저에서 실행 가능한 시각화 도우미 텐서보드가 있음

[텐서]

텐서는 모든데이터를 뜻하며, 딥러닝에서 데이터를 표현하는 방식이다.



스칼라: 차원이 없는 텐서

벡터: 1차원 텐서

2차원 행렬:2차원 텐서

- 회색조 이미지: 하나의 채널에 2차원 행렬로 표현

텐서: n차원 행렬

- 텐서의 차원을 텐서라 함
- RGB 이미지: R,G,B 각 3개의 채널마다 2차원 행렬로 표현하는데, 이를 텐서로 표현

Tensorflow 계산 과정

- 모두 그래프라고 부르는 객체 내에 저장되어 실행
- 그래프를 계산하려면 외부 컴퓨터에 이 그래프 정보를 전달하고 그 결과값을 받아와야 함

Session

- 통신과정을 담당하는 것이 세션이라고 부르는 객체이며, 선언, 실행, 종료 과정이필요

세션 선언문	① sess = tf.Session() ② sess = tf.InteractiveSession()
세션 실행문	sess.run(*텐서 or 연산*)
세션 종료문	sess.close() (단, with 절에서는 불필요)

데이터 흐름 그래프로 이루어짐

텐서 형태의 데이터들이 딥러닝 모델을 구성하는 연산들의 그래프를 따라 흐르면서 연산이 일어남.

Tensor + DataFlow

딥러닝에서 데이터를 의미하는 텐서와 데이터그래프를 따라 연산이 수행되는 형태를 합쳐

텐서플로란 이름이 나오게 됨

[텐서의 기본문법]

구글의 Colab

- 파이썬과 머신러닝, 딥러닝 개발 클라우드 서비스

Colab 특징

- 구글 드라이브를 기본 저장소로 사용
- 클라우드 기반의 무료 Jupyter 노트북 개발환경
- 구글 계정 전용의 가상머신 지원(GPU, TPU)
- 깃허브, 구글드라이브와 연계

```
[ ]  import tensorflow as tf

     hello = tf.constant('Hello World!')

     sess = tf.Session()
     print(sess.run(hello))
     sess.close()
```

```
b'Hello World!'
```

```
[ ]  a = 10
     b = 10
     print(tf.add(a, b))

     sess=tf.Session()
     print(sess.run(tf.add(a, b)))
     sess.close()
```

```
Tensor("Add_3:0", shape=(), dtype=int32)
20
```

[코랩에서 1.x을 사용중이라면 이후 그대로 import]

```
a = tf.constant(3)
a
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=3>
```

```
[ ]  a.shape
```

```
TensorShape([])
```

```
[ ]  a.numpy()
```

```
3
```

텐서 출력 메소드 numpy()

조건 연사 tf.cond()

```
[ ]  x = tf.constant(1.)
     bool = tf.constant(True)
     res = tf.cond(bool, lambda: tf.add(x, 1.), lambda: tf.add(x, 10.))

     print(res)
     print(res.numpy())
```

```
tf.Tensor(2.0, shape=(), dtype=float32)
2.0
```

-pred를 검사해 참이면
true_fc 반환

-pred를 검사해 거짓이면
false_fc 반환

```
[ ]  x = tf.constant(2)
     y = tf.constant(5)
     def f1(): return tf.multiply(x, 17)
     def f2(): return tf.add(y, 23)
     r = tf.cond(tf.less(x, y), f1, f2)

     r.numpy()
```

```
34
```

[텐서의 브로드캐스팅]

Broadcast의 사전적인 의미는 '퍼뜨리다'라는 뜻이 있는데, 이와 마찬가지로 두 행렬 A, B 중 크기가 작은 행렬을 크기가 큰 행렬과 모양(shape)이 맞게끔 늘려주는 것을 의미한다.

```
[ ]  x = tf.constant([[0], [10], [20], [30]])
     y = tf.constant([0, 1, 2])

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

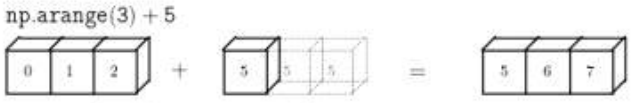
```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

np.arange나 다른 관련된 함수를 사용하기 위해서는 numpy를 import 해줘야한다.
np.arange(3)은 3 이전의 수 0,1,2를 가져옴

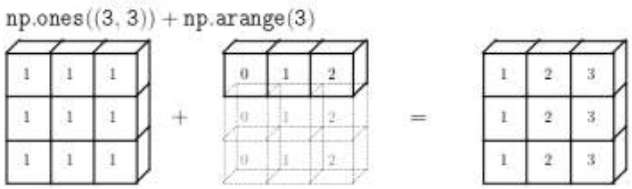
```
[ ]  import numpy as np

     print(np.arange(3))
     print(np.ones((3, 3)))
     print()

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

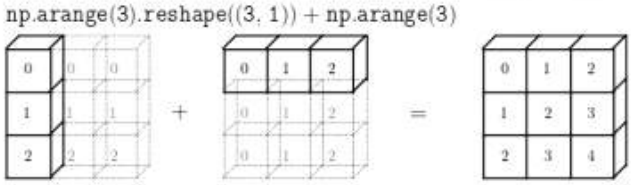
```
[0 1 2]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

tf.Tensor([0 1 2], shape=(3,), dtype=int64)
tf.Tensor([5], shape=(1,), dtype=int64)
tf.Tensor([5 6 7], shape=(3,), dtype=int64)
```

np.ones는 3x3 배열을 1로 채우는 함수이다.



x+y는 이런식으로 더 해져서 [5,6,7] 값이 나온다.





이외에도 위와 같은 방식으로 더 해져서 값이 나오게 된다.

```
[ ]  import numpy as np

     print(np.arange(3))
     print(np.ones((3, 3)))
     print()

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

```
[0 1 2]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

tf.Tensor([0 1 2], shape=(3,), dtype=int64)
tf.Tensor([5], shape=(1,), dtype=int64)
tf.Tensor([5 6 7], shape=(3,), dtype=int64)
```

```
[ ]  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]]
```

행렬의 곱셈은 앞의 행렬에서, 뒤의 행렬의 열을 순차적으로 곱해준다.
아래 그림을 보면 쉽게 이해할 수 있다.


```
[ ]  # 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.]]
```

곱셉의 사용되는 명령어는 **tf.matmul()**이다
2차원 행렬 곱셈이며. 위에 그림처럼 적용하면
1*2 + 2*1 = 4
3*2 + 4*1 = 10
1*0 + 2*2 = 4
3*0 + 4*2 = 8이 되어
[[4 4][10 8]] 이 나오게 된다.

행렬의 차수변환은 tf.rank() 명령어를 사용하며, 차원의 수를 반환하는 것이다.
0은 **스칼라**, 1은 **벡터**. 2는 **행렬**. 3은 **3-텐서**, 4는 **n-텐서**이다.

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

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

```
[ ]  tf.rank(my_image)
```

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

```
[ ]  tf.rank(my_image).numpy()
```

```
4
```

tf.zeros는 shape에 지정한 형태의 텐서를 만들고,
모든 원소의 값을 0으로 **초기화** 하는것이다
행렬의 모양은 **4차**이고 **원소수**는 2*5*5*3한 값이다.

shape와 reshape라는 명령어로 지정한 형태의 텐서를 만들거나 변경할 수 있다.
tf.ones는 원소의 값을 모두 1로 하는 명령어이다.

