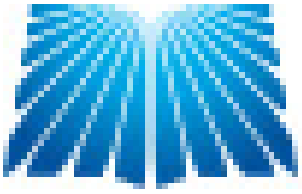


동양미래대학교

지능형 컴퓨팅 포트폴리오

컴퓨터정보공학과 노 지원

위 포트폴리오는 강의 및 강의 자료를 참고했으며, 추가적으로 알고 싶은 내용은 인터넷를 참고하여 작성했습니다.

CONTENT

1 주차

2 주차

3 주차

2. 텐서플로 난수 활용

4 주차

1. MNIST 활용

2. MNIST 데이터 딥러닝 모델 적용 예측

5 주차

1. MNIST 손글씨 예측과 오류 확인

2. MNIST 손글씨 예측과 결과 확인

3. MNIST 손글씨 임의 20 개 정답과 예측, 그림 그리기

4. MNIST 손글씨 예측이 틀린 임의 20 개 정답과 예측 그리기

5. MNIST 손글씨 다양한 구현

6 주차

1. 인공 신경망 퍼셉트론의 이해

2. 인공 신경망 행렬 연산

3. 인공 신경망 행렬 연산 코드

4. 논리 게이트 AND OR XOR 신경망 구현

7 주차

1. 회귀와 분류(regression and classification)

2. 선형 회귀(linear regression)

3. 최적화 과정

4. 선형 회귀 예측

1 주차

1. 인공지능과 딥러닝의 개요

1) AI 시작

A. 앨런 튜링

- 1950 년, 논문 <Computing machinery and intelligence> 을 발표
- 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 근거가 된다 (ex 챗봇)

- 영화 'Imitation Game'

B. 인공지능(Artificial Intelligence)의 처음 사용

- 1956 년 디트모스대 학술대회 : 세계 최초의 AI 프로그램인 논리 연산기(Logic Theorist)를 발표

2) AI 와 딥러닝 역사

A. 1940 년대 부터 시작한 분야

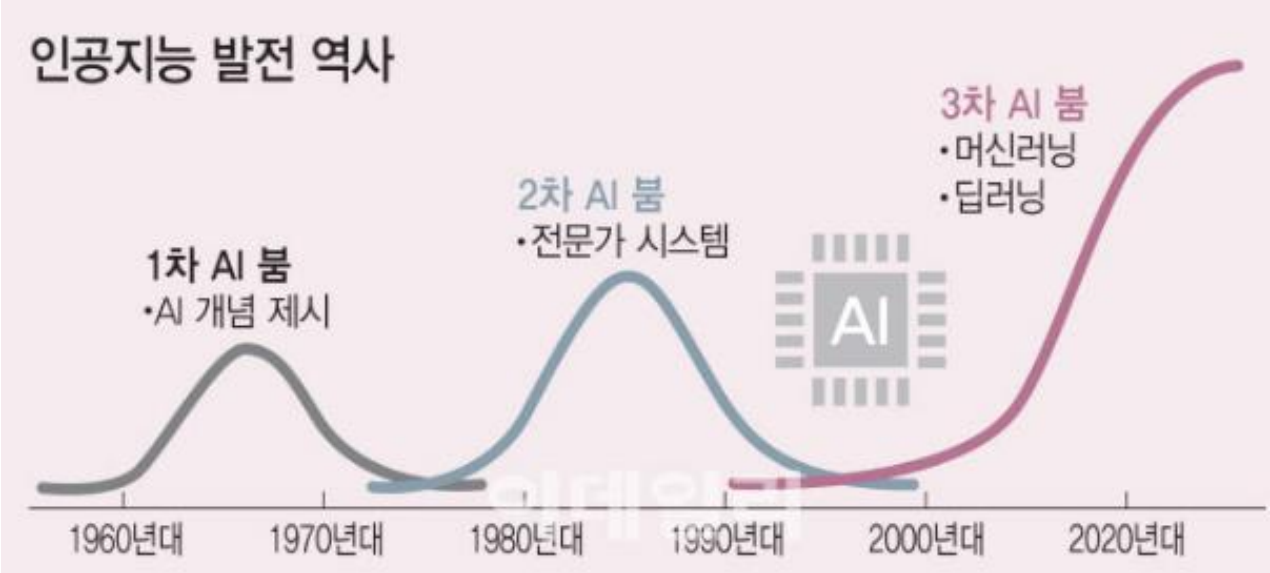
- 두번의 혹한 기를 지냄

AI 의 첫번째 혹한기 1969 - 1980 : 마빈 민스키의 인공 신경망의 퍼셉트론에 대한 비판으로 촉발

AI 의 두번째 혹한기 1987 - 1993 : 규칙 기반의 전문가시스템의 의구심과 한계영화 'Imitation Game'

B. 2010 년 이후 여러 문제 해결

- 최고 전성기를 누림



[출처] 이데일리 기사 중 https://www.edaily.co.kr/news/read?newsId=01279206625937512&mediaCodeNo=257]

3) 인공지능과 머신러닝, 딥러닝

A. 인공지능(AI : Artificial Intelligence)

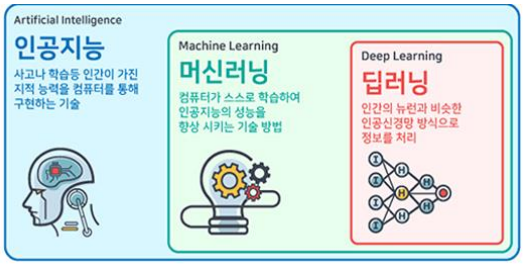
- 컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 모든 기술

- 머신 러닝(machine learning)

