지능형 컴퓨팅과정 포트폴리오 경진대회

Introduction to Artificial Intelligence	
과목	인공지능응용프로그래밍
교수	강환수교수님
학과	컴퓨터정보공학과
학번	20190706
이름	김종휘

머리말

인공지능은 평소 제게 있어서 가까우면서도 멀게 느껴졌습니다. 많은 매체에서 보고 들었고 이론적으로도 배웠지만 잘 와 닫지 않았습니다. 하지만 이번에 인공지능응용프로그램을 들으면서 1학기에 배웠던 Python을 사용하여 직접 코드를 작성해보고 텐서플로우에서 머신러닝의 다양한 예제들을 통해 좀 더 쉽게 인공지능에 접근하는 기회가 되었습니다.
처음 배우는 머신러닝을 공부하는 과정 중 생소한 단어들과 전문용어들이 많아 크고 작은 어려움 들이 있었습니다. 하지만 온라인 강의를 반복적으로 듣고 정리하다 보니 단어들은 자연스레 익혀지며 어려움 또한 해결되었습니다.

이 포트폴리오는 수업 중 중요하다고 판단하며 시험 출제가 예상되는 부분들을 기록하여 공부한 내용 들을 정리한 것입니다.

1주차

인공지능 > 머신러닝 > 딥러닝
위 3개를 데이터 분석이 겹친다.

AI의 시작

- 1950년 논문 <앨런 튜링>
- 이미테이션 게임

AI의 첫 번째 암흑기 1969 - 1980 마빈 민스키
AI의 두 번째 암흑기 1987 - 1993
2010년 이후 최고의 전성기

인공지능

컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 모든 기술

머신러닝

기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야
SVM : 수학적인 방식의 학습 알고리즘

★ 머신러닝 분류 개요

지도학습(supervised learninng)
- 올바른 입력과 출력의 쌍으로 구성된 정답의 훈련 데이터로 학습시키는 방법
비지도 (자율)학습(unsupervised learning)
- 정답이 없는 훈련 데이터 사용하여 관계를 찾아내는 방법
(군집, 시각화, 차원축소, 연관규칙 학습)
강화학습(reinforcement lerarning)
- 잘한 행동에 보상, 잘못된 행동에 벌을 주는 경험을 통해 지식 학습하는 방법
ex) 알파고

딥러닝

다중 계층의 신경망 모델을 사용하는 머신러닝의 일종

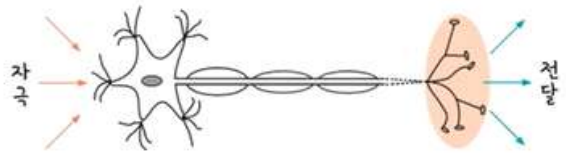
머신러닝과 딥 러닝의 차이점

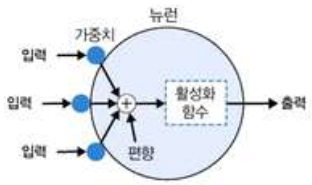
	머신러닝	딥 러닝
데이터 의존성	적은 데이터	많은 데이터
하드웨어 의존성	저가형 머신	고가형 머신
기능 공학	기능을 이해해야 함	기능을 이해할 필요 없음
실행 시간	몇 분에서 몇 시간	최대 몇주

★ 퍼셉트론

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

현재 항공기, 드론, 자율주행, 필체 음성인식, 언어 번역 등 여러 분야에서 사용





인공 신경망 (ANN) : 인간의 신경세포인 뉴런을 모방 (퍼셉트론을 포함한 큰 범위)
MLP : 뉴런 여러 개를 연결 입력층, 출력층, 중간의 은닉층으로 구성됨
심층신경망 (DNN) : 다중 계층의 신경망

☆ 그래픽 처리 장치 GPU(Graphics Processing Unit)
- 그래픽 연산 처리를 하는 전용 프로세서
- 1999년 엔비디아에서 처음 사용

GPGPU
- CPU 보다 계산 속도가 빨라 CPU 프로세스를 돕는다.

☆ CUDA : NVIDIA의 GPU를 사용하기 위한 라이브러리 소프트웨어

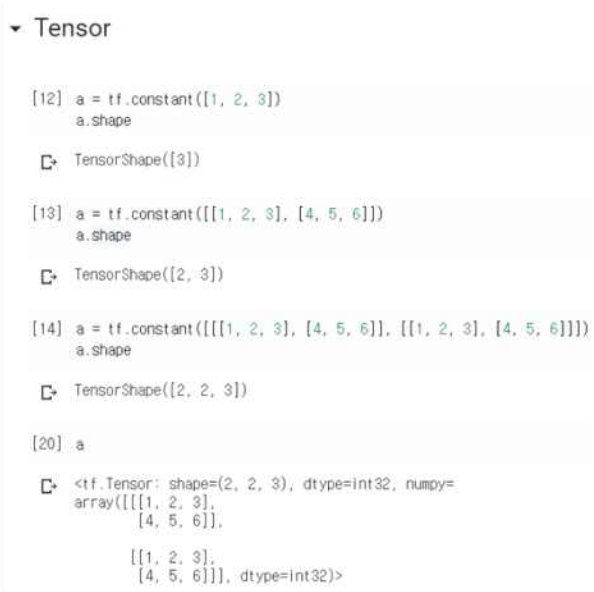
2주차

텐서플로 : 구글에서 만든 오픈소스 딥러닝 라이브러리
 - 다양한기능 제공, 개발용 API 제공, python, java, go 등 다양한 언어 지원
케라스 : 텐서플로의 포함된 고급 API
넘파이 : 행렬이나 대규모 다차원 배열을 쉽게 처리하도록 지원하는 파이썬 라이브러리
텐서 : 딥러닝에서 데이터 표현하는 방식

0-D 텐서 : 스칼라
차원이 없는 텐서 - 10
1-D 텐서 : 벡터
1차원 텐서 - [10, 20, 30]
2-D 텐서 : 행렬
2차원 텐서 - [[1,2,3], [4,5,6]]
텐서 : n차원 행렬

TensorFlow에서 텐서 계산 과정
모두 그래프라고 부르는 객체 내에 저장되어 실행
그래프를 계산하기 위해 세션(Session)이라는 객체 필요
1.0버전 때 사용 2.0 때는 사용 불가능

구글의 Colab
- 파이썬과 머신러닝, 딥러닝 개발 클라우드 서비스
- 구글 드라이브, 깃허브와 연계가능
셀의 마지막에 있으면 print를 안해도 출력이 보임



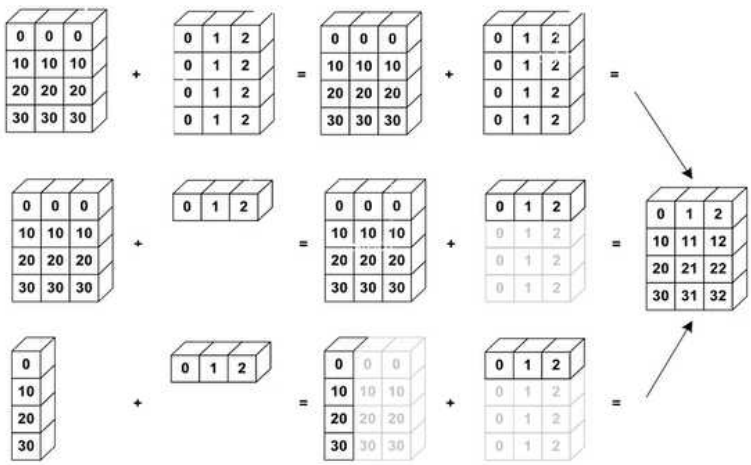
1차원 배열 텐서

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

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

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

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

     [ 6  8 10]

[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.]
```

Python

★ 텐서의 브로드 캐스팅

- shape이 다르더라도 연산이 가능하도록 가지고 있는 값을 이용해 shape를 맞춤



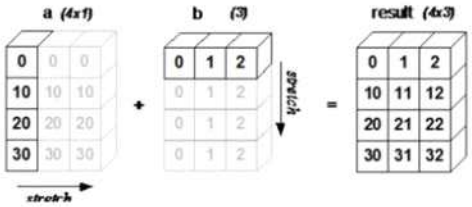
arange(3) => [0, 1, 2]

ones((3, 3)) => 1을 채워준 3행 3열 (그 1은 실수이다.)

arange(3).reshape(3,1) => 같은 원소의 구조 변경

브로드캐스팅 코드 1

- **Numpy**
  - mp.arange()