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

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

```
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)
```

tf.ones([3,4,5])는
원소의 값을 1로한 4*5개의 원소의 수를 가진 행렬을 3개 만들어라 라는 뜻이다.

```
# 기존 내용을 6x10 행렬로 형태 변경
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)>
```

기존에있는 rank_three_tensor를 tf.reshape로 6*10 행렬로 형태 변경한 것이다.

```
# 기존 내용을 3x20 행렬로 형태 변경
# -1은 차원 크기를 계산하여 자동으로 결정하라는 의미
matrixB = tf.reshape(matrix, [3, -1])
matrixB
```

```
<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)>
```

tf.reshape(matrix, [3. -1])의 -1은 차원 크기를 자동으로 계산하여 결정하라는 뜻이다.
전에 6*10 행렬이기에 60개의 원소의 수였다. 그렇기 때문에 자동으로 3*20이 된 것이다.

자료형 변환

tf.Tensor의 자료형을 다른 것으로 변경해주는 명령어이다.

```
[ ]  # 정수형 텐서를 실수형으로 변환
     float_tensor = tf.cast(tf.constant([1, 2, 3]), dtype=tf.float32)
     float_tensor
```

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

```
[ ]  float_tensor.dtype
```

```
tf.float32
```

dtype=tf.float로 변경하였기에 정수형에서 실수형으로 변환된 것을 확인할 수 있다.

변수Variable

변수형 Variable은 객체로 생성이 된다.

텐서플로 그래프에서 tf.variable의 값을 사용하려면 이를 단순히 tf.Tensor로 취급

```
[ ]  v = tf.Variable(0.0)
     v
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=0.0>
```

```
[ ]  w = v + 10
     w
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=10.0>
```

```
[ ]  w.numpy()
```

```
10.0
```

```
[ ]  v = tf.Variable(2.0)
     v.assign_add(5)
     v
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=7.0>
```

```
[ ]  v.read_value()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=7.0>
```

메소드 assign, assign_add

-값을 변수에 할당하는 것

메소드 read_value

-현재 변수값 읽기

[텐서플로 난수]

균등분포 난수

tf.random.uniform([1],0,1) –› [1]은 배열,0은 시작,1은 끝이다.

```
#3.7 랜덤한 수 얻기(균일 분포)
rand = tf.random.uniform([1], 0, 1)
print(rand)
```

```
tf.Tensor([0.7896708], shape=(1,), dtype=float32)
```

– 0~1까지의 수로 만들라는 뜻이다.

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

```
tf.Tensor(
[[0.04188013 0.11457467 0.21168363 0.68798304]
 [0.89811087 0.42754436 0.6907078  0.858446  ]
 [0.92062104 0.6143166  0.3572985  0.14260638]
 [0.07380903 0.49818766 0.01119924 0.10025978]
 [0.5582659  0.8529248  0.07200933 0.204229  ]], shape=(5, 4), dtype=float32)
```

– 5x4형태로 0~1까지의 수로 만들라는 뜻이다.

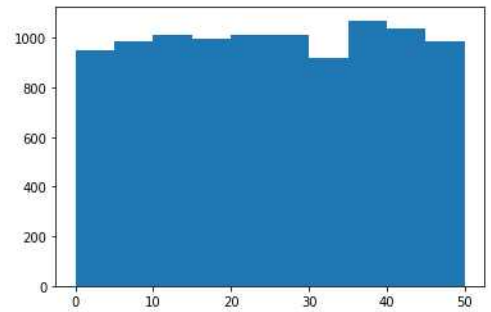
```
[ ]  rand = tf.random.uniform([1000], 0, 10)
     print(rand[:10])
```

```
tf.Tensor(
[3.5438478 2.3234856 0.4235363 1.1751997 0.9174347 2.7804935 7.1180067
 1.5477157 3.3756459 3.6877894], shape=(10,), dtype=float32)
```

–1000개를 0~10까지의 수로 만들되, 10개만 출력하라는 뜻이며, rand[:10]의 :는 슬라이싱이라고 한다.

```
[ ]  import matplotlib.pyplot as plt
     rand = tf.random.uniform([10000], 0, 50)
     plt.hist(rand, bins=10)
```

```
(array([ 951.,  985., 1012.,  999., 1015., 1015.,  922., 1071., 1041.,
         989.]),
 array([8.7618828e-04, 5.0001888e+00, 9.9995012e+00, 1.4998815e+01,
        1.9998127e+01, 2.4997440e+01, 2.9996752e+01, 3.4996067e+01,
        3.9995377e+01, 4.4994690e+01, 4.9994003e+01], dtype=float32),
 <a list of 10 Patch objects>)
```



그래프화하기 위해서는 matplotlib.pyplot을 import 해야한다.
–10000개를 0~50까지의 수로 만들고 plt.hist 명령어로 시각화해서 보여주라는 뜻이며, bins는 굴곡을 의미함.

정규분포 난수

tf.random.normal([4],0,1) -> [4]는 크기, 0은 평균, 1은 표준편차 이다.

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

```
tf.Tensor([ 0.60418266 -0.7893311   1.157336    0.6584586 ], shape=(4,), dtype=float32)
```

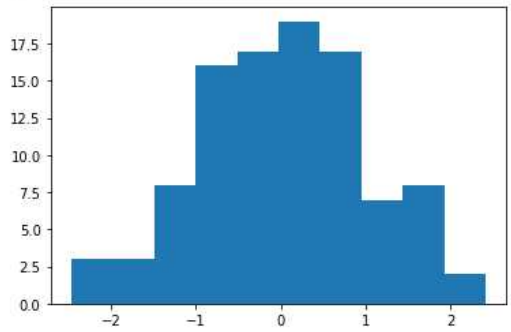
```
[ ]  # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
     rand = tf.random.normal([2, 4],0 ,2)
     print(rand)
```

```
tf.Tensor(
[[-2.124051   0.9770188 -3.501095  -4.1315928]
 [-1.4304713 -1.3903373  2.9812193 -1.6362426]], shape=(2, 4), dtype=float32)
```

2x4형태로 0의 가까운 값이며 2 정도의 표춘편차를 가진 값을 만들라는 뜻이다.

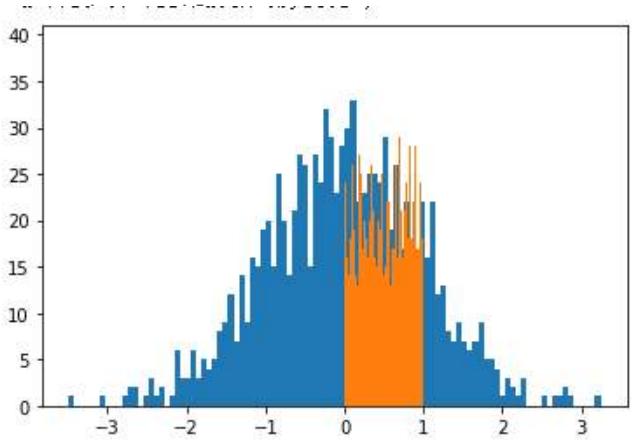
```
import matplotlib.pyplot as plt
rand = tf.random.normal([100], 0, 1)
plt.hist(rand, bins=10)
```

```
(array([ 3.,  3.,  8., 16., 17., 19., 17.,  7.,  8.,  2.]),
 array([-2.471905  , -1.9830686 , -1.4942322 , -1.0053957 , -0.51655924,
        -0.02772284,  0.4611136 ,  0.94995004,  1.4387865 ,  1.9276229 ,
         2.4164593 ], dtype=float32),
 <a list of 10 Patch objects>)
```



정규분포 100개 그리기이며,
여기서도 matplotlib.pyplot를 import 함.
정규분포는 균등분포처럼 완만한 모양과 달리 굴곡형 모양을 지님.