: 머신러닝은 기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야

✓ SVM(Support Vector Machine) : 수학적인 방식의 학습 알고리즘

✓ 딥러닝 : 다중 계층의 신경망 모델을 사용하느 머신러닝의 일종



4) 머신러닝

A. 기계학습이라고도 부르는 머신러닝(machine learning)

- 주어진 데이터를 기반으로 기계가 스스로 학습하여 성능을 향상시키거나 최적의 해답을 찾기 위한 학습 방법

- 스스로 데이터를 반복적으로 학습하여 기술을 터득하는 방식

✓ 명시적으로 프로그래밍을 하지 않아도 컴퓨터가 학습을 할 수 있도록 해주는 인공지능의 한 형태

✓ 더 많은 데이터가 유입되면, 컴퓨터는 더 많이 학습을 하고, 시간이 흐르면서 더 스마트해져서 작업을 수행하는 능력과 정확도가 향상

. 머신러닝은 지도학습과 자율학습 그리고 강화학습 분류

- **지도학습(supervised learning)** : 올바른 입력과 출력의 쌍으로 구성된 정답의 훈련 데이터로부터 입출력 간의 함수를 학습시키는 방법

- **비지도(자율)학습(unsupervised learning)** : 정답이 없는 훈련 데이터(unlabeled data)를 사용하여 데이터 내에 숨어있는 어떤 관계를 찾아내는 방법
ex) 군집화(클러스터링) 알고리즘

- **강화학습(reinforcement learning)** : 잘한 행동에 대해 보상을 주고 잘못한 행동에 대해 벌을 주는 경험을 통해 지식을 학습하는 방법
ex) 알파고

5) 머신러닝과 딥러닝 비교

	기계학습	딥러닝
데이터의 의존성	중소형 데이터 세트에서 탁월한 성능	큰 데이터에서 뛰어난 성능
하드웨어 의존성	저가형 머신	GPU 가 있는 강력한 기계
기능 공학	데이터를 나타내는 기능을 이해	데이터를 나타내는 최고의 기능을 이해할 필요가 없음
실행 시간	몇분에서 몇시간	최대 몇주

2. 인공신경망과 DNN

1) 인공신경망에서 시작된 딥러닝

A. 퍼셉트론(perceptron)

- 세계 최초의 인공신경망을 제안 : 1957 년 코넬대 교수, 심리학자인 프랭크 로젠블랫
- 신경망에서는 방대한 양의 데이터를 신경망으로 유입 : 데이터를 정확하게 구분하도록 시스템을 학습시켜 원하는 결과를 얻어냄

B. 현재, 활용 분야

- 항공기나 드론의 자율비행
- 자동자의 자율주행
- 필체 인식
- 음성인식
- 언어 번역

2) ANN 개요

A. 인공신경망(ANN : Artificial Neural Network) 사용

- 인간의 뇌는 1000 억개의 뉴런으로 구성 : 뇌를 구성하는 신경세포 뉴련(Neuron)의 동작원리에 기초한 기술

- 인간의 신경세포인 뉴런(Neuron)을 모방하여 만든 가상의 신경 : 뇌와 유사한 방식으로 입력되는 정보를 학습하고 판별하는 신경 모델

3) 인공신경망 구조와 MLP

A. MLP(Multi Layer Perceptron)

- 입력층(input layer)와 출력층(output layer) : 다수의 신호를 받아 하나의 신호를 출력

- 중간의 은닉층(hidden layer) : 여러 개의 층으로 연결하여 하나의 신경망을 구성

4) DNN(deep neual network)

A. 심층 신경망(deep neural network)

- 다중 계층인 심층신경망을 사용

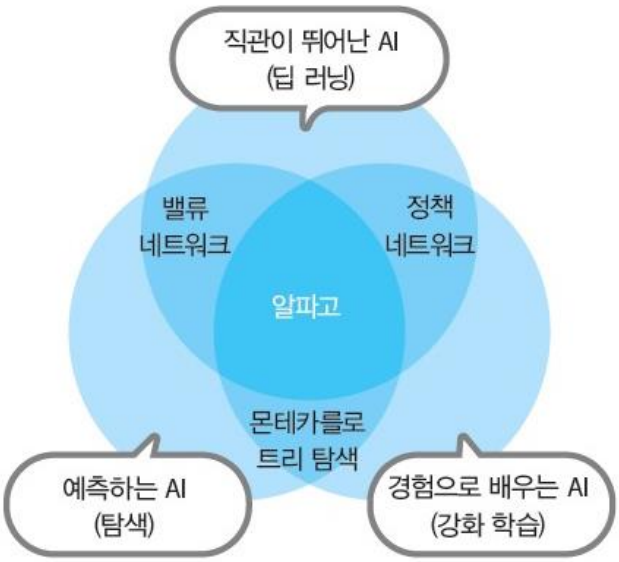
✓ 학습 성능을 높이고 고유 특징들만 스스로 추출하여 학습하는 알고리즘

✓ 입력 값에 대해 여러 단계의 심층신경망을 거쳐 자율적으로 사고 및 결론 도출

3. DNN 활용과 GPU

1) 딥러닝 활용

A. 이세돌을 이긴 알파고



[출처] 0 과 1 이 꿈꾸는 세상 제이펍 https://jpub.tistory.com/938

- 다중 계층의 신경망 구조로 반복 계산에는 많은 계산 능력이 필요하고 이를 고성능의 컴퓨터로 해결

- 스마트폰, 자동차, 스피커, 냉장고, TV 등 모든 주변 기기들에 인공지능이 더해져 지능화되고 있음

B. 이세돌을 이긴 알파고

- 챗봇 , 생체인식, 자동차 번호판 인식 , X-ray 사진 판독 , 드론의 자율비행 , 자동차 자율주행 etc

4. 구글 딥마인드

1) 딥마인드(DeepMind)

- 원래 데미스 하사비스가 2010 년 창업한 영국의 벤처 기업
- 2014 년에 구글에서 인수

2) 2016 년의 알파고

- 구글의 딥마인드에서 개발한 인공지능 바둑 프로그램 : 머신러닝의 강화학습과 신경망의 딥러닝에 적용
- 인터넷상에 있는 3000 만 건의 기본 데이터를 기반으로 1 차적으로 학습 : 다시 컴퓨터끼리 대국을 시켜 기력 향상
- 딥마인드의 알파고는 2017 년 말에 바둑 프로그램의 역할을 종료