```
[17] 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]]
```

```
[44] 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)
```

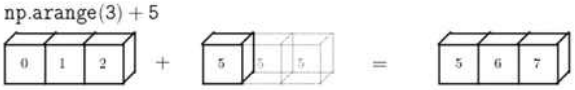
```
[0 1 2]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

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)
```


브로드캐스팅 코드 2

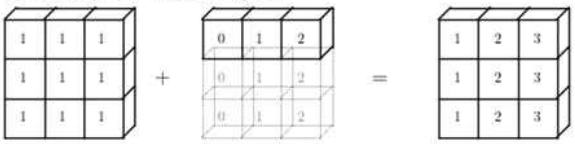
```
[45] x = tf.constant((np.arange(3)))
     y = tf.constant([5], dtype=tf.int64)
     print((x+y).numpy())

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

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

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

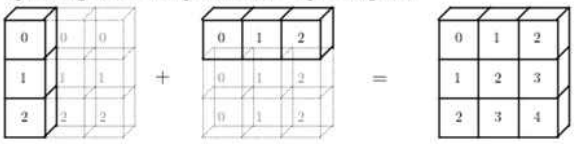
np.arange(3) + 5



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



np.arange(3).reshape((3, 1)) + np.arange(3)

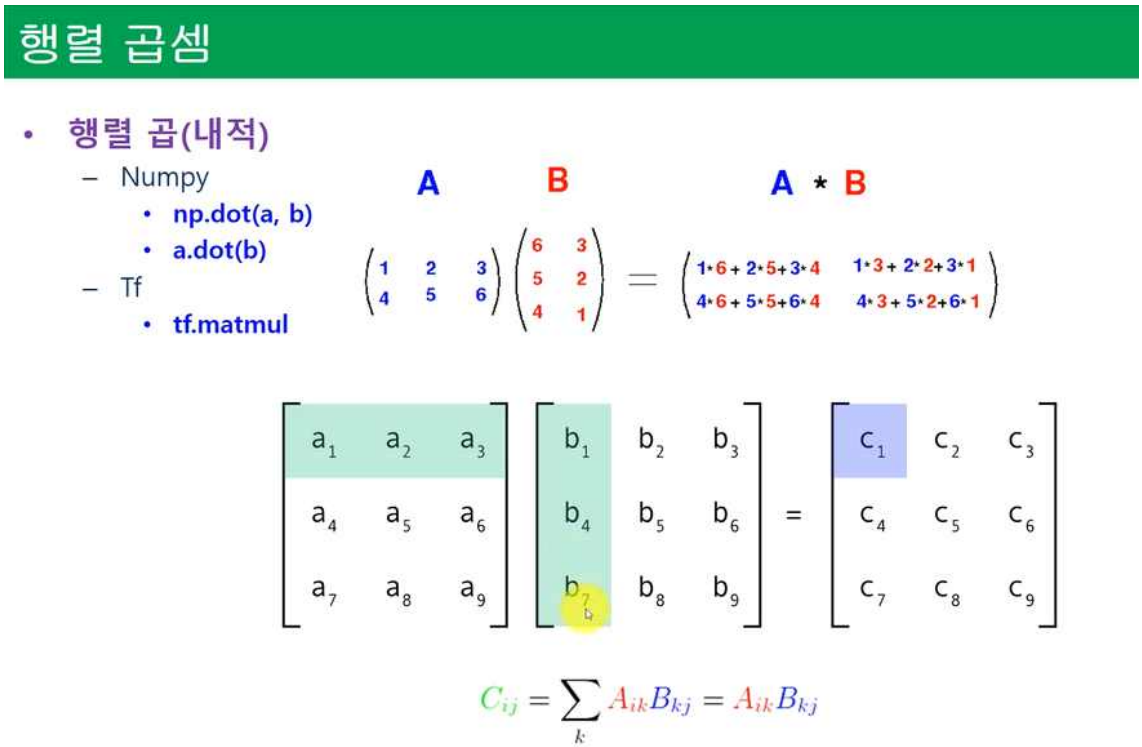


3주차

★ 행렬 곱셈

- Numpy
  1. np.dot(a, b)
  2. a.dot(b)
- Tf

tf.matmul



A = 2 x 3 B = 3 x 2 => 2 x 2
A의 행과 B의 열이 반드시 같아야 한다
교환 법칙 x

shape : 모양을 출력
ones([3, 4, 5]) = 3 x 4 x 5
reshape : 기존의 내용을 형태 변경
-1은 차원 크기를 계산하여 동으로 결정
reshape(matrix, [3, -1])
matrix가 60일시 = 3 x 20 행렬로 변경

행렬 곱셈

- **tf.matmul()**
  - 2차원 행렬 곱셈

```
[7]  x = [[2.]]
     m = tf.matmul(x, x)
     print(m)
     print(m.numpy())

     tf.Tensor([[4.]], shape=(1, 1), dtype=float32)
     [[4.]]
```

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

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

     [[ 4.  4.]
      [10.  8.]]
```

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

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

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

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

Python

보통 2차원 행렬을 가지고 행렬 곱셈을 한다.
print(a * b) 는 matmul(a,b)와 다르다.

행렬의 같은 위치 원소와의 곱

행렬, 원소와의 곱

```
[15] # 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)
```

```
[14] # NumPy값 사용
     import numpy as np

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

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

4주차

★ MNIST 데이터셋

- 딥러닝 손글씨 인식에 사용되는 데이터셋
- 손으로 쓴 자릿수에 대한 데이터 집합
- 필기 숫자 이미지와 정답인 레이블의 쌍으로 구성

784 픽셀 (28 x 28), 흑백 이미지, 훈련데이터 6만개 테스트데이터 1만개

케라스 딥러닝 구현 (5개 과정)

딥러닝 모델 만듬 -> 훈련방법 설정 -> 훈련 -> 테스트 평가 -> 정답 예측

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

x = 문제 y = 정답

train = 학습데이터

test = 테스트데이터

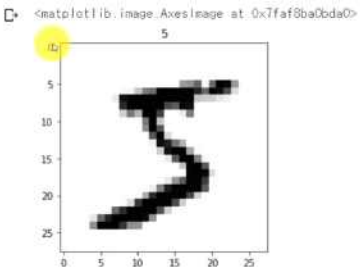
```
#MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_test), (x_test, y_test) = mnist.load_data()
#MNIST 형태를 알아봅시다. 데이터 수 행렬 형태 등
print (x_train.shape, y_train.shape)
print (x_test.shape, y_test.shape)
#MNIST 훈련 데이터의 내부 첫 내용도 알아보자
print (x_train[0 ])
print (y_train[0 ])
#MNIST 테스트 데이터의 내부 첫 내용도 알아보자
print (x_test[0 ])
print (y_test[0 ])
```

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

행렬 내용 직접 출력

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

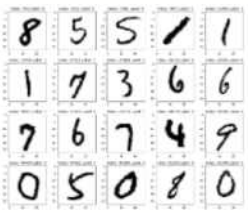
훈련 데이터 첫 손글씨 써보기



```
import  matplotlib.pyplot as  plt
n = 0
ttl = str(y_train[n]) # 정답
plt.figure(figsize=(6 , 4 ))# 사이즈 6*4
plt.title(ttl)                # 위에 5 출력
plt.imshow(x_train[n], cmap='Greys')
```

첫 손글씨와 마지막 손글씨 그려보기

```
# MNIST 데이터(훈련, 테스트)의 내부 첫 내용을 그려보자
import  matplotlib.pyplot as  plt
tmp = "Label: " + str(y_train[0 ])        #학습용 첫 번째  (0번째)
plt.title(tmp)
plt.imshow(x_train[0 ], cmap="Greys")
plt.show()
tmp = "Label: " + str(y_test[0 ])         #테스트용 첫 번째  (0번째)
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                    # 학습용 마지막 (59999째)
tmp = "Label: " + str(y_train[idx])
plt.title(tmp)
plt.imshow(x_train[idx], cmap="Greys")
plt.show()
idx = len (x_test) -1                     # 테스트용 마지막 (9999째)
tmp = "Label: " + str(y_test[idx])
plt.title(tmp)
plt.imshow(x_test[idx], cmap="Blues")
plt.show()
```