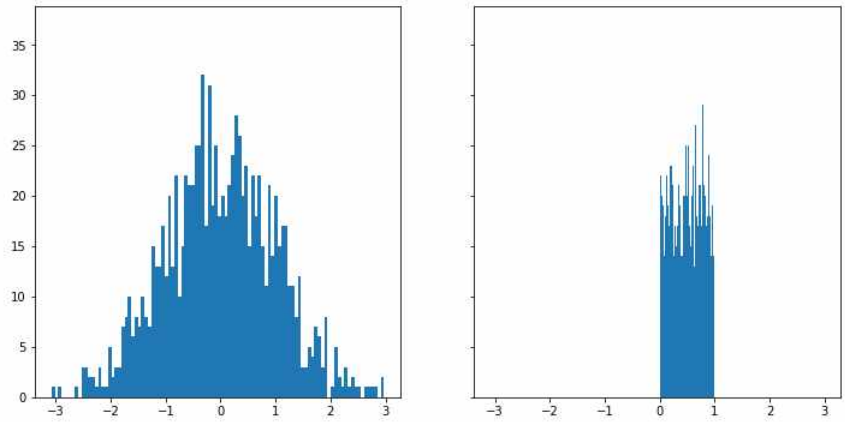
```
import matplotlib.pyplot as plt
rand1 = tf.random.normal([1000], 0, 1)
rand2 = tf.random.uniform([2000], 0, 1)
plt.hist(rand1, bins = 100)
plt.hist(rand2, bins = 100)
```



균등분포와 정규분포의 비교를 위해 2개의 대치되는 그래프를 만드는 소스이며, 아래사진이 나타나는 모양이다.

```
[ ]  import matplotlib.pyplot as plt
     rand1 = tf.random.normal([1000], 0, 1)
     rand2 = tf.random.uniform([2000], 0, 1)

     plt.rcParams["figure.figsize"] = (12,6)
     fig, axes = plt.subplots(1, 2, sharex=True, sharey=True)
     axes[0].hist(rand1, bins=100)
     axes[1].hist(rand2, bins=100)
```

그래프를 따로 그릴 수도 있으며,
사이즈를 12*6의 크기로 그래프 틀을 만들고
1*2형태로 그래프를 나타내며,
sharex,y와는 x와 y축의 눈금을
표시하는 것이다.



tf.random.shuffle(a)

```
import numpy as np
a = np.arange(10)
print(a)
tf.random.shuffle(a)
```

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

-arange로 10까지의 수를 나열하며, a 변수에 저장하며 shuffle을 한다.

```
[ ]  import numpy as np
     a = np.arange(20).reshape(4, 5)
     a
```

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

- 20까지의 수로 나열하며 4*5형태로 형태를 변경한다음에 출력하라는 뜻이다.

```
[ ]  tf.random.shuffle(a, )
```

```
<tf.Tensor: shape=(4, 5), dtype=int64, numpy=
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9]])>
```

-셔플을 하고 출력해서 나타난 모습

[MNIST 데이터셋]

딥러닝 손글씨 인식에 사용되는 데이터셋을 MNIST 데이터셋이라고 한다.

-필기숫자 이미지와 정답인 레이블의 쌍으로 구성.

-일반적으로 2차원 행렬로 구성되어있으며, 28x28인 784픽셀이다.

-내부값은 0~255이며 이 값들은 수정이 가능하다.

-Yann Lecun의 웹 사이트에서 배포함.

-Label: 필기 숫자 이미지가 나타내는 실제숫자 0~9

데이터를 가져오는 방법: tensorflow.keras.datasets.mnist

MNIST데이터 훈련 데이터 구조

-총 6만개의 손 글씨가 있다.

훈련 데이터 손글씨와 정답

- x_train, y_train : 6만개

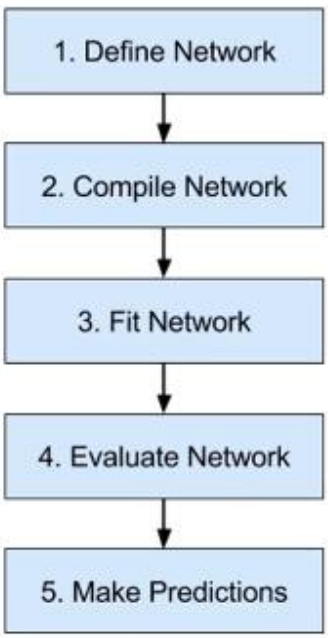
테스트 데이터 손글씨와 정답

- x_test, y_test : 1만개

```
import tensorflow as tf

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

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

케라스 딥러닝 구현 과정



- 딥러닝 모델을 만들어

- 주요 훈련방법을 설정하고
(최적화방법, 손실 함수, 훈련 모니터용 지표)

- 훈련시켜

-테스트 데이터를 평가하고

- 정답을 예측

첫 손글씨와 데이터와 정답

x_train[0], y_train[0]

```
[ ]  x_train.shape

     (60000, 28, 28)
```

– 훈련데이터의 형태를 보여달라는 명령어이다.
6만개이며 28x28픽셀이다.

```
[ ]  y_train.shape

     (60000,)
```

– 훈련데이터의 정답의 개수 형태를 보여달라는 명령어이다.

```
x_train[0]
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253,
        253, 205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1,
        154, 253,  90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        139, 253, 190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0],
```

x_train의 첫 번째를 출력하는것이며
28x28픽셀로 보여준다.

```
y_train[0]

5
```

–첫번째 훈련데이터의 정답은 5이다.

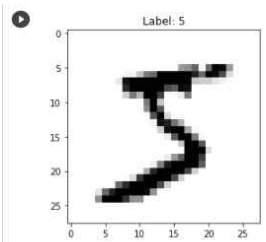
```
[ ]  import sys

     for x in x_train[0]:
         for i in x:
             sys.stdout.write('%3d' %i)
         sys.stdout.write('₩n')
```

```
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  3 18 18 18126136175 26166255247127  0  0  0  0
  0  0  0  0  0  0  0  0  0 30 36 94154170253253253253253225172253242195 64  0  0  0  0
  0  0  0  0  0  0  0 49238253253253253253253253253251 93 82 82 56 39  0  0  0  0  0
  0  0  0  0  0  0  0 18219253253253253253198182247241  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0 80156107253253205 11  0 43154  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0 14  1154253 90  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0 0139253190  2  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0 11190253 70  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0 35241225160108  1  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0 81240253253119 25  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 45186253253150 27  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 16 93252253187  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0249253249 64  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 46130183253253207  2  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 39148229253253253250182  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 24114221253253253253201 78  0  0  0  0  0
  0  0  0  0  0  0  0  0  0 23 66213253253253253198 81  2  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0 18171219253253253253195 80  9  0  0  0  0  0  0  0  0  0
  0  0  0  0 55172226253253253253244133 11  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0 13625325325325321213513216  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

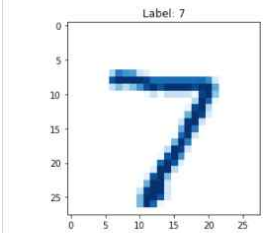
–이미지 28x28에 784개의
정수값(0~255)으로 출력함.