5. GPU

A. 그래픽 처리 장치 GPU(Graphics Procressing Unit)

- 그래픽 연산 처리를 하는 전용 프로세서
- GPU 란 용어는 1999 년 엔비디아에서 처음 사용

B. GPUGPU(General Purpose Graphics Procressing Unit)

- 일반 CPU 프로세서를 돕는 보조 프로세서로서의 GPU
- 중앙 처리 장치가 받았던 응용 프로그램들의 계산에 GPU 를 사용하는 기술

GPU 컴퓨팅이란 GPUGPU 를 연산에 참여

고속의 병렬처리로 대량의 행렬과 벡터를 다루는 뛰어난 성능을 발휘

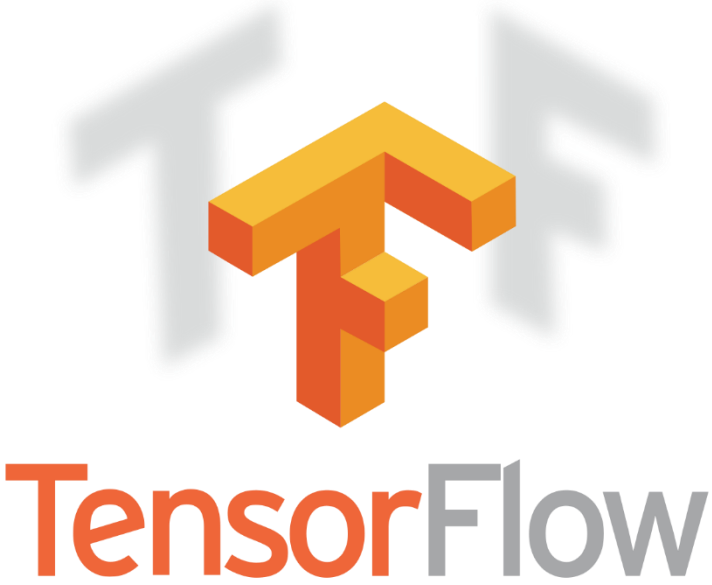
- 딥러닝의 심층신경망에서 빅데이터를 처리하기 위해 대량의 행렬과 벡터를 사용

GPU 사용에 매우 효과적

- 12 개 GPU 가 2,000 개의 CPU 와 비슷한 계산 능력

2 주차

1. 텐서플로 개요



1) 딥러닝 라이브러리(플랫폼) 개요

A. 딥러닝 구현을 위한 클래스 및 함수 제공

B. 다양한 라이브러리 활용 : 텐서플로, 케라스, 파이토지

2) 케라스의 개요

A. 독자적인 고수준 라이브러리

B. 현재는 Tensorflow 의 고수준 API 로도 사용

3) 텐서플로(TensorFlow) 개요

A. 구글(Google)에서 만든 라이브러리

- 연구 및 프로덕션용 오픈소스 딥러닝 라이브러리
- Python, Java, Go 등 다양한 언어를 지원
- 파이썬을 최우선으로 지원

4) 텐서보드(TensorBoard)

A. 브라우저에서 실행 가능한 시각화 도우미

- 딥러닝 학습 과정을 추적하는데 유용하게 사용

5) 텐서 개요

A. Tesor(텐서) : 모든 데이터

- 딥러닝에서 데이터를 표현하는 방식

  0-D 텐서 : 스칼라

  1-D 텐서 : 벡터

  2-D 텐서 : 행렬 등

- N 차원 행렬(배열)

텐서는 행렬로 표현할 수 있는 N 차원 형태의 배열을 높은 차원으로 확장

• • •

6) TensorFlow 계산 과정 [전통적 방식 - 1.0]

A. TensorFlow 에서 텐서(Tensor) 계산 과정

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

B. Session

- 이 통신과정을 담당하는 것이 세션(Session)이라고 부르는 객체
- 생성, 사용, 종료 과정이 필요
- 세션 생성 : Session 객체 생성
- 세션 사용 : run 메소드에 그래프를 입력하면 출력값을 계산하여 반환
- 세션 종료 : close 메소드 , with 문을 사용하면 명시적으로 호출 불필요

2. 코랩으로 시작하는 텐서플로 기초 프로그래밍 (2.0)

1) 텐서플로 2.0 으로 실행

A. 1.x 를 사용 중 일 때

- 메뉴 > 런타임 > 런타임 다시 시작 (단축키 Ctrl +M)

B. Import 방법

```
import tensorflow as tf
tf.__version__
```

C. 텐서 출력

- 값만 보기 위해서는 메소드 numpy()

2) Tensor Shapes with code

↓Vector

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

```
TensorShape([3])
```

↓Matrix(행렬) - 2행 3열

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

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

↓n차원 행렬 (2 * 2 * 3)

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

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

```
[ ]  a
```

```
<tf.Tensor: shape=(2, 2, 3), dtype=int32, numpy=
array([[[1, 2, 3],
        [4, 5, 6]],

       [[1, 2, 3],
        [4, 5, 6]]], dtype=int32)>
```

3) 조건 연산 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
```

4) 1 차원 배열 텐서

A. 합

- **(중요)텐서와 텐서의 합은 각각의 값끼리 더해진다**

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

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

```
[ 6  8 10]
```

5) 배열 텐서 연산

A. 텐서의 브로드캐스팅

- Shape()이 다르더라도 연산은 가능하도록

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

# 중간고사 제출 합니다!
x = tf.constant((np.ones((3, 3))))
y = tf.constant(np.arange (3), dtype=tf.double) # 위 dtype과 맞춰 주기 위해
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]]
```

3 주차

1. 텐서플로 코딩

1) 행렬 곱셈

A. Numpy

- np.dot(a, b)
- a.dot(b)