훈련용 데이터 60000개중 임의 손글씨 출력

```
#랜덤하게 20개의 훈련용 자료를 그려보자.
from  random import  sample
nrows, ncols = 4 , 5   #출력 가로 세로 수
idx = sorted (sample(range (len (x_train)), nrows * ncols))
count = 0
plt.figure(figsize=(12 , 10 ))   #전체 그려지는 사이즈 가로 12 세로 10
for  n in  idx:                    #리스트 값
  count += 1
  plt.subplot(nrows, ncols, count)   #count 1~20번
  tmp = "Index: " + str(n) + " Label: " + str(y_train[n])
  plt.title(tmp)
  plt.imshow(x_train[n], cmap="Greys")
plt.tight_layout()
plt.show()
```

데이터 전처리(정규화)

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

tf.keras.layers.Dropout(0.2) (드롭아웃)

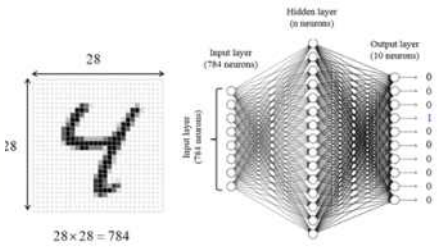
- 훈련 중에 20%를 중간에 끊음

모델 요약

```
# 훈련에 사용할 옵티마이저와 손실 함수, 출력정보를 모델에 설정
```

MNIST 딥러닝 구현 전 소스, 약 97% 정확도

```
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()

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

# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성 (쌓는 층을 ','로 구분)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28 , 28 )),    #1차원으로 평탄화
    tf.keras.layers.Dense(120   , activation='relu'),    #히든 층 뉴런의 수
 / 활성화 함수 relu
    tf.keras.layers.Dropout(.2 ),                       #중간 20% 끊기
    tf.keras.layers.Dense(10   , activation='softmax')   # 아웃 풋 10개 /활
성화 함수 softmax 확률값으로 나타냄

])
#훈련에 사용할 옵티마이저와 손실 함수, 출력정보를 선택
model.compile (optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
               # metrics=[‘accuracy’, ‘mse’])
#모델 요약 표시
model.summary()
#모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5 )
#모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```



5주차

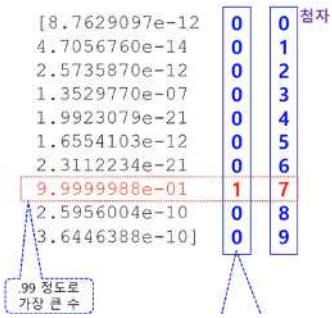
테스트 데이터의 첫 번째 손글씨 예측 결과를 확인

- 첫 번째 손글씨만 알아보더라도 3차원 배열로 입력
- 슬라이스 해서 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)
[[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]
```

```
import  numpy as  np
# 10개의 수를 더하면?
one_pred = pred_result[0 ]
print (one_pred.sum ())      #1.0
#혹시 가장 큰 수가 있는 첨자가 결과
one = np.argmax(one_pred)
print (one)                  # 7
```



Tensorflow 메소드

tf.reduce_sum()

tf.argmax()

One Hot Encoding

데이터가 취할 수 있는 모든 단일 범주에 대해 하나의 새 열을 생성

가장 큰 값의 첨자 구하기

```
import numpy as np

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))
```

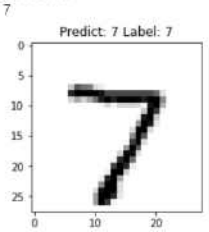
```
2
3
[1 0 2]
```

```
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)
```

```
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")
```

```
1.0000001
7
```



Flatten

2차원 배열(28 x 28 픽셀)의 이미지 포맷을 28*28=784 픽셀의 1차원 배열로 변환

Dropout

over-fitting을 줄이기 위한 regularization 기술 네트워크에서 일시적으로 유닛(인공 뉴런, artificial neurons)을 배제하고, 그 배제된 유닛의 연결을 모두 끊는다.

테스트 데이터 모두 예측해보기

```
from  random import  sample
import  numpy as  np

# x_test로 직접 결과 처리
pred_result = model.predict(x_test)   #x_test전부 pred_result에 옮김 (만개)
print (pred_result.shape)             #10000개를 10개
print (pred_result[0 ])               #첫번째의 결과
print (np.argmax(pred_result[0 ]))    #정답

# 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)
[6.0543699e-09 4.7576525e-09 4.4922908e-06 6.0105299e-06 1.7090477e-11
 1.2512787e-07 1.7061310e-13 9.9998450e-01 1.9574335e-08 4.8704469e-06]
7
[7 2 1 ... 4 5 6]
[7 2 1 ... 4 5 6]
```

임의의 20개 예측 값과 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 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()
```

예측이 잘못된 샘플 찾기

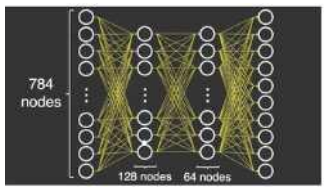
```
# 예측이 틀린 것 첨자를 저장할 리스트
mispred = []
# 예측한 softmax의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)
# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1 )
for  n in  range (0 , len (y_test)):
  if  pred_labels[n] != y_test[n]:       # 예측이 틀린 조건
    mispred.append(n)
print ('정답이 틀린 수', len (mispred))
#랜덤하게 틀린 것 20개의 첨자 리스트 생성
samples = sample(mispred, 20 )
print (samples)
```

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

★ MNIST 손글씨 다양한 구현

중간층을 늘리고 훈련 횟수를 증가 (정확도 증가!)

```
# 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)
```

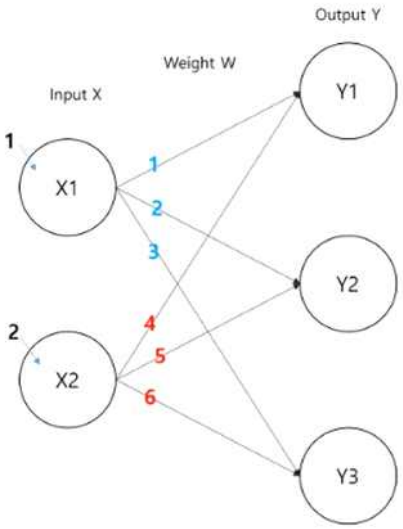
메소드 flatten() 미사용

reshape()로 평탄화 작업을 수행 후 Dense() 층 사용
flatten()를 사용하지 않아도 가능한다. 정확도, 모델 변동 없음

```
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')
])
```

6주차
★ and, or, xor 딥러닝 코딩

계산 사례



X * W = Y

1 x 2 * 2 x 3 = 1 x 3

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

© sacko

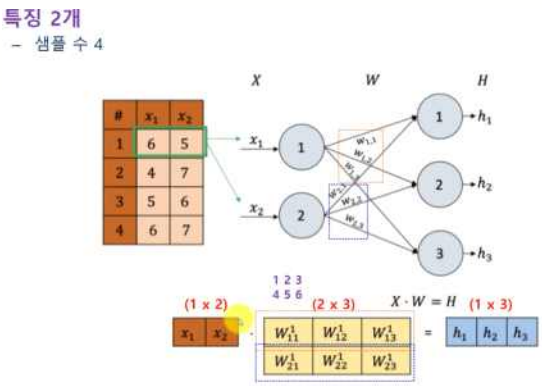
```
Y1 = (1*1) + (2*4) = 9
Y2 = (1*2) + (2*5) = 12
Y3 = (1*3) + (2*6) = 15
```

```
x = [[1 , 2 ]]
w = [[1 , 2 , 3 ], [4 , 5 , 6 ]]
y = tf.matmul(x, w)                # (1 * 2) (2 * 3) => (1 * 3)
y.numpy()
```

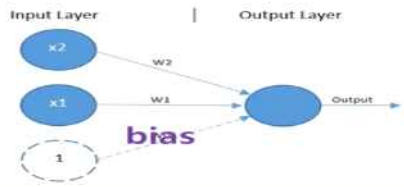
결과
array([[9, 12, 15]], dtype=int32)