–x_train의 첫 번째 정답이 5였는데
실제로 어떻게 출력이 되는지
확인하는 방법이다.

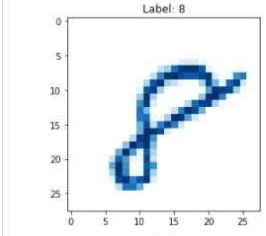
```
##############################################
# MNIST 데이터(훈련, 테스트)의 내부 첫 내용을 그려보자.
import matplotlib.pyplot as plt

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 ='Blues')
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()
##############################################
```



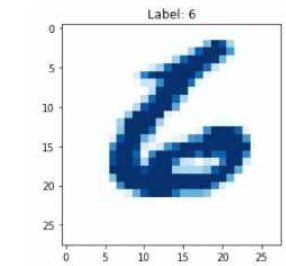
- tmp에 훈련데이터 첫 번째의 정답을 저장하고 타이틀로 표시하며, imshow로 이미지를 표시하기위해 x_train[0]값을 불러옴.



- 두 번째는 테스트데이터 첫 번째정답을 저장하고 타이틀로표시하며, x_test값을 이용함.



- idx에 x_train의 마지막값을 저장함.
  x_train[idx]로 값을 표시함.



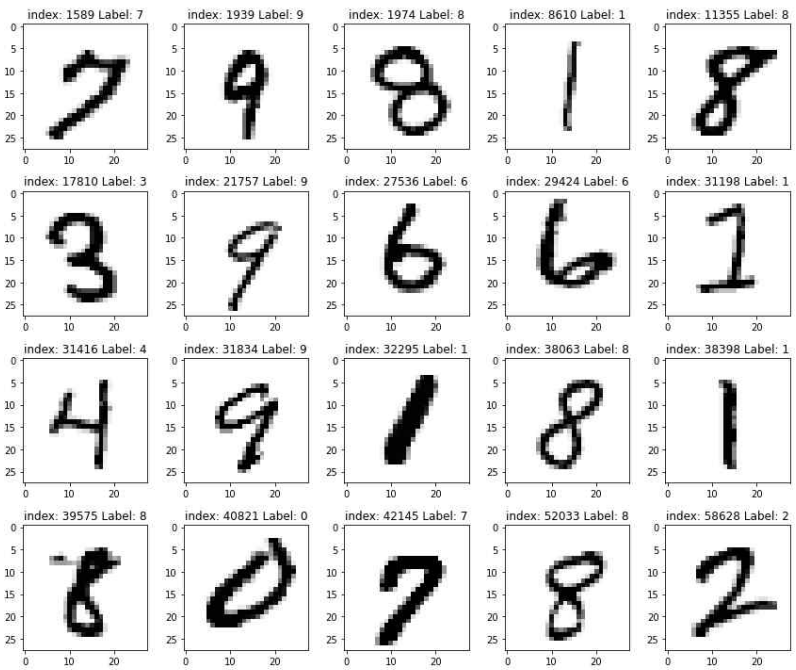
- idx에 x_train의 마지막값을 저장함.
다음은 위와같은 방식으로 x_test[idx]로 표시함.

훈련용 데이터 6만개중에서 임의 수 20개 선택하는 출력문이다.
0~59999 중의 선택할 번호 선정

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

     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)
```

```
[8832, 9347, 12123, 13094, 16527, 20044, 21432, 26613, 30417, 32225, 33550,
```

```
################################
# 랜덤하게 20개의 훈련용 자료를 그려 보자.
from random import sample

nrows, ncols = 4, 5 #출력 가로 세로 수
idx = sorted(sample(range(len(x_train)), nrows * ncols)) # 출력할 첨자 선정
#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()
#################################
```



[딥러닝 과정]

[주요 용어]

데이터셋

-훈련용과 테스트용

Train data set, Test data set

x(입력, 문제), y(정답, 레이블), 전처리

모델

- 딥러닝 핵심 신경망, 여러 층 구성

완전 연결층(Dense), 1차원 배열로 평탄화(Flatten())

학습 방법의 여러 요소들

- 옵티마이저, 최적화 방법

경사하강법

-손실함수

Cross entropy, MSE

딥러닝 훈련

-Epochs

총 훈련횟수

[딥러닝 과정 세부설명]

① 훈련과 정답 데이터 지정

MNIST 데이터셋을 로드하여 준비

-전처리(샘플값을 정수에서 부동소수로 변환하는 것을 말함, 픽셀값은 0에서 1사이의 값으로 표현됨)

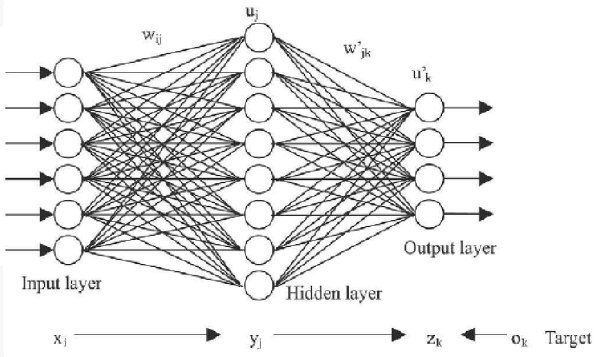
```
##################################################
import tensorflow as tf

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
```

② 모델 구성

층을 차례대로 쌓아 모델을 생성

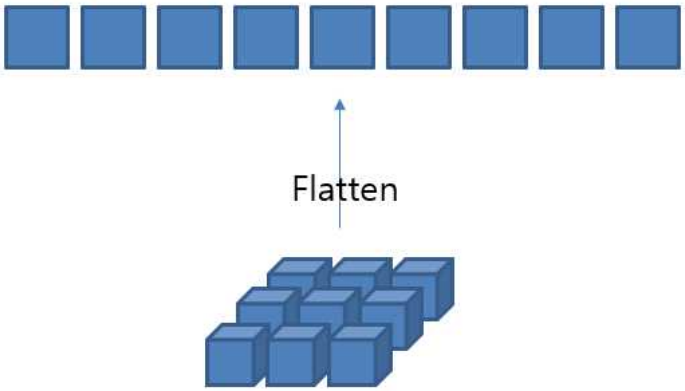
- 신경망 구성

입력층, 중간층, 출력층




```
[ ]  # 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(Flatten): 2차원/3차원의 Matrix/Tensor 구조를 1차원의 Vector로 변환하는 과정



단순신경망 모델과 Dense층

- 중간 은닉층이 없는 구조(입력층과 출력층만 존재)

Dense()