B. Tf

- tf.matmul

```
# 연산자 오버로딩 지원
print(a)
# 텐서로부터 numpy 값 얻기:
print(a.numpy())
print(b)
print(b.numpy())
print(a * b)
```

```
tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)
[[1 2]
 [3 4]]
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
[[2 3]
 [4 5]]
tf.Tensor(
[[ 2  6]
 [12 20]], shape=(2, 2), dtype=int32)
```

```
# NumPy값 사용
import numpy as np

c = np.multiply(a , b)
print(c)
```

```
[[ 2  6]
 [12 20]]
```

2) 텐서플로 연산

- tf.add() : 더하기
- tf.multiply() : 곱하기
- tf.pow() : 제곱
- tf.reduce_mean() : 평균
- tf.reduce_sum() : 총 합

3) 행렬의 차수 반환

A. tf.rank

행렬의 차수 반환

↓0이 들어 있는 4차원 행렬

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

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

↓4차원이기 때문에 4 출력

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

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

4) Shape 와 reshape

- 기존 내용을 6x10 행렬 형태로 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)>
```

5) 자료형 반환

A. tf.cast

- tf.Tensor 의 자료형을 다른 것으로 변경

정수형 텐서 → 실수형으로 반환

```
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)>
```

2. 텐서플로 난수 활용

1) 균일 분포

A. uniform

- 5 행 4 열의 행렬의 값을 0-1 사이의 난수로 채우기

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

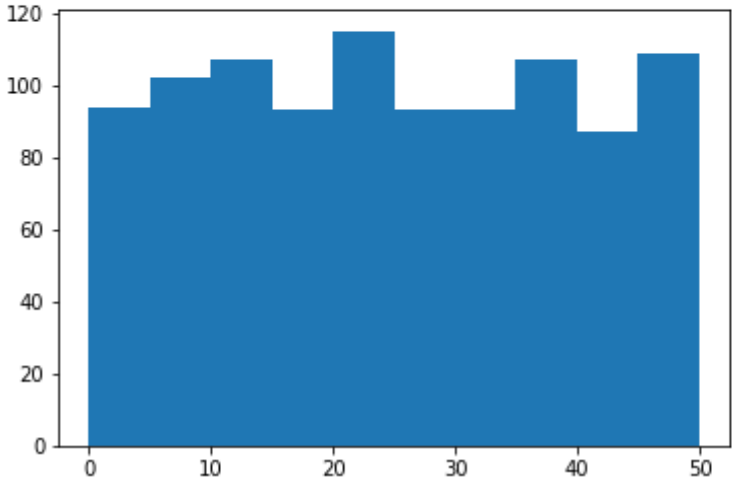
```
tf.Tensor(
[[0.64873457 0.5893861  0.89811504 0.79765785]
 [0.80308497 0.14385486 0.14311087 0.15088856]
 [0.14416373 0.12395048 0.87088704 0.16797483]
 [0.22269821 0.23501396 0.7622087  0.90915406]
 [0.91447544 0.63808215 0.33737397 0.6587868 ]], shape=(5, 4), dtype=float32)
```

B. 균등분포 1000 개 그리기

- bins : 막대 그래프 수

```
import matplotlib.pyplot as plt
rand = tf.random.uniform([1000],0,50)
plt.hist(rand, bins =10)
```

```
(array([ 94., 102., 107.,  93., 115.,  93.,  93., 107.,  87., 109.]),
 array([1.8495321e-02, 5.0107737e+00, 1.0003052e+01, 1.4995330e+01,
        1.9987608e+01, 2.4979885e+01, 2.9972164e+01, 3.4964443e+01,
        3.9956718e+01, 4.4948997e+01, 4.9941277e+01], dtype=float32),
 <a list of 10 Patch objects>)
```



2) 정규 분포

A. normal

- 크기, 평균, 표준편차

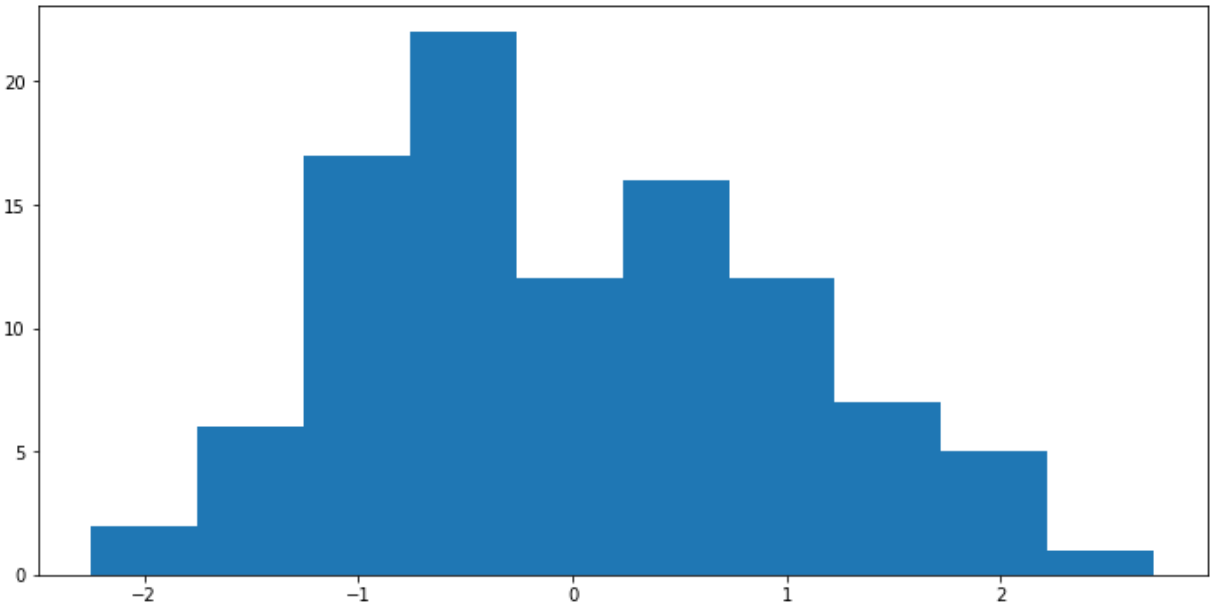
```
# 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
rand = tf.random.normal([4],0,1)
print(rand)
```