```
샘플 수 4개의 행렬 연산                # (4*2) (2*3) => (4*3)
x = [[6 , 5 ], [4 , 7 ], [5 , 6 ], [6 , 7 ]]
w = [[1 , 2 , 3 ], [4 , 5 , 6 ]]
y = tf.matmul(x, w)
y.numpy()
```

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



AND 게이트 구현
가중치 2개와 편향 1개




```
# 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 /
    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()

input_shape=(2,) : 입력값 형태[1, 1]   lr(학습률) : 0.3 loss(손실값)
Output : 1, Param (가중치 + 편향) : 3
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(x, y, epochs=500 , batch_size=1 )
500번 학습
Epoch 1/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0926
Epoch 2/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0908
Epoch 3/500
4/4 [==============================] - 0s 1ms/step - loss: 0.0891
```

OR 게이트 구현

```
and에서 x, y값만 변경
x = np.array([[1 ,1 ], [1 ,0 ], [0 ,1 ], [0 ,0 ]])    #입력
y = np.array([[1 ], [1 ], [1 ], [0 ]])                #아웃풋
```

XOR 게이트 구현

뉴런 3 개의 2층으로 가능

- 모델이 구해야 할 총 매개변수(가중치와 편향)
  - 3 * 2 + 3 * 1 = 9개

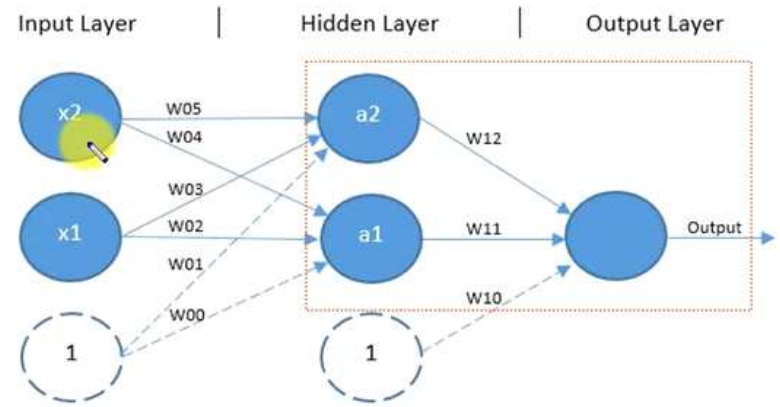


Figure 4: Multilayer Pereceptron Architecture for XOr

Dense를 2개 생성
(3 * 2) (3 * 1) 패러미터 수 = 9

```
 # 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()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 2)                 6
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
```

```
history = model.fit(x, y, epochs = 1500, batch_size=1)

# 3.28 tf.keras 를 이용한 XOR 네트워크 학습
plt.plot(history.history['loss'])

# tf.keras를 이용한 XOR 네트워크 평가
model.predict(x)

# 네트워크 가중치와 편향 확인
for  weight in  model.weights:
  print (weight)
  print ()
```

Sequential 모델과 딥러닝 구조

- 입력, 은닉, 출력 층
  - 패러미터 수
    - (입력측 뉴런 수 + 1) * (출력측 뉴런 수)

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 2)                 6
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

```
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')
])
```

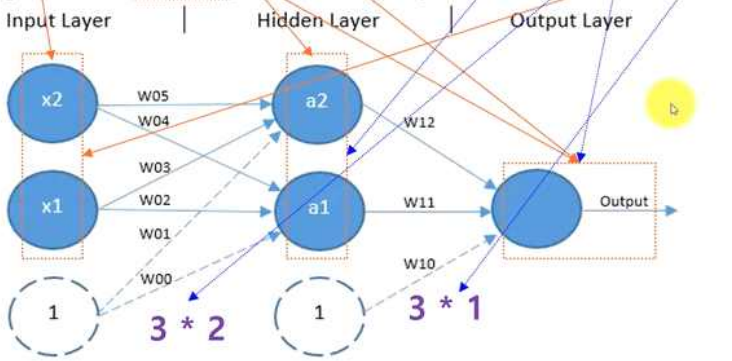


Figure 4: Multilayer Pereceptron Architecture for XOr

7주차

회귀(regresson)와 분류(classification)

회귀모델
- 연속적인 값을 예측

분류 모델
- 불연속적인 값을 예측

가중치와 편향을 파라미터라 함

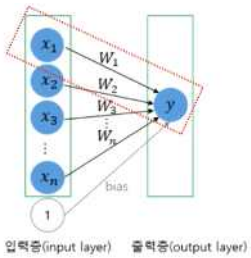
가설 : 가중치와 편향, 기울기와 절편
손실 함수 : MSE(평균제곱오차), Categorical crossentropy,
Spars Categorical crossentropy

오차역전파
순전파 : 입력층에서 출력층으로 계산해 최종 오차를 계산하는 방법
역전파 : 오차 결과 값을 통해 다시 역으로 input 방향으로 오차가 적어지도록 다시 보내며 가중치를 다시 수정하는 방법, 1986년 제프리 힌튼이 적용 (엄청난 속도 증가)
경사 하강법 : 비용 함수의 값을 최소로 하는 W와 b를 찾는 방법

★ 회귀 코딩
y = 2x에 해당하는 값을 예측

선형 회귀 문제

- y = 2x 에 해당하는 값을 예측
  - 훈련(학습) 데이터
    - x_train = [1, 2, 3, 4]
      y_train = [2, 4, 6, 8]
  - 테스트 데이터
    - x_test = [1.2, 2.3, 3.4, 4.5]
      y_test = [2.4, 4.6, 6.8, 9.0]
  - 예측, 다음 x에 대해 예측되는 y를 출력
    - [3.5, 5, 5.5, 6]



```
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
         # 출력 1개 입력도 1개
    Dense(1 , input_shape=(1 , ), activation='linear')
    #Dense(1, input_dim=1)
])
# 3. 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저와 손실 함수, 출력 정보를 지정
# Mean Absolute Error, Mean Squared Error
model.compile (optimizer='SGD', loss='mse', metrics=['mae', 'mse'])
# 모델을 표시 (시각화)
model.summary()

# 4. 생성된 모델로 훈련 데이터 학습
# 훈련과정 정보를 history 객체에 저장
model.fit(x_train, y_train, epochs=1000 )
# 5. 테스트 데이터로 성능 평가
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))
# x = [3.5, 5, 5.5, 6]의 예측
print (model.predict([3.5 , 5 , 5.5 , 6 ]))

# 예측 값만 1차원으로
print(pred.flatten())
print(pred.squeeze())
```

```
정확도: [9.136034350376576e-05, 0.008526384830474854, 9.136034350376576e-05]
[[ 6.9956603]
 [ 9.984034 ]
 [10.980158 ]
 [11.976282 ]]
```

확률적 경사 하강법 사용
- optimizer = ‘SGD’

```
[[ 6.9934297]
 [ 9.975829 ]
 [10.969961 ]
 [11.964094 ]]
[ 6.9934297  9.975829  10.969961  11.964094 ]
[ 6.9934297  9.975829  10.969961  11.964094 ]
```

mae : 평균 절대 오차
- 모든 예측과 정답과의 오차 합의 평균

mse : 오차 평균 제곱합
- 모든 예측과 정답과의 오차 제곱 합의 평균

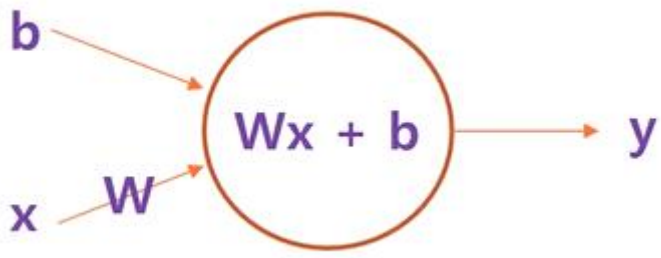
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
```

y = 2x + 1 예측

- 케라스와 numpy 사용
- 학습에 3개 데이터
  - x = [0, 1, 2, 3, 4]
    - x[:3]
  - y = [1, 3, 5, ?, ?]
    - y[:3]
- 예측
  - 뒤 2개 데이터 사용
  - x = [0, 1, 2, 3, 4]
    - x[3:]
  - y = [1, 3, 5, ?, ?]
    - y[3:]