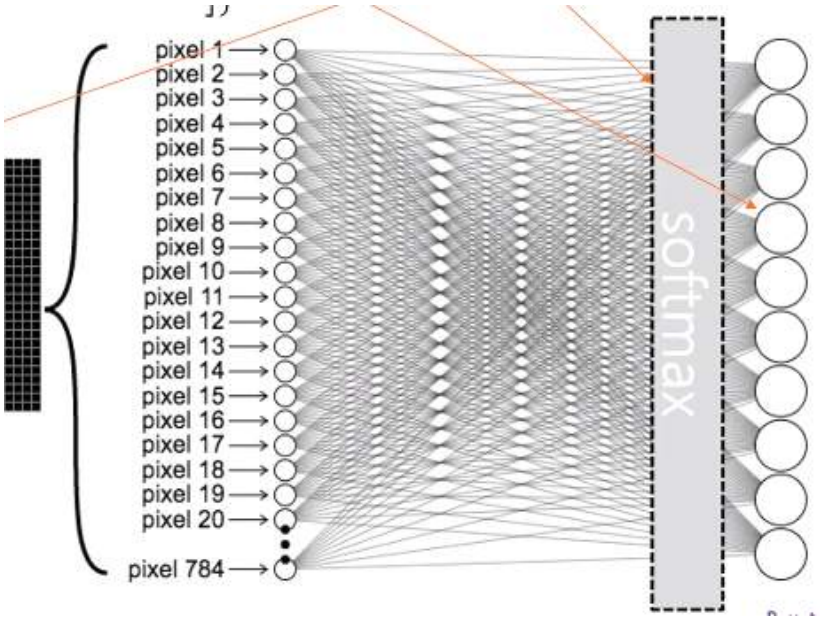
- 완전연결층

입력: 784

-이미지의 각 픽셀 값

출력: 10

- 각 위치의 값이 될 크기



드롭아웃: tf.keras.layers.Dropout(0.2)

-훈련 중에 20%를 중간에 끊음(예측할 EO는 모두 사용)

랜덤하게 뉴런을 끊음으로써 모델을 좀 더 단순하게 만든다.

활성화 함수: activation function

ReLU

-우선 가장 많이 사용되는 함수는 ReLU이다.

간단하고 사용이 쉽기 때문에 우선적으로 ReLU를 사용한다.

Softmax

- Softmax(소프트맥스)는 입력받은 값을 출력으로 0~1사이의 값으로 모두 정규화하며 출력 값들의 총합은 항상 1이 되는 특성을 가진 함수이다.

③ 학습에 필요한 최적화 방법과 손실함수 등 설정

훈련에 사용할 옵티마이저와 손실함수등을 선택

옵티마이저

-입력된 데이터와 손실 함수를 기반으로 모델을 업데이트하는 메커니즘이다.

손실함수

- 훈련 데이터에서 신경망의 성능을 측정하는 방법

훈련과 테스트 과정을 모니터링할 지표

- 여기에서는 정확도만 고려

```
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 모델에 선정
model.compile(optimizer='adam',
              loss='sparse_catrgorical_crossentropy',
              metrics=['accuracy'])
              # metrics=['accuracy', 'mse'])

# 모델 요약 표시
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

모델요약(model.summary())

- compile전에도 summary()가능

각 층의 구조와 패라미터수 표시

-가중치와 편향

총 패라미터수

- 모델이 구해야 할 수의 개수

- 100480 + 1290 = 101770개

④ 모델을 훈련

model.fit()

-훈련 횟수 epochs에 지정, 훈련 데이터에 모델을 학습(모델의 매개변수를 정하는 과정)

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

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0669 - accuracy: 0.9787
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0583 - accuracy: 0.9812
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0557 - accuracy: 0.9818
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0501 - accuracy: 0.9834
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0454 - accuracy: 0.9846
<tensorflow.python.keras.callbacks.History at 0x7f9904203b00>
```

⑤ 테스트 데이터로 성능 평가

모델을 평가

- 테스트 세트에서도 모델이 잘 작동하는지 확인

- model.evaluate()

손실값과, 예측 정확도 반환(loss, accuracy)

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

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0722 - accuracy: 0.9769
[0.07219623774290085, 0.9768999814987183]
```

딥러닝 과정 전소스

```
##################################################
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

# 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')
])

# 모델 요약 표시
model.summary()

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

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

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

[데이터 예측]

[예측결과 확인]

model.predict(input)

- input값

모델의 fit(), evaluate()에 입력과 같은 형태가 필요

28x28 이미지가 여러 개인 3차원

- 슬라이스해서 사용

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

결과

-정수

손글씨 값의 정수

-실제

(1,10)의 이차원 배열

-결과

10개의 0~1의 실수

```
1 # 테스트 데이터의 첫 번째 손글씨 예측 결과를 확인
2 print(x_test[:1].shape)
3
4 pred_result = model.predict(x_test[:1])
5 print(pred_result.shape)
6 print(pred_result)
7 print(pred_result[0])
```

```
(1, 28, 28)
(1, 10)
[[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]]
[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]
```

10개의 실수는 확률값이며, 10개 합이 1이다.

argmax()로 가장 큰수의 위치 첨자를 반환할 수 있다.

```
import numpy as np
# 10개의 수를 더하면?
one_pred = pred_result[0]
print(one_pred.sum())

#혹시 가장 큰 수가 있는 첨자가 결과
one = np.argmax(one_pred)
print(one)
```

One Hot Encoding

- 데이터가 취할 수 있는 모든 단일 범주에 대해 하나의 새 열을 생성하는 것
- 모든행에서 범주에 속하는 경우 1을, 그렇지 않으면 0을 배치함.

● 메소드 np.argmax()

```
-----------------------
import numpy as np
# 원핫 인코딩과 argmax학습
print(np.argmax([5, 4, 10, 1, 2]))
print(np.argmax([3, 1, 4, 9, 6, 7,2]))
print(np.argmax([0.1, 0.8, 0.1],[0.7, 0.2, 0.1],[0.2, 0.1, 0.7]], axis=1))
-----------------------
2                 - [5, 4, 10, 1, 2] 5부터 0,1,2 순서기 때문에 10이 크기에 2로 출력
3                 - [3, 1, 4, 9, 6, 7,2] 9이기에 3 출력
[1 0 2]           - 0.8이기에 1, 0.7이기에 0, 0.7이기에 2가 출력된다.
-----------------------
```

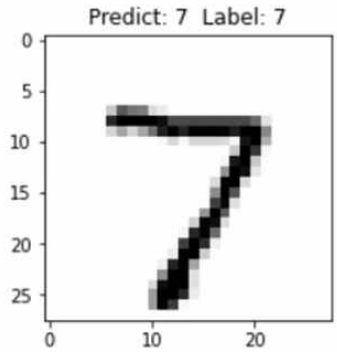
● 메소드 tf.argmax()

```
-----------------------
import numpy as np
# 원핫 인코딩과 argmax학습
print(np.argmax([5, 4, 10, 1, 2]))
print(np.argmax([3, 1, 4, 9, 6, 7,2]))
print(np.argmax([0.1, 0.8, 0.1],[0.7, 0.2, 0.1],[0.2, 0.1, 0.7]], axis=1))
-----------------------
tf.Tensor(2, shape=(), dtype=int64)    - 텐서로 출력되는것이고, 결과는 같음
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor([1,0,2], shape=(3.), dtype=int64)
-----------------------
```