```
tf.Tensor([-0.18063734 -1.3515652  -0.25891647 -0.41491646], shape=(4,), dtype=float32)
```

B. 정규 분포 100 개 그리기

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

```
(array([ 2.,  6., 17., 22., 12., 16., 12.,  7.,  5.,  1.]),
 array([-2.2474983 , -1.7511833 , -1.2548683 , -0.75855327, -0.26223826,
         0.23407674,  0.73039174,  1.2267067 ,  1.7230217 ,  2.2193367 ,
         2.7156518 ], dtype=float32),
 <a list of 10 Patch objects>)
```

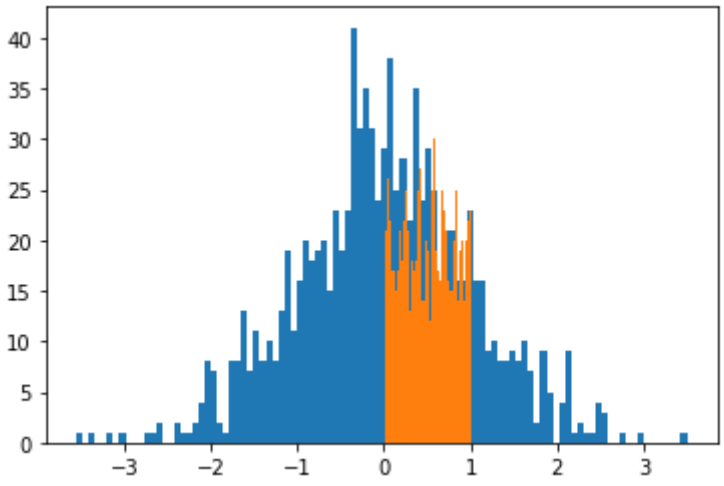


C. 균등분포와 정규분포의 비교

- uniform : 배열 커질수록 사각형 모양

- normal : 크기 커질수록 종 모양

```
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)
```



4 주차

1. MNIST 이해

1) MNIST 데이터 셋

A. MNIST(Modified National Institute of Standards and Technology)

- 미국 국립 표준 기술원(NIST)
- 손으로 쓴 자릿수에 대한 데이터 집합
- 숫자의 범위는 0 에서 9 까지, 총 10 개의 패턴을 의미
- 크기가 28 x 28 픽셀인 회색조 이미지
- 이미지의 정답: 필기 숫자 이미지가 나타 내는 실제 숫자, 0 에서 9
- 대표적인 두 가지의 데이터 가져오는 방법
    - ✓ tensorflow.keras.datasets.mnist
    - ✓ tensorflow.examples.tutorials.mnist

2) 케라스 딥러닝 구현

A. DEFINE

- 딥러닝 모델 만듦

B. COMPILE

- 최적화 방법, 손실 함수, 훈련 모니터링 지표

C. FIT

D. EVALUATE

E. PREDICT

3) MNIST 손글씨 데이터 코드

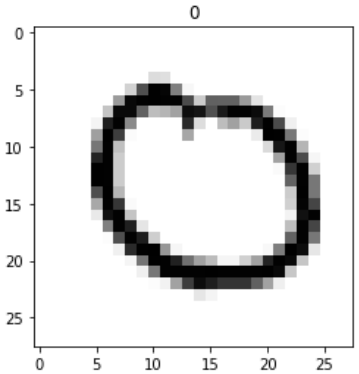
A. 손글씨 데이터 구조 확인

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

# MNIST 형태를 알아 봅시다. 데이터 수, 행렬 형태 등
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)

# MNIST 훈련 데이터의 내부 첫 내용도 알아보자.
print(x_train[0])
print(y_train[0])

# MNIST 테스트 데이터의 내부 첫 내용도 알아보자.
print(x_test[0])
print(y_test[0])
```

B. 훈련 데이터 첫 손글씨

```
import matplotlib.pyplot as plt

n = 108
ttl = str(y_train[n])
plt.figure(figsize=(6, 4))
plt.title(ttl)
plt.imshow(x_train[n], cmap='Greys')
```

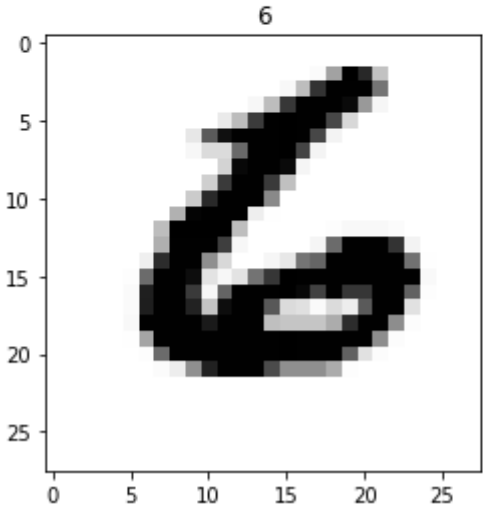
```
<matplotlib.image.AxesImage at 0x7fabfb4bf978>
```



C. 테스트 데이터 첫 손글씨

```
n = len(y_test)-1
ttl = str(y_test[n])
plt.figure(figsize=(6, 4))
plt.title(ttl)
plt.imshow(x_test[n], cmap='Greys')
```

```
<matplotlib.image.AxesImage at 0x7f1bf1e94e80>
```



D. 훈련용 데이터 60000 개 중에서 임의 손글씨 출력

```
#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()
```

2. MNIST 데이터 딥러닝 모델 적용 예측

1) 딥러닝 구현 순서

A. 훈련과 정답 데이터 지정

- 데이터 전처리

B. 모델 구성

C. 학습에 필요한 최적화 방법과 손실 함수 등 설정

- 구성된 모델 요약

D. 생성된 모델로 훈련 데이터 학습

E. 테스트 데이터로 성능 평가

- 테스트 데이터 또는 다른 데이터로 결과 예측

2) 주요 용어

A. 데이터셋

- 훈련용과 테스트용

✓ Train data set, Test data set

✓ x(입력, 문제), y(정답, 레이블)