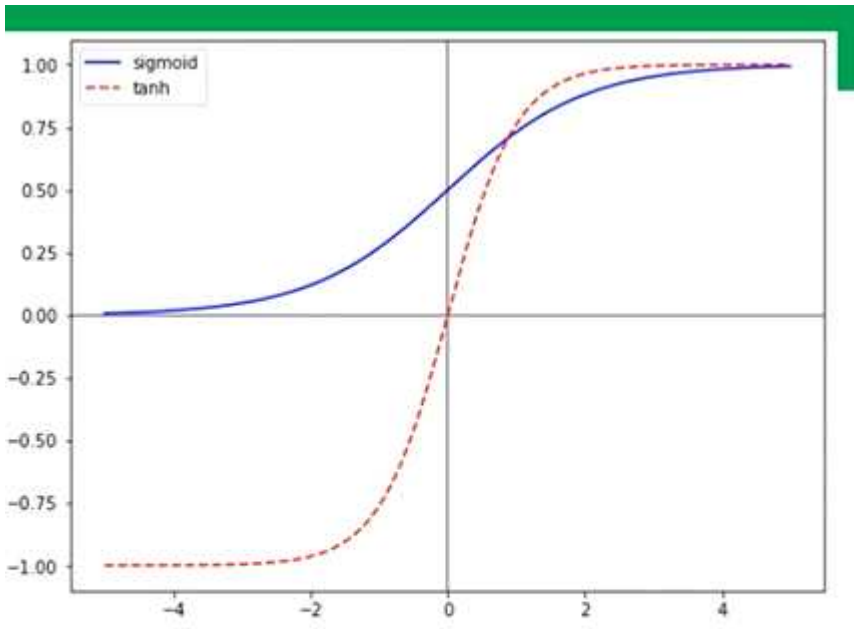
```
import  tensorflow as  tf
import  numpy as  np

#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 , )))
#model = Sequential([Dense(1, input_shape=(1, ), activation='linear')

#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())
```

```
Targets(정답): [7 9]
Predictions(예측): [6.9956455 8.992945 ]
```



9주차

인구증가율과 고령인구비율



$$\tanh(z) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \sigma(x) = \frac{1}{1 + e^{-x}}.$$

활성화 함수 tanh

```
# 4.10 딥러닝 네트워크의 회귀선 확인

# 그림 4.2 출력 코드
import  math
def  sigmoid (x ) :
    return  1  / (1  + math.exp(-x))
x = np.arange(-5 , 5 , 0.01 )
sigmoid_x = [sigmoid(z) for  z in  x]
tanh_x = [math.tanh(z) for  z in  x]
plt.figure(figsize=(8 , 6 ))
plt.axhline(0 , color='gray')
plt.axvline(0 , color='gray')
plt.plot(x, sigmoid_x, 'b-', label='sigmoid')
plt.plot(x, tanh_x, 'r--', label='tanh')
plt.legend()
plt.show()
```

중간층
- 뉴런 6개
출력층
- 뉴런 1개

```
# 4.7 딥러닝 네트워크를 이용한 회귀
import  tensorflow as  tf
import  numpy as  np
# 인구증가율과 고령인구비율
X = [0.3 , 0.78 , 1.26 , 0.03 , 1.11 , 0.24 , 0.24 , -0.47 , -0.77 , -0.37
, -0.85 , -0.41 , -0.27 , 0.02 , -0.76 , 2.66 ]
Y = [12.27 , 14.44 , 11.83 , 18.75 , 17.52 , 16.37 , 19.78 , 19.51 , 12.65
, 14.74 , 10.72 , 21.94 , 12.83 , 15.51 , 17.14 , 14.42 ]
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=6 , activation='tanh', input_shape=(1 ,)),
    tf.keras.layers.Dense(units=1 )
])
model.compile (optimizer=tf.keras.optimizers.SGD(lr=0.1 ), loss='mse')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 12
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
```

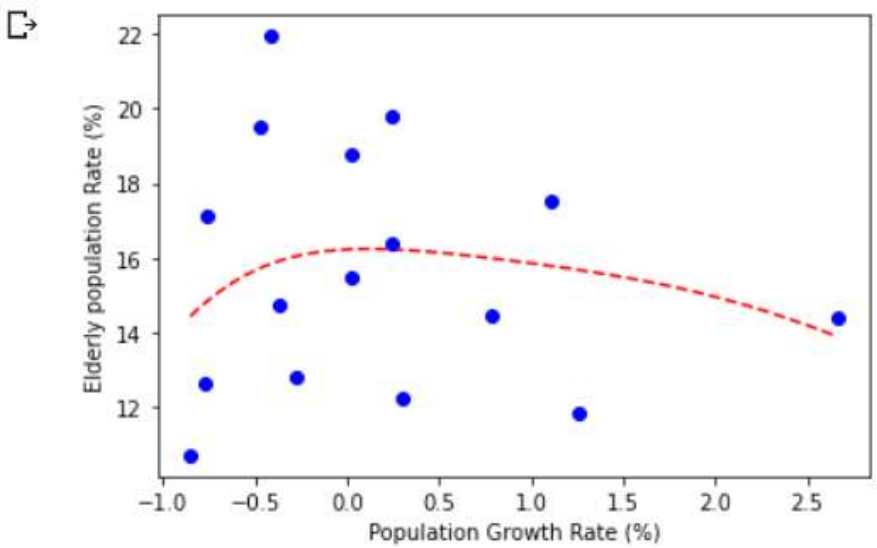
```
# 4.8 딥러닝 네트워크의 학습
model.fit(X, Y, epochs=10 )
```

```
Epoch 1/10
1/1 [==============================] - 0s 1ms/step - loss: 9.3669
Epoch 2/10
1/1 [==============================] - 0s 996us/step - loss: 9.3340
Epoch 3/10
1/1 [==============================] - 0s 1ms/step - loss: 9.2972
Epoch 4/10
1/1 [==============================] - 0s 2ms/step - loss: 9.2567
Epoch 5/10
1/1 [==============================] - 0s 1ms/step - loss: 9.2123
Epoch 6/10
1/1 [==============================] - 0s 1ms/step - loss: 9.1644
Epoch 7/10
1/1 [==============================] - 0s 1ms/step - loss: 9.1138
Epoch 8/10
1/1 [==============================] - 0s 1ms/step - loss: 9.0614
Epoch 9/10
1/1 [==============================] - 0s 1ms/step - loss: 9.0083
Epoch 10/10
1/1 [==============================] - 0s 977us/step - loss: 8.9553
<tensorflow.python.keras.callbacks.History at 0x7fb498f481d0>
```

```
# 4.9 딥러닝 네트워크의 Y값 예측
model.predict(X)
```

```
array([[16.2184  ],
       [15.998303],
       [15.681905],
       [16.237036],
       [15.788992],
       [16.232372],
       [16.232372],
       [15.731767],
       [14.795802],
       [15.918132],
       [14.439468],
       [15.850084],
       [16.05417 ],
       [16.235107],
       [14.837127],
       [13.905417]], dtype=float32)
```

```
# 4.10 딥러닝 네트워크의 회귀선 확인
import  matplotlib.pyplot as  plt
line_x = np.arange(min (X), max (X), 0.01 )
line_y = model.predict(line_x)
plt.plot(line_x, line_y, 'r--')        빨간색 점선
plt.plot(X,Y,'bo')                     파란색 점
plt.xlabel('Population Growth Rate (%)')
plt.ylabel('Elderly population Rate (%)')
plt.show()
```



입력층 중간층 출력층으로는 위 자료 정도만 나온다.
층을 크게 할 시 좀 더 구부러진 선이 나올 수 있다.