● 예측(predict())

```
-----------------------
import numpy as np
# 10개의 수를 더하면?
one_pred = pred_result[0]
print(one_pred.sum())

#혹시 가장 큰 수가 있는 첨자가 결과
one = np.argmax(one_pred)
print(one)
#시각화하기 위해 plt 호출
import matplotlib.pyplot as plt
plt.figure(figsize=(5, 3))
tmp = "Predict: " + str(one) + " Label: " + str(y_test[0])
plt.title(tmp)
plt.imshow(x_test[0], cmap='Greys')
-----------------------
```

```
1.0
7
<matplotlib.image.AxesImage at 0x7f4a7890f2b0>
```



● **테스트 데이터 모두 예측해보기**

```
from random import sample
import numpy as np

# x_test로 직접 결과 처리
pred_result = model.predict(x_test)
print(pred_result.shape)
print(pred_result[0])
print(np.argmax(pred_result[0]))

# 원핫 인코딩을 일반 데이터로 변환
pred_labels = np.argmax(pred_result, axis=1)
# 예측한 답 출력
print(pred_labels)
# 실제 정답 출력
print(y_test)
#####################################################
```

```
(10000, 10)
[3.1434331e-07 2.5280498e-08 6.2301833e-06 9.8214645e-05 3.8718386e-11
6.4800524e-08 7.3486254e-14 9.9989331e-01 7.0924173e-08 1.7541108e-06]
7
[7 2 1 ... 4 5 6]
[7 2 1 ... 4 5 6]
```

● **임의의 20개 예측값과 정답**

```
################################################
from random import sample
import numpy as np

# 예측한 softmax의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)

# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)

#랜덤하게 20개의 훈련용 자료를 예측 값과 정답, 그림을 그려 보자.
nrows, ncols = 5, 4  #출력 가로 세로 수
samples = sorted(sample(range(len(x_test)), nrows * ncols)) # 출력할 첨자 선정
```

pred_result

– 모델의 예측결과, 확률값

pred_labels

– 모델의 예측결과, 정수

samples

– 출력할 20개의 첨자 리스트

● 예측이 틀린 것은 'Blues'로 그리기

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

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

pred_labels[n]==y_test[n]

– 예측이 맞는 경우

리스트 pred_labels

–모델의 예측결과, 정수

리스트 y_test

– 훈련 데이터 정답

예측이 틀린 것은 'Blues'로 그리기

● 예측이 틀린 20개 찾기

```
from random import sample
import numpy as np

##################################################
# 예측 틀린 것 첨자를 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)
```

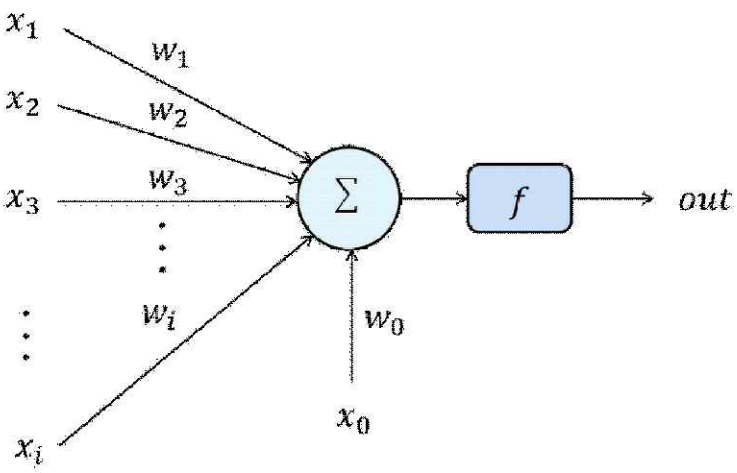
```
정답이 틀린 수 195
[6625, 6093, 3946, 5676, 9587, 8311, 3520, 9679, 3558, 4571, 2953, 1112,
3503, 5642, 2369, 6400, 4551, 1247, 3943, 5734]
```

정확도를 높이려면 중간층을 늘리고 훈련 횟수를 증가하면 된다.
-128개 뉴런, 64개 뉴런, 10개출력, 훈련횟수 20회
훈련횟수 20 = epochs=20

```
# 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(64, 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=20)
```

[인공신경망]

인공신경세포에는 뉴런과, 신경망이 있다.



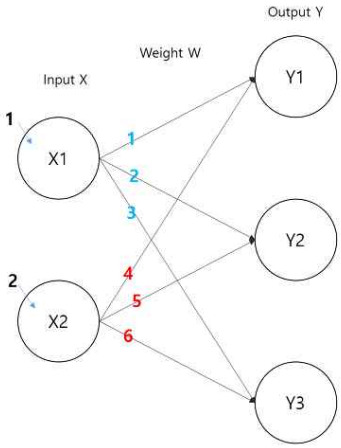
뉴런

- 입력, 편향

신경망

- 뉴런의 연결

뉴런의 행렬연산의 예



X * W = Y

1 x 2 * 2 x 3 = 1 x 3

$(1 \quad 2) \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad (9 \quad 12 \quad 15)$

```
[14] x = [[1, 2]]
     w = [[1, 2, 3], [4, 5, 6]]

     y = tf.matmul(x, w)
     y.numpy()

     array([[ 9, 12, 15]], dtype=int32)
```

© sacko

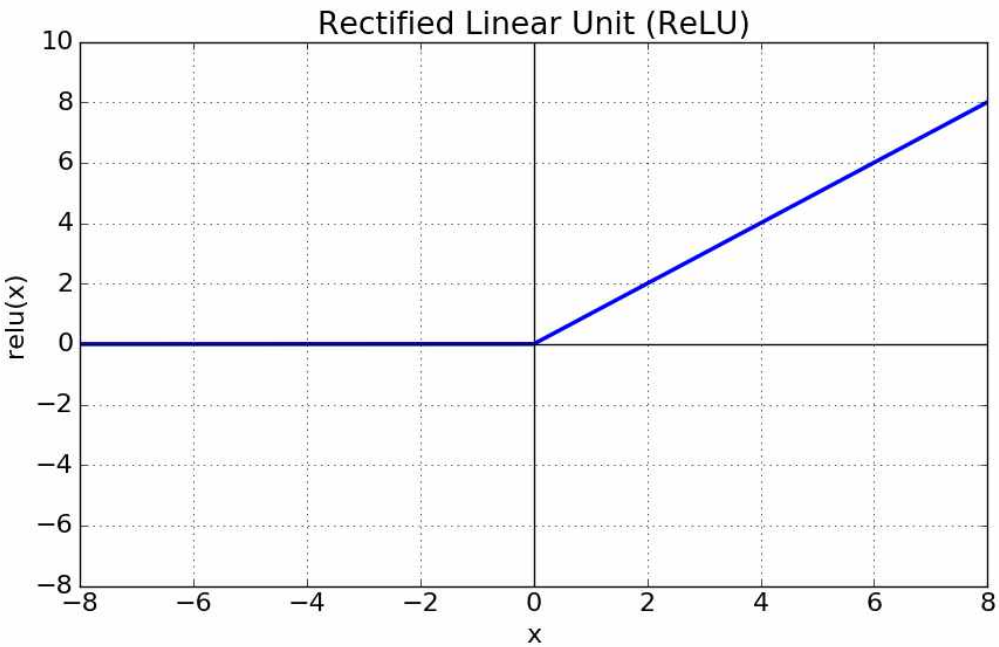
활성화 함수

- 뉴런의 출력값을 정하는 함수

ReLU

-x가 0보다 크면 기울기가 1인 직선, 0보다 작으면 함수 값이 0이 된다.