B. 모델

- 딥러닝 핵심 신경망, 여러 층 구성
- 완전 연결층 : DENSE()

- 1 차원배열로 평탄화 : FLATTEN()

C. 학습 방법의 여러 요소들

- 옵티마이저(optimizer), 최적화 방법, 손실 함수

D. 딥러닝 훈련

- Epochs

3) 딥러닝 전 소스

```
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 설정
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)
```

5 주차

1. MNIST 손글씨 예측과 오류 확인

1) 테스트 데이터의 첫 번째 손글씨 예측 결과

A. model.predict(input)

- input 값 : 모델의 fit(), evaluate()에 입력과 같은 형태가 필요
- 첫 번재 손글씨만 알고 싶어도 슬라이스해서 사용 → x_test[:1]

```
# 테스트 데이터의 첫 번째 손글씨 예측 결과를 확인
print(x_test[:1].shape)

pred_result = model.predict(x_test[:1])
print(pred_result.shape)
print(pred_result)
print(pred_result[0])
```

```
(1, 28, 28)
(1, 10)
[[6.6551010e-08 4.6572151e-08 1.6924533e-06 4.6316294e-05 5.3104167e-11
  2.7725026e-08 1.3161651e-13 9.9994850e-01 2.1508410e-07 3.0247127e-06]]
[6.6551010e-08 4.6572151e-08 1.6924533e-06 4.6316294e-05 5.3104167e-11
 2.7725026e-08 1.3161651e-13 9.9994850e-01 2.1508410e-07 3.0247127e-06]
```

- 정답으로 나온 10 개의 실수는 확률 값

B. One hot encoding , argmax()

- one hot encoding 하나의 자리만 1, 나머지 0
- argmax()로 가장 큰 수 위치 첨자 반환

```
import numpy as np

# 원 핫 인코딩과 argmax학습
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))
```

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

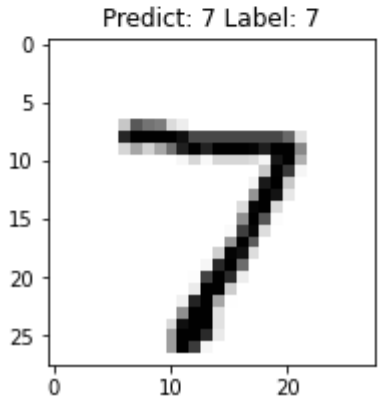
2. MNIST 손글씨 예측과 결과 확인

1) 손글씨 예측, 결과 비교

A. 맞은 결과는 7

```
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)

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")
```

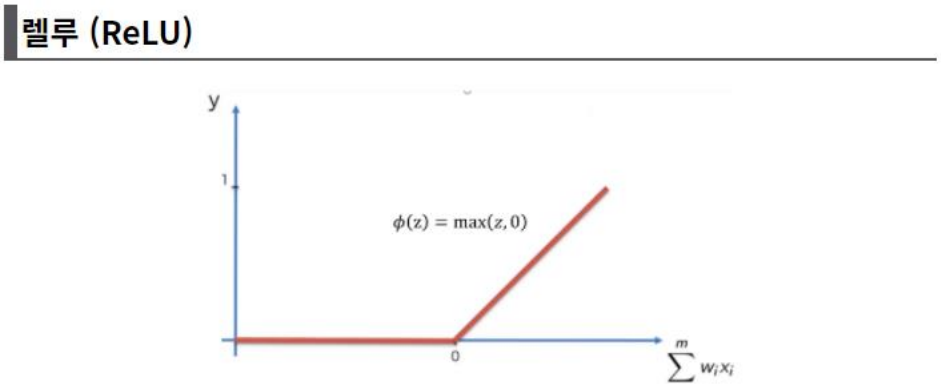
```
0.9999999
7
```



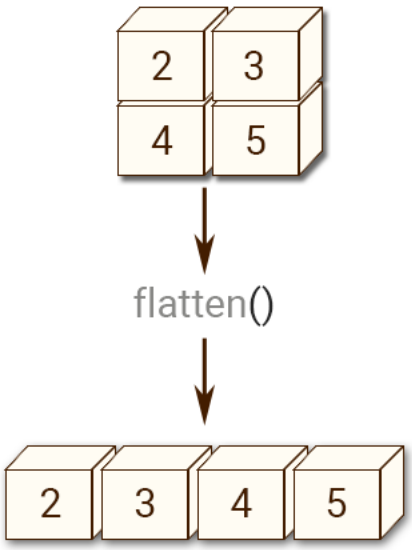
2) 활성화 함수 softmax()

- Transfer Function 으로부터 전달 받은 값을 출력할 때 일정 기준에 따라 출력값을 변화시키는 비선형 함수

- 대표적인 활성화 함수 : Sigmoid, tanh, ReLU



3) 평탄화 메소드 flatten



- 평탄화 작업

4) 드롭아웃

A. 개념

- 2012 년 토론토 대학의 힌튼 교수와 그 제자들이 개발
- 층에서 결과 값을 일정 비율로 제거하는 방법

B. 코드

```
import tensorflow as tf

tf.random.set_seed(0)
layer = tf.keras.layers.Dropout(.3, input_shape=(2, ))
outputs = layer(data, training=True)
print(outputs)
np.sum(outputs)
```

```
tf.Tensor(
[[ 0.         0.       ]
 [ 4.285714   5.714286 ]
 [ 7.1428576  8.571428 ]
 [10.        11.428572 ]
 [12.857143   0.       ]], shape=(5, 2), dtype=float32)
60.0
```

- 30% 드롭아웃

3. MNIST 손글씨 임의 20 개 정답과 예측, 그림 그리기

1) 임의의 20 개 예측 값과 정답 그리기

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

```
# 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()
```