케라스 모델 미사용 텐서플로 프로그래밍

optimizer
- 최적화 과정(복잡한 미분 계산 및 가중치 수정)을 자동으로 진행
  (SGD, adam)

학습률(learning rate)
- 보통 0.1 ~ 0.0001

변수 Variables

딥러닝 학습에서 최적화 과정
- 모델의 매개변수 즉, 가중치 및 편향을 조정하는 것

변수 tf.Variable
- 프로그램에 의해 변화하는 공유된 지속 상태를 표현하는 가장 좋은 방법
- 모델 파라미터를 저장하는데 tf.Variable을 사용

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

메소드 minimize()

첫 번째 인자
- 최소화할 손실 함수

두 번째 인자 var_list
- 학습시킬 변수 리스트, 가중치와 편향

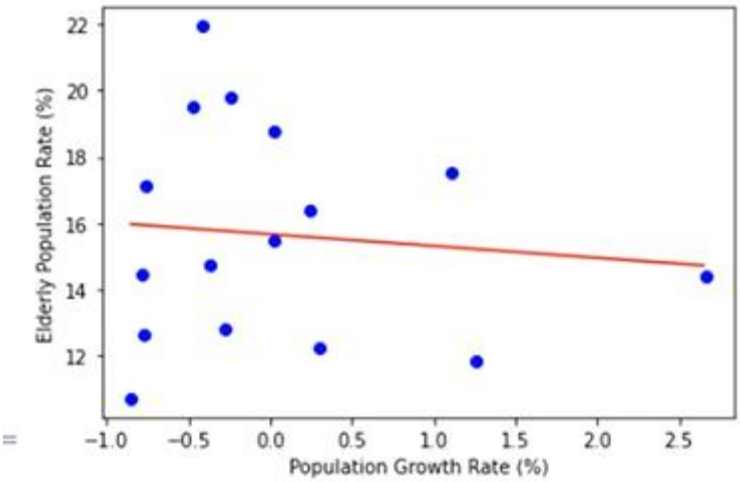
1000번의 학습을 거쳐
- 잔차의 제곱 평균을 최소화하는 적절한 값 a, b에 도달을 기대

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

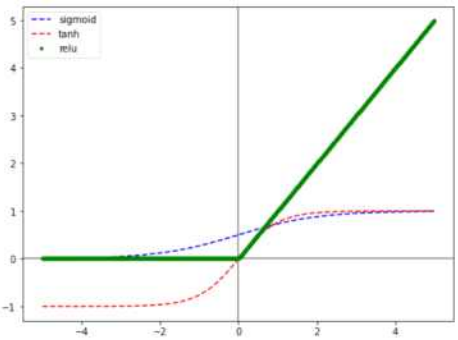
```
# 4.4 텐서플로를 이용해서 회귀선 구하기
import  tensorflow as  tf
import  numpy as  np
# 인구증가율과 고령인구비율
X = [0.3 , 0.78 , 1.26 , 0.03 , 1.11 , 0.24 , 0.24 , -0.47 , -0.77 , -0.37
, -0.85 , -0.41 , -0.27 , 0.02 , -0.76 , 2.66 ]
Y = [12.27 , 14.44 , 11.83 , 18.75 , 17.52 , 16.37 , 19.78 , 19.51 , 12.65
, 14.74 , 10.72 , 21.94 , 12.83 , 15.51 , 17.14 , 14.42 ]
# a와 b를 랜덤한 값으로 초기화합니다.
# a = tf.Variable(random.random())
# b = tf.Variable(random.random())
a = tf.Variable(tf.random.uniform([1 ], 0 , 1 ))
b = tf.Variable(tf.random.uniform([1 ], 0 , 1 ))
# 잔차의 제곱의 평균을 반환하는 함수입니다.
def  comput_loss () :
    y_pred = a* X + b
    loss = tf.reduce_mean((Y - y_pred) ** 2 )
    return  loss
optimizer = tf.keras.optimizers.Adam(lr=0.07 )
for  i in  range (1000 ):
  # 잔차의 제곱의 평균을 최소화(minimize)합니다.
  optimizer.minimize(comput_loss, var_list=[a, b])
  if  i % 100  == 99 :
        print (i, 'a:', a.numpy(), 'b:', b.numpy(), 'loss:', comput_loss().numpy())
```

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

```
import  matplotlib.pyplot as  plt

line_x = np.arange(min (X), max (X), 0.01 )
line_y = a * line_x + b

# 그래프를 그립니다.
plt.plot(line_x, line_y, 'r--')
plt.plot(X,Y,'bo')
plt.xlabel('Population Growth Rate (%)')
plt.ylabel('Elderly population Rate (%)')
plt.show()
```



주요 활성화 함수

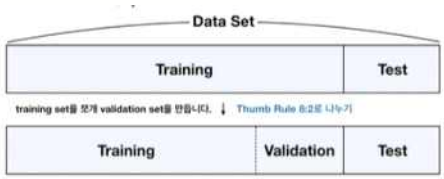


-ReLU
-Sigmoid
-Tanh

```
# 활성화 함수
import  math
def  sigmoid (x ):
    return  1  / (1  + math.exp(-x))
x = np.arange(-5 , 5 , 0.01 )
sigmoid_x = [sigmoid(z) for  z in  x]
tanh_x = [math.tanh(z) for  z in  x]
relu = [0  if  z < 0  else  z for  z in  x]
plt.figure(figsize=(8 , 6 ))
plt.axhline(0 , color='gray')
plt.axvline(0 , color='gray')
plt.plot(x, sigmoid_x, 'b--', label='sigmoid')
plt.plot(x, tanh_x, 'r--', label='tanh')
plt.plot(x, relu, 'g.', label='relu')
plt.legend()
plt.show()
```

보스톤 주택 가격 예측

1978년 보스톤 지역 주택 가격 데이터 셋
- 506개 타운의 주택 가격 중앙 값, 천 달러 단위
- 범죄율 방 수, 고속도로까지 거리 등 13가지 특성
- 학습 데이터 : 404개
- 테스트 데이터 : 102개




```
# 4.11 데이터 불러오기
from  tensorflow.keras.datasets import  boston_housing
(train_X, train_Y), (test_X, test_Y) = boston_housing.load_data()
print (train_X.shape, test_X.shape)
print (train_X[0 ])
print (train_Y[0 ])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/bost
57344/57026 [==============================] - 0s 0us/step
(404, 13) (102, 13)
[  1.23247   0.        8.14      0.        0.538     6.142    91.7
   3.9769    4.      307.       21.      396.9      18.72    ]
15.2
```

데이터 결과 값 15.2 (평균 집 가격 15200 달러)

14가지 속성중 13가지 사용

속성의 단위 등 다양한 값
 - 정규화 필요

[01] CRIM	자치시(town) 별 1인당 범죄율
[02] ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03] INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04] CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05] NOX	10ppm 당 농축 일산화질소
[06] RM	주택 1가구당 평균 방의 개수
[07] AGE	1940년 이전에 건축된 소유주택의 비율
[08] DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09] RAD	방사형 도로까지의 접근성 지수
[10] TAX	10,000 달러 당 재산세율
[11] PTRATIO	자치시(town)별 학생/교사 비율
[12] B	1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13] LSTAT	모집단의 하위계층의 비율(%)
[14] MEDV	본인 소유의 주택가격(중앙값) (단위: $1,000)