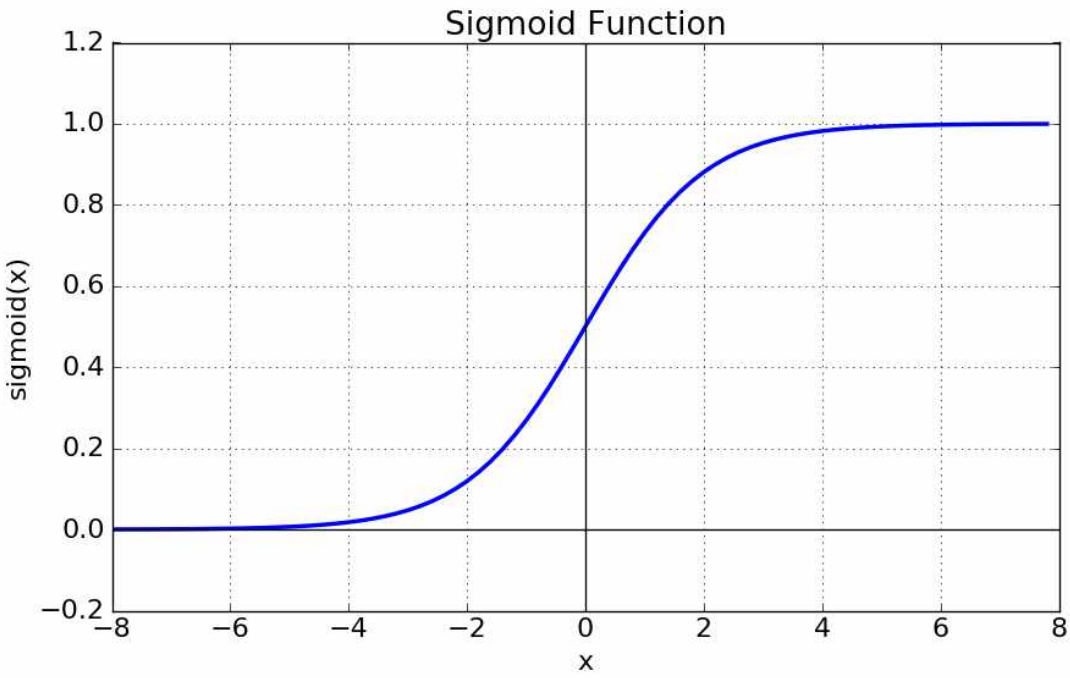
-sigmoid, tanh 함수보다 학습이 빠르고, 연산 비용이 적고, 구현이 매우 간단하다는 특징이 있다.



sigmoid

-시그모이드 함수는 Logistic 함수라고 불리기도 하며, x의 값에 따라 0~1의 값을 출력하는 S자형 함수이다.

-예전에 많이 사용되었다.



이밖에도 Tanh함수, ELU함수, Maxout등 여러 가지 활성화함수들이 더 있다.

[논리게이트 AND OR XOR 신경망]

AND게이트

-뉴런구조: 입력 2개, 편향, 출력1

-구할 값: 가중치 2개와 편향 1개

Input 1	Input 2	AND
1	1	1
1	0	0
0	1	0
0	0	0

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

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

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

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(x, y, epochs=400, batch_size=1)
```

```
4/4 [==============================] - 0s 1ms/step - loss: 0.0145
Epoch 372/400
4/4 [==============================] - 0s 2ms/step - loss: 0.0144
Epoch 373/400
4/4 [==============================] - 0s 2ms/step - loss: 0.0144
Epoch 374/400
```

가중치와 편향 값

```
for weight in model.weights:
    print(weight)
```

Figure 2: Single Layer Perceptron Network

```
<tf.Variable 'dense_6/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[3.7209592],
       [3.723007 ]], dtype=float32)>
<tf.Variable 'dense_6/bias:0' shape=(1,) dtype=float32, numpy=array([-5.6813374], dtype=float32)>
```

```
model.weights[0]
```

```
<tf.Variable 'dense_6/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[3.7209592],
       [3.723007 ]], dtype=float32)>
```

```
model.weights[1]
```

```
<tf.Variable 'dense_6/bias:0' shape=(1,) dtype=float32, numpy=array([-5.6813374], dtype=float32)>
```

OR게이트

–OR연산 또한 AND연산과 비슷합니다. 다른 점은, 하나라도 참이라면 결과가 참이라는 점입니다.

Input 1	Input 2	OR
1	1	1
1	0	0
0	1	0
0	0	0

XOR게이트

–하나의 퍼셉트론으로는 XOR게이트는 불가능, 뉴런 3개의 2층으로 가능

– 모델이 구해야할 총 매개변수(가중치와 편향)

3(입력) 2 + 3(은닉) * 1(출력) = 9개

– 마빈 민스키와 시모어 페퍼트가 증명

Input 1	Input 2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

● **seqauntial 모델**