4. MNIST 손글씨 예측이 틀린 임의 20 개 정답과 예측 그리기

1) 예측이 잘못된 20 개 샘플 찾기

- 틀린 것을 임의의 20 개를 찾아 첨자를 리스트 samples 에 저장

```
from random import sample
import numpy as np

# 예측이 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]:       # 예측이 틀린 조건
    mispred.append(n)
print('정답이 틀린 수', len(mispred))

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

```
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()
```

5. MNIST 손글씨 다양한 구현

1) 중간층을 늘리고 훈련 횟수 증가

```
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(64, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

#훈련에 사용할 옵티마이저와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
#모델 요약 표시
model.summary()
#모델을 훈련 데이터로 총 20번 훈련
model.fit(x_train, y_train, epochs=20)
#모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

2) 메소드 flatten() 미사용

- reshape()로 평탄화 작업을 수행 후 Dense() 층 사용

```
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

# 평탄화 작업 수행
x_train = x_train.reshape((60000, 28*28))
x_test = x_test.reshape((10000, 28*28))

# 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', input_shape=(28 * 28,)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

#모델 요약 표시
model.summary()

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

#모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=20)
#모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

6 주차

1. 인공 신경망 퍼셉트론의 이해

1) 퍼셉트론 기초 개념

- 다수의 신호를 입력받아서 하나의 신호를 출력함
- 뉴런에서 신호를 전달하는 돌기가 있는 것처럼 퍼셉트론에도 존재 → **weight(가중치)**

2) 퍼셉트론 학습 방법

- 임의로 설정된 weight 로 시작해 학습 데이터를 퍼셉트론 모형에 입력하며

분류가 잘못 됐을 때 weight 를 개선해 나감

Perceptron Learning Algorithm (PLA)

Initialize weights: $\vec{w} = (w_0, \dots, w_d)^T = (0, \dots, 0)^T \in \mathbb{R}^{d+1}$

while(mis-classified examples exist):

Select a mis-classified example: $(\vec{x}^{(t)}, y^{(t)})$ ← Misclassified example means: With the current weights $y^{(t)} \neq sign(\vec{w}^T\vec{x}^{(t)})$

Update weights: $\vec{w} := \vec{w} + y^{(t)} \cdot \vec{x}^{(t)}$ (more generally $\eta y^{(t)}\vec{x}^{(t)}$)

return $\vec{w}$

1. A challenge: Algorithm will not terminate for non-linearly separable data (outliers, noise).
2. Unstable: jump from good perceptron to really bad one within one update.
3. Attempting to minimize: $\min_{\vec{w}} \frac{1}{N}\sum_{t=1}^{N} [\![y^{(t)} \neq sign(\vec{w}^T\vec{x}^{(t)})]\!]$ ← NP-hard.

75

3) 가중치와 편향

- **가중치**(weight)는 입력신호가 결과 출력에 주는 영향도를 조절하는 매개변수

- **편향**(bias)은 뉴런(또는 노드; x 를 의미)이 얼마나 쉽게 **활성화**(1 로 출력; activation)되느냐를 조정하는(adjust) 매개변수

2. 인공 신경망 행렬 연산

1) 인공 신경망과 행렬 연산 관계

A. 행렬의 필요성

- 단층 퍼셉트론에 입력 계층의 노드가 무수히 많거나 계층의 수가 무수히 많은 경우, 입력 값과 가중치의 곱 그리고 활성함수 수작업으로 계산 한계

→ 효율적인 접근법이 **행렬**

B. 행렬을 통한 신경망 가중치 계산

$$\begin{pmatrix} w_{1.1} & w_{2.1} \\ w_{1.2} & w_{2.2} \end{pmatrix} \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix} = \begin{pmatrix} (input\_1 * w_{1.1}) + (input\_2 * w_{2.1}) \\ (input\_1 * w_{1.2}) + (input\_2 * w_{2.2}) \end{pmatrix}$$

$$\begin{pmatrix} (input\_1 * w_{1.1}) + (input\_2 * w_{2.1}) \\ (input\_1 * w_{1.2}) + (input\_2 * w_{2.2}) \end{pmatrix} = \begin{pmatrix} output\_1 \\ output\_2 \end{pmatrix}$$

- 축약해서 단순하게 다음과 같이 표현 함

X = W * I

W : 가중치 행렬, I : 입력 값들의 행렬, X : 입력 값과 가중치의 곱과 합, 계층 2 로 들어온 신호들의 조정된 합으로 이루어진 행렬

3. 인공 신경망 행렬 연산 코드

1) 행렬 연산

- 샘플 수 4 개의 행렬 연산

```
x = [[6, 5], [4, 7], [5, 6], [6, 7]]
w = [[1, 2, 3], [4, 5, 6]]

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

```
array([[26, 37, 48],
       [32, 43, 54],
       [29, 40, 51],
       [34, 47, 60]], dtype=int32)
```

4. 논리 게이트 AND OR XOR 신경망 구현

1) AND 게이트 구현

- 항상 참일 때만, 참을 결과로 출력하는 연산

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]])    #입력
y = np.array([[1], [0], [0], [0]])            #아웃풋

model = tf.keras.Sequential([   #units : 결과 / input_shape=(2,) : [1, 1]
    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()

history = model.fit(x, y, epochs=500, batch_size=1)
```

2) OR 게이트 구현

- 하나라도 참이라면 결과가 참

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

```
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()

history = model.fit(x, y, epochs=500, batch_size=1)
```

3) OR 게이트 구현

- 홀수 개의 입력이 참일 때, 결과가 참

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