자료의 정규화
- 특성의 단위가 다름
- 정규화가 학습 효율에 좋음

정규화 방법
- 학습 데이터 : (train_X - 학습데이터평균) / 학습데이터 표준편차
- 정규 분포를 가정

- 테스트 데이터 : (test_X - 학습데이터평균) / 학습데이터 표준편차
- 테스트데이터가 정규 분포를 가정할수 없으므로

딥러닝 모델

총 4개의 층
- 출력 층은 회귀 모델, 주택 가격이므로 1

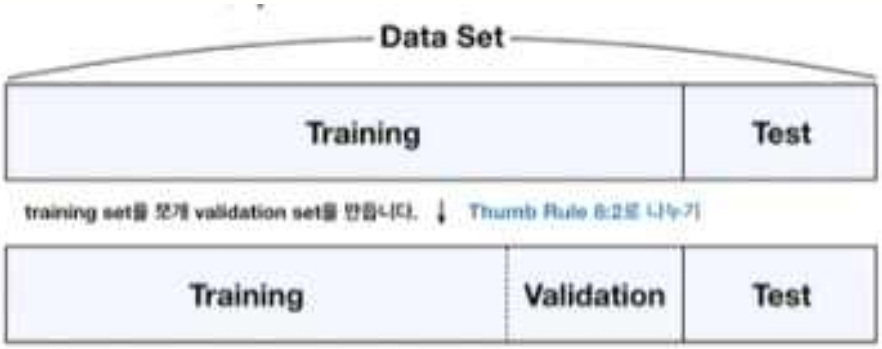
학습률 : lr = 0.07
손실 함수 : mse

```
# 4.13 Boston Housing Dataset 회귀 모델 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=52    , activation='relu', input_shape=(13
,)),
    tf.keras.layers.Dense(units=39 , activation='relu'),
    tf.keras.layers.Dense(units=26 , activation='relu'),
    tf.keras.layers.Dense(units=1 )
])
model.compile (optimizer=tf.keras.optimizers.Adam(lr=0.07 ), loss='mse')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 52)                728
_________________________________________________________________
dense_3 (Dense)              (None, 39)                2067
_________________________________________________________________
dense_4 (Dense)              (None, 26)                1040
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 27
=================================================================
Total params: 3,862
Trainable params: 3,862
Non-trainable params: 0
```

배치 사이즈와 검증 데이터

-훈련과 검증분리, 훈련 데이터 404개 중 일부를 검증데이터로 사용



validation_split:
- 검증용 데이터의 비율
validation_split = 0.25 시 75%:25%

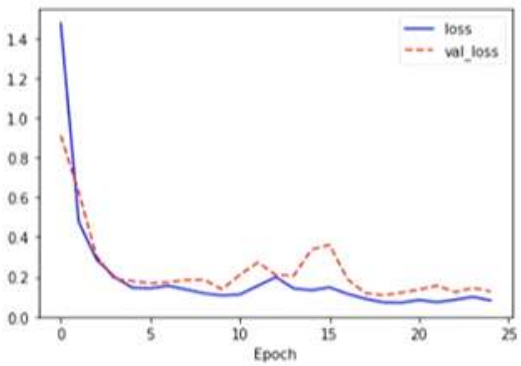
만일 .2면
- 훈련:검증 == 80%:20% 비중으로 준비

- batch_size
훈련에서 가중치와 편향의 패러미터를 수정하는 데이터 단위 수
(크게 할수록 빠르다)

- train_size
훈련 데이터 수

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

```
# 4.15 회귀 모델 학습 결과 시각화
import  matplotlib.pyplot as  plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



loss : 손실함수
일반적으로 loss는 꾸준히 감소

val_loss : 검증용 데이터 손실값
val_loss는 loss보다 높음
항상 감소하지도 않음

손실값 : 작을수록 좋음
검증 손실이 적을수록 테스트
평가의 손실도 적음

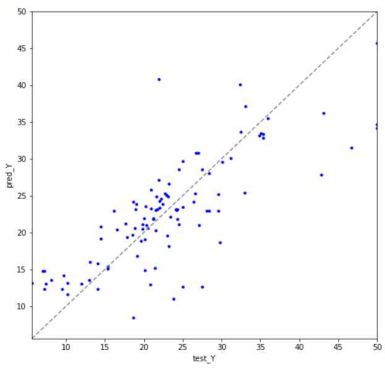
```
# 4.16 회귀 모델 평가
model.evaluate(test_X, test_Y)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 33.0656
33.06561279296875
```

예측 시각화

테스트 데이터의 예측과 실제 주택 가격 비교
- 각 점들이 점선의 대각선에 있어야 좋은 예측이다.

plt.axis(xmin,xmax,ymin,ymax))는
축의 범위를 지정한다.




```
# 4.17 실제 주택 가격과 예측 주택 가격 시각화
import  matplotlib.pyplot as  plt
pred_Y = model.predict(test_X)
plt.figure(figsize=(8 , 8 ))
plt.plot(test_Y, pred_Y, 'b.')
plt.axis([min (test_Y), max (test_Y), min (test_Y), max (test_Y)])
# y=x에 해당하는 대각선
plt.plot([min (test_Y), max (test_Y)], [min (test_Y), max (test_Y)], ls="--", c=".5")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')
plt.show()
```

자동으로 학습 중단

검증 손실이 적을수록 테스트 평가의 손실도 적음
검증 데이터에 대한 성적이 좋도록 유도
- 과적합에 의해 검증 손실이 증가하면 학습을 중단 되도록 지정
- 함수 callbacks 사용
-
검증데이터가 상승하면 강제로 멈추게 할 수 있다.

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

```
Epoch 1/25
10/10 [==============================] - 0s 6ms/step - loss: 18.8581 - val_loss: 23.9813
Epoch 2/25
10/10 [==============================] - 0s 3ms/step - loss: 18.2040 - val_loss: 20.2056
Epoch 3/25
10/10 [==============================] - 0s 3ms/step - loss: 18.0396 - val_loss: 19.8429
Epoch 4/25
10/10 [==============================] - 0s 3ms/step - loss: 16.1440 - val_loss: 27.2558
Epoch 5/25
10/10 [==============================] - 0s 3ms/step - loss: 21.1603 - val_loss: 25.0639
Epoch 6/25
10/10 [==============================] - 0s 3ms/step - loss: 25.6630 - val_loss: 20.7993
```

3번째 이후 4, 5, 6 모두 기록이 커졌으므로 학습 중지하였다.

★ 일찍 멈춤 기능

- tf.keras.callbacks.EarlyStopping

monitor='val_loss' - 지켜볼 기준 값이 검증 손실

patience=3
- 3회의 실행동안 최고 기록을 갱신하지 못하면 (더 낮아지지 않으면) 학습을 멈춘다.

회귀(regression)
- 가격이나 확률 같이 연속된 출력 값을 예측하는 것이 목적
분류(classification)
- 여러 개의 클래스 중 하나의 클래스를 선택하는 것이 목적
ex) 사진에 사과, 오렌지가 포함될 시 어떤 과일인지 인식하는 것

10주차

자동차 연비(auto mpg: mile per gallon) 데이터로 회귀분석

자동차 연비를 예측하는 모델
- Auto MPG 데이터 셋을 사용
- 1970년대 후반과 1980년대 초반의 데이터
- 이 기간에 출시된 자동차 정보를 모델에 제공

Auto MPG 데이터 셋

- **판다스를 사용하여 데이터를 읽기**

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

특정 문자가 있는 행은 주석으로 간주하고 읽지 않고 건너뜀

dataset_path에서 값을 받아온다.
cal_names : 열 데이터에 이름을 지정 (8개)
names : 열 이름 지정
na_values : not available 값이 없거나 잘못 들어있는 값
comment : 주석 처리
sep = “ ”, 스페이스로 컬럼 구분
skipinitialspace=True 공백이 있으면 뛰어넘어라
dataset : 판다스의 dataframe이라는 타입 **(중요)**
tail(10) : 뒤쪽 10개를 보여준다. 값 없으면 마지막 5개
head(10) : 앞쪽 10개 보여준다.

	MPG	Cylinders	Displacement	Horsepower	weight	Acceleration	Model Year	Origin
388	26.0	4	156.0	92.0	2585.0	14.5	82	1
389	22.0	6	232.0	112.0	2835.0	14.7	82	1
390	32.0	4	144.0	96.0	2665.0	13.9	82	3
391	36.0	4	135.0	84.0	2370.0	13.0	82	1
392	27.0	4	151.0	90.0	2950.0	17.3	82	1
393	27.0	4	140.0	86.0	2790.0	15.6	82	1
394	44.0	4	97.0	52.0	2130.0	24.6	82	2
395	32.0	4	135.0	84.0	2295.0	11.6	82	1
396	28.0	4	120.0	79.0	2625.0	18.6	82	1
397	31.0	4	119.0	82.0	2720.0	19.4	82	1

```
dataset.shape
(398, 8)
```

2차원 행 398개 열 8개

```
# 데이터 정제, 비어있는 열의 행의 수 알아내기
dataset.isna().sum ()
```

```
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
weight          0
Acceleration    0
Model Year      0
Origin          0
dtype: int64
```

빠져있거나 잘못된 데이터 찾기
Horsepower(마력) 6개 빠져있음