```
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')
])
```

Dense층

–가장 기본적인 층

인자 units, activation

– 뉴런수, 활성화 함수

인자 input_shape

–첫 번째 층에서만 정의

–입력의 차원을 명시

Sequential 모델과 딥러닝 구조

입력, 은닉, 출력층

– 패러미터 수

(입력측 뉴런수 +1) * (출력측 뉴런 수)

[회귀와 분류]

회귀모델

-연속적인 값을 예측

ex)광고를 클릭할 확률이 얼마인가?

분류모델

-불연속적인 값을 예측

ex)이 메시지가 스팸인가요 아닌가요?

[회귀 종류]

선형회귀분석

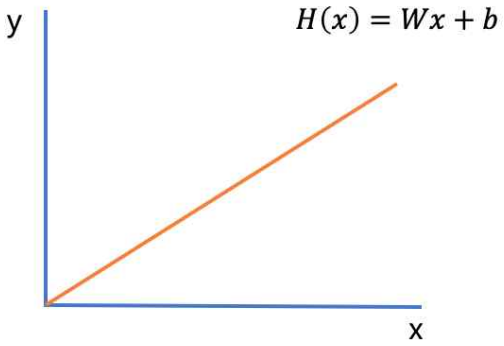
-선형 회귀분석은 변수들 사이의 관계를 분석하는 데 사용하는 통계학적 방법입니다.

-데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 방법.

-이 방법의 장점은 알고리즘의 개념이 복잡하지 않고 다양한 문제에 폭 넓게 적용할 수 있다는 것입니다.

- Y = wX + b

가중치 w와 편향인 b를 구하는 것



단순 선형 회귀분석

- 입력: 특징이 하나 출력:하나의 값

ex) 키로 몸무게 측정

다중 선형 회귀 분석

- 입력: 특징이 여러개 출력: 하나의 값

ex) 아파트 평수

로지스틱 회귀

- 입력: 하나 또는 여러개, 출력: 0 아니면 1

ex)환자 정보로 죽음을 추정

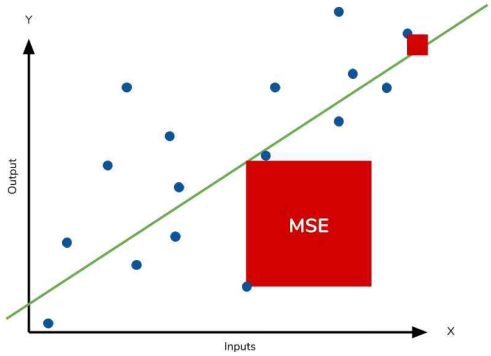
손실함수

-실제값과 가설로부터 얻은 값의 오차를 계산하는 식

-손실함수 값을 최소화 하는 최적의 w와 b를 찾아내려한다.

- 비용함수, 목적함수라고도 부르며, MSE(평균제곱오차)를 사용

● MSE 그래프



-오차는 실제데이터와 예측선의 차이의 제곱의 합이다.

-cost(W,b)를 최소가 되게 만드는 w와 b를 구하면 결과적으로
y와 x의 관계를 가장 잘 나타내는 직선을 그릴 수 있게 됨.

옵티마이저(최적화 과정)

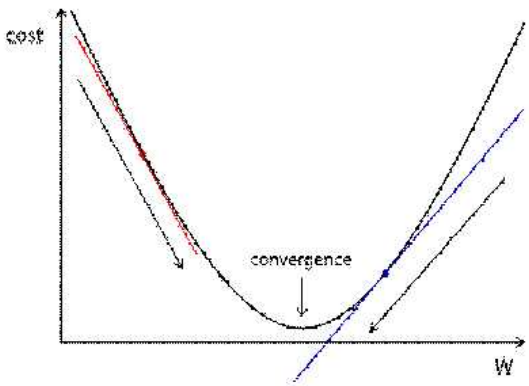
-머신러닝에서 학습

-최적화 알고리즘

- 적절한w와 b를 찾아내는 과정

경사하강법

- 비용함수의 값을 최소로 하는 w와 b를 찾는 방법
- 경사 따라 내려오기
- 항상 볼록 함수 모양을 함



-이때 빨간점의 위치를 A라 하고 파란점의 위치를 B라 하자.
A의 위치에서의 경사의 기울기는 음수 이기 때문에 W의 값을 증가 시키게 되고 B의 위치에서의
경사의 기울기는 양수 이기 때문에 W의 값을 감소 시키게 된다.

cost를 최소화 하는 w를 구하기 위한 식

- 현재 w에서의 접선의 기울기와 ∝와 곱한 값을 현재 w에서 빼서 새로운 w의 값으로 다음 손실을 계산.

초매개변수

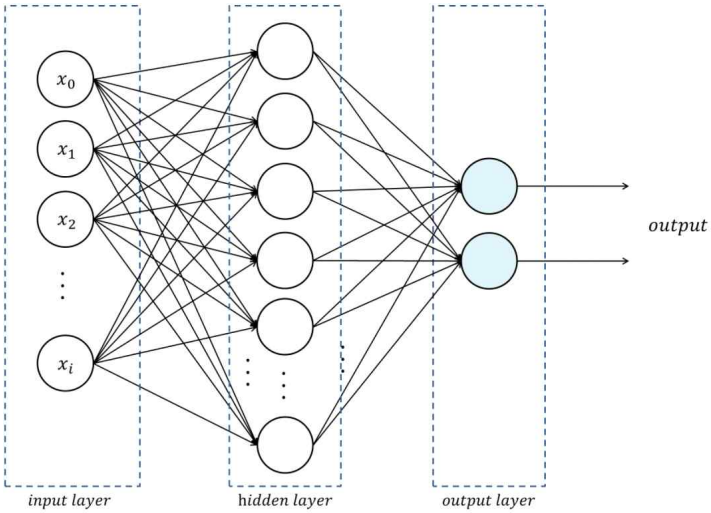
- 딥러닝에서 우리가 설정하는 값
- 학습률은 초 매개변수 중 하나

[오차역전파]

다층 퍼셉트론으로 학습 한다는 것은 최종 출력값과 실제값의 오차가 최소화 되도록 가중치와 바이어스를 계산하여 결정하는 것이다.

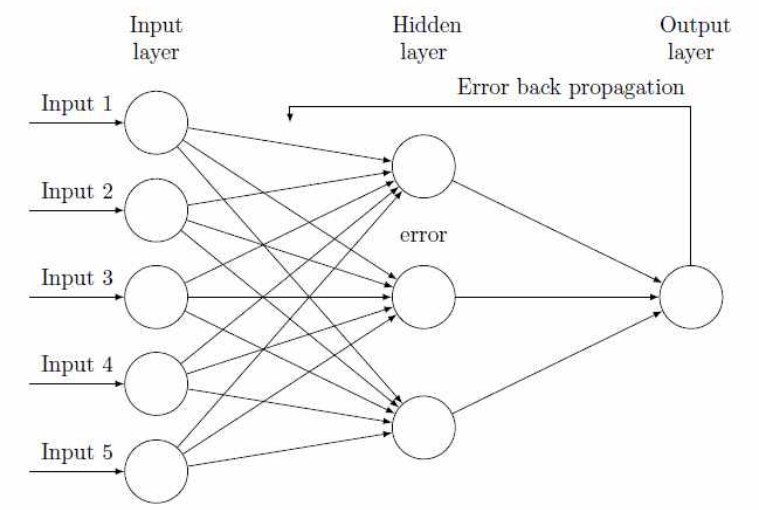
순전파

- 뉴럴 네트워크 모델의 입력층부터 출력층까지 순서대로 변수들을 계산하고 저장하는 것을 의미합니다.



역전파

- 오차 결과 값을 통해서 다시 역으로 입력방향으로 오차가 적어지도록 다시 보내며 가중치를 다시 수정하는 방법



[선형회귀 케라스 구현]

①문제와 정답 데이터 지정

```
x_train=[1, 2, 3, 4]
y_train=[2, 4, 6, 8]
```

②모델 구성

```
model= tf.keras.models.Sequential([
tf.keras.layers.Dense(1, input_shape=(1, ), activation='linear')])
```

- 입력은 1차원 ,출력도 1차원
- 활성화함수 linear

③ 학습에 필요한 최적화 방법과 손실 함수등 지정

```
model.compile(optimizer='SGD', loss='mse', metrics=['mae','mse'])
model.summary()
```

- 확률적 경사하강법
- mae(평균 절대 오차), mse(오차 평균 제곱합)
- 모델 요약 표시

④ 생성된 모델로 훈련 데이터 학습

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

- 훈련과정 정보를 history 객체에 저장

⑤ 테스트 데이터로 성능평가

```
x_test = [1.2, 2.3, 3.4, 4.5]
y_test = [2.4, 4.6, 6.8, 9.0]
print('정확도:', model.evaluate(x_test, y_test))
print(model.predict([3.5, 5, 5.5, 6]))
```

[케라스로 예측 순서]

①케라스 패키지 import

```
import tensorflow as tf
import numpy as np
```

②데이터 지정

```
x = numpy.array([0, 1, 2, 3, 4])
y = numpy.array([1, 3, 5, 7, 9])
```

③인공신경망 모델 구성

```
model = tf.keras.models.Sequntial()
model.add(tf.keras.layers.Dense(출력수, input_shape=(입력수,)))
```

④최적화 방법과 손실함수 지정해 모델 생성

```
model.compile('SGD', 'mse')
```

⑤생성된 모델로 훈련 데이터 학습

```
model.fit(...)
```

⑥성능 평가

```
model.evaluate(...)
```

⑦테스트 데이터로 결과 예측

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

감사합니다.