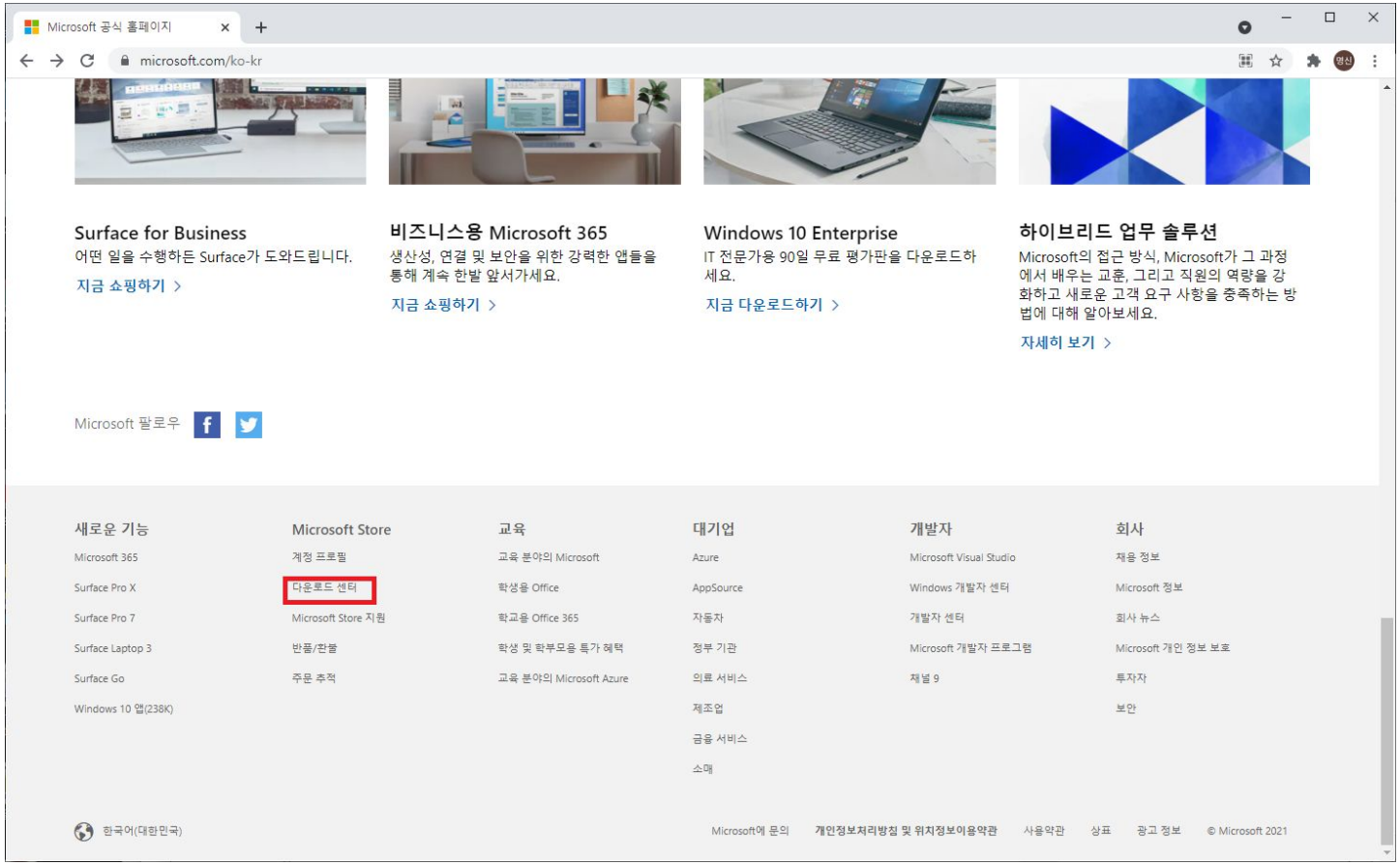
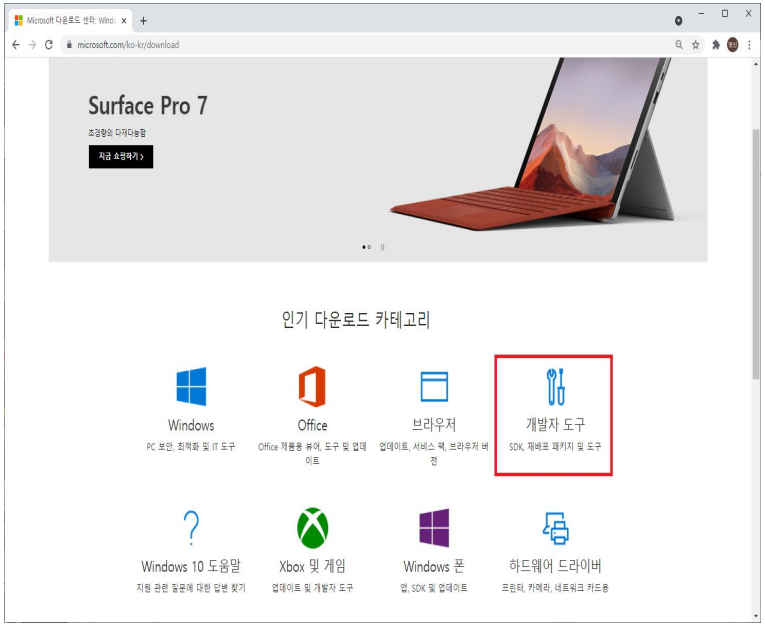
Tensorflow