```
# 비어 있는 열이 하나라도 있는 행을 제거
dataset = dataset.dropna()
dataset.shape
(392, 8)
```

2차원 행 392개 열 8개 (비어있는 6개 제거)

```
# 열 'Origin'을 빼내 origin에 저장
origin = dataset.pop('Origin')
origin
```

```
0      1
1      1
2      1
3      1
4      1
      ..
393    1
394    2
395    1
396    1
397    1
Name: Origin, Length: 392, dtype: int64
```

pop을 하여 Origin의 series를 origin로 옮겨서 기존 테이블에서 분리

	MPG	Cylinders	Displacement	Horsepower	weight	Acceleration	Model Year
0	18.0	8	307.0	130.0	3504.0	12.0	70
1	15.0	8	350.0	165.0	3693.0	11.5	70
2	18.0	8	318.0	150.0	3436.0	11.0	70
3	16.0	8	304.0	150.0	3433.0	12.0	70
4	17.0	8	302.0	140.0	3449.0	10.5	70
...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790.0	15.6	82
394	44.0	4	97.0	52.0	2130.0	24.6	82
395	32.0	4	135.0	84.0	2295.0	11.6	82
396	28.0	4	120.0	79.0	2625.0	18.6	82
397	31.0	4	119.0	82.0	2720.0	19.4	82

392 rows × 7 columns

Origin 열이 사라진 것을 확인할 수 있다.

데이터셋을 훈련 세트와 테스트 세트로 분할

80:20으로
- 테스트 세트는 모델을 최종적으로 평가할 때 사용

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

```
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)

print(train_dataset.shape, test_dataset.shape)
```

```
(314, 7) (78, 7)
```

기존의 392개의 행을 80:20으로 나누어 훈련 데이터 314와
테스트데이터 78개로 구분된다.

```
# "Origin" 열은 수치형이 아니고 범주형이므로 원-핫 인코딩으로 변환
dataset['USA'] = (origin == 1 )*1.0
dataset['Europe'] = (origin == 2 )*1.0
dataset['Japan'] = (origin == 3 )*1.0
dataset.tail()
```

origin이 1이면 USA에 1.0이 2면 Europe에 들어가고 3이면 Japan에 들어간다

```
(314, 10) (78, 10)
```

그 후 열이 10개로 바뀌었다.

```
#전반적인 통계도 확인
train_stats = train_dataset.describe()
print (train_stats)
```

	MPG	Cylinders	Displacement	...	USA	Europe	Japan
count	314.000000	314.000000	314.000000	...	314.000000	314.000000	314.000000
mean	23.310510	5.477707	195.318471	...	0.624204	0.178344	0.197452
std	7.728652	1.699788	104.331589	...	0.485101	0.383413	0.398712
min	10.000000	3.000000	68.000000	...	0.000000	0.000000	0.000000
25%	17.000000	4.000000	105.500000	...	0.000000	0.000000	0.000000
50%	22.000000	4.000000	151.000000	...	1.000000	0.000000	0.000000
75%	28.950000	8.000000	265.750000	...	1.000000	0.000000	0.000000
max	46.600000	8.000000	455.000000	...	1.000000	1.000000	1.000000

10주차

정답인 MPG를 추출

```
train_labels = train_dataset.pop('MPG')   - 훈련정답
test_labels = test_dataset.pop('MPG')     - 테스트 정답
```

★데이터 정규화
특성의 스케일과 범위가 다르면 정규화 하는 것을 권장
-의도적으로 훈련 세트만 사용하여 통계치를 생성
테스트 세트를 정규화할 때에도 훈련 데이터의 평균과 표준편차 사용
- 테스트 세트를 모델이 훈련에 사용했던 것과 동일한 분포로 투영하기 위해

모델을 구성

두 개의 완전 연결 은닉층으로 Sequential 모델
- 출력 층은 하나의 연속적인 값을 반환
- 나중에 두 번째 모델을 만들기 쉽도록 build_model 함수로 모델 구성 단계를 감쌈

```
def  build_model ():
    model = keras.Sequential([
            layers.Dense(64          , activation='relu', input_shape=[len
(train_dataset.keys())]),
            layers.Dense(64 , activation='relu'),
            layers.Dense(1 )
    ])
    optimizer = tf.keras.optimizers.RMSprop(0.001 )
    model.compile (loss='mse', optimizer=optimizer, metrics=['mae', 'mse'])
    return  model
model = build_model()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 64)                640
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 4,865
Trainable params: 4,865
Non-trainable params: 0
_________________________________________________________________
```

(9 + 1) * 64 = 640
(64 + 1) * 64 = 4160
(64 + 1) * 1 = 65

모델 훈련

1000번의 에포크 동안 훈련
- 훈련 정확도와 검증 정확도는 history 객체에 기록

```
# 에포크 중간 중간에 점(.)을 출력해 훈련 진행 과정을 표시
class  PrintDot (keras .callbacks .Callback ):
    def  on_epoch_end (self , epoch , logs ):
        if  epoch % 100  == 0 : print ('')
        print ('.', end='')
EPOCHS = 1000
history = model.fit(normed_train_data, train_labels, epochs = EPOCHS,
                    validation_split = 0.2 , verbose=0 , callbacks = [PrintDot()])
```

20% 검증용, 80% 실제 훈련용
callbacks 모델을 훈련 시킬 때 반영을 지정할 수 있다.
PrintDot()으로 훈련하는 과정에 점을 찍는다.

콜백
학습 과정에 한 에폭마다 적용할 함수의 세트\

★ EarlyStopping 콜백

- model.fit 메서드를 수정하여 검증 점수가 향상되지 않으면 자동으로 훈련을 멈추도록 한다.

```
model = build_model()

# patience 매개변수는 성능 향상을 체크할 에포크 횟수입니다
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
history = model.fit(normed_train_data, train_labels, epochs=EPOCHS,
                    validation_split = 0.2, verbose=0, callbacks=[early_stop, PrintDot()])

plot_history(history)
```

keras.callback.EarlyStopping(monitor='val_loss', patience=10)

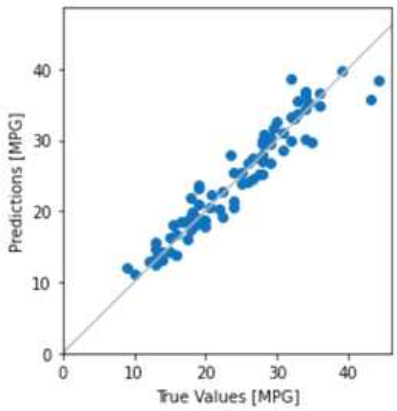
옵션 monitor, patience
- 손실 val_loss가 10회 초과해 감소하지 않으면 중단한다.

테스트 세트에 있는 샘플을 사용해 MPG 값을 예측

예측 (1)

```
test_predictions = model.predict(normed_test_data).flatten()
plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal') # 각 축의 범위와 축의 스케일을 동일하게 설정
plt.axis('square') # 각 축의 범위를 xmax - xmin = ymax - ymin 이 되도록 설정
print (plt.xlim()[1 ])
plt.xlim([0 , plt.xlim()[1 ]])
print (plt.ylim()[1 ])
plt.ylim([0 , plt.ylim()[1 ]])
_=plt.plot([-100 , 100 ], [-100 , 100 ], c='.7')
```

```
46.065
48.60996788024902
```



소감

포트폴리오를 작성하며 강의 다시 돌려 보았습니다. 처음 봤을 때 이해하지 못하고 넘어간 부분들이 나와 복습하며 정리하였습니다. 모르는 것들을 정리하면서 뿌듯함을 느낌과 동시에 바로 정리하지 않았던 점을 반성하게 되었습니다.
인공지능 응용프로그래밍의 포트폴리오를 정리하다 보니 다른 수업들도 다시금 정리하고 싶은 다짐이 들었습니다. 이 다짐을 통해 전공지식을 다시 한번 되짚어 보며 자신감을 키워나간다면 현재 진행 중인 취업 준비에 있어서도 큰 도움이 될 수 있을 것 같습니다. 감사합니다.