```
  # 3.27 tf.keras를 이용한 XOR 네트워크 계산
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()
```

7 주차

1. 회귀와 분류(regression and classification)

1) 회귀

- 회귀는 보통 연속적인 숫자, 즉 예측값이 float 형태인 문제들을 해결하는데 사용=
- **출력에 연속성 있음**

2) 분류

- 예/아니오로 구분될 수 있는 문제
- **불연속적인 값을 예측**

2. 선형 회귀(linear regression)

1) 선형 회귀 모델

- 선형 회귀는 한 개 이상의 독립 변수 x 와 y 의 선형 관계를 모델링
- 어떤 변수의 값에 따라서 특정 변수의 값이 영향 받음 (다른변수의 값을 변하기하는 변수를 x, x 에 의해서 값이 종속적으로 변하는 변수 y 라고 가정)
- 이때 x 값은 독립적으로 변할 수 있는 반면, y 값은 계속해서 x 의 값에 의해 종속적으로 결정 되므로 x 는 독립 변수, y 는 종속 변수라고 함

A. 단순 선형 회귀 분석(Simple Linear Regression Analysis)

- 입력 : 특징이 하나 , 출력 : 하나의 값 (ex) 키로 몸무게 추정

$$y = Wx + b$$

B. 다중 선형 회귀 분석(Multiple Linear Regression Analysis)

- 입력 : 특징이 여러 개, 출력 : 하나의 값

$$y = W_1x_1 + W_2x_2 + \ldots W_nx_n + b$$

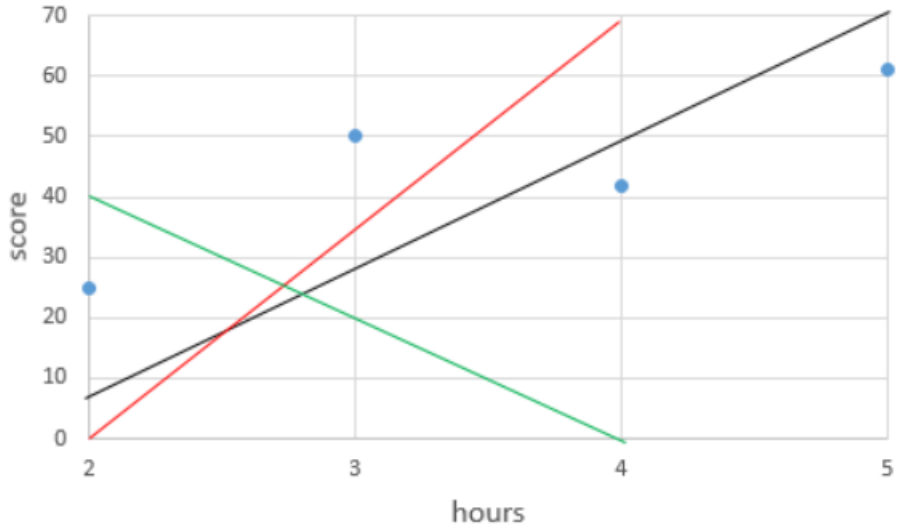
C. 로지스틱 회귀(Logistic Regression)

- 이진 분류(Binary Classification)
- 입력 : 하나 or 여러 개, 출력 : 0 or 1

2) 가설(Hypothesis)

- x 와 y 의 관계를 유추하기 위해서 수학적으로 식을 세워보게 되는데 머신러닝에서는 이러한 식을 가설이라고 함

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



3) 손실 함수(Loss function)

- 목적 함수(Objective function), 비용 함수(Cost function)
- 실제 값과 예측 값에 대한 오차에 대한 식

A. MSE(Mean Squared Error)

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- 예측한 값과 실제 값 사이의 평균 제곱 오차를 정의
- 차가 커질수록 제곱 연산으로 값이 더욱 뚜렷
- 제곱으로 오차가 양수이든 음수이든 누적 값을 증가시킴

3. 최적화 과정

1) 최적화(Optimization)

- 최적화 알고리즘
- 적절한 W 와 b 를 찾아내는 과정

Gradient Descent(경사하강법) : 비용함수의 값을 최소로 하는 W 와 b 찾는 방법

2) 경사 하강법

- 접선의 기울기 : 맨아래의 볼록한 부분에서는 결국 접선의 기울기 0
- cost 가 최소화가 되는 지점은 접선의 기울기가 0 이 되는 지점

- 비용 함수를 미분하여 현재 W 에서의 접선의 기울기를 구하고 접선의 기울기가 낮은 방향으로 W 의 값을 변경하고 다시 미분함. 이 과정을 접선의 기울기가 0 인 곳을 향해 W 의 값을 변경하는 작업 반복하는 것

4. 선형 회귀 예측

1) Y= 2X

- 하나의 Dense 층
- 활성화 함수 linear

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 1. 문제와 정답 데이터 지정
x_train = [1, 2, 3, 4]
y_train = [2, 4, 6, 8]

# 2. 모델 구성(생성)
model = Sequential([
        # 출력, 입력= 여러 개 원소의 일차원 배열, 그대로 출력
    Dense(1, input_shape=(1, ), activation='linear')
    #Dense(1, input_dim=1)
])

model.compile(optimizer='SGD', loss='mse', metrics=['mae', 'mse'])
model.summary()
model.fit(x_train, y_train, epochs=1000)

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]))
```

2) Y= 2X +1

```
import tensorflow as tf
import numpy as np

#훈련과 테스트 데이터
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

#인공신경망 모델 사용
model = tf.keras.models.Sequential()

#은닉계층 하나 추가
model.add(tf.keras.layers.Dense(1, input_shape=(1, )))

#모델의 패라미터를 지정하고 모델 구조를 생성
#최적화 알고리즘 : 확률적 경사 하강법(SGD: Stochastic Gradient Descent)
#손실 함수(loss function): 평균제곱오차(MSE: Mean Square Error)
model.compile('SGD', 'mse')

#생성된 모델로 훈련 자료로 입력(x[:2])과 출력(y[:2])을 사용하여 학습
#키워드 매개변수 epoch(에폭) : 훈련반복횟수
#키워드 매개변수 verbose: 학습진행사항 표시
model.fit(x[:3], y[:3], epochs=1000, verbose=0)

#테스트 자료의 결과를 출력
print('Targets(정답):', y[3:])

#학습된 모델로 테스트 자료로 결과를 예측(model.predict)하여 출력
print('Predictions(예측):', model.predict(x[3:]).flatten())
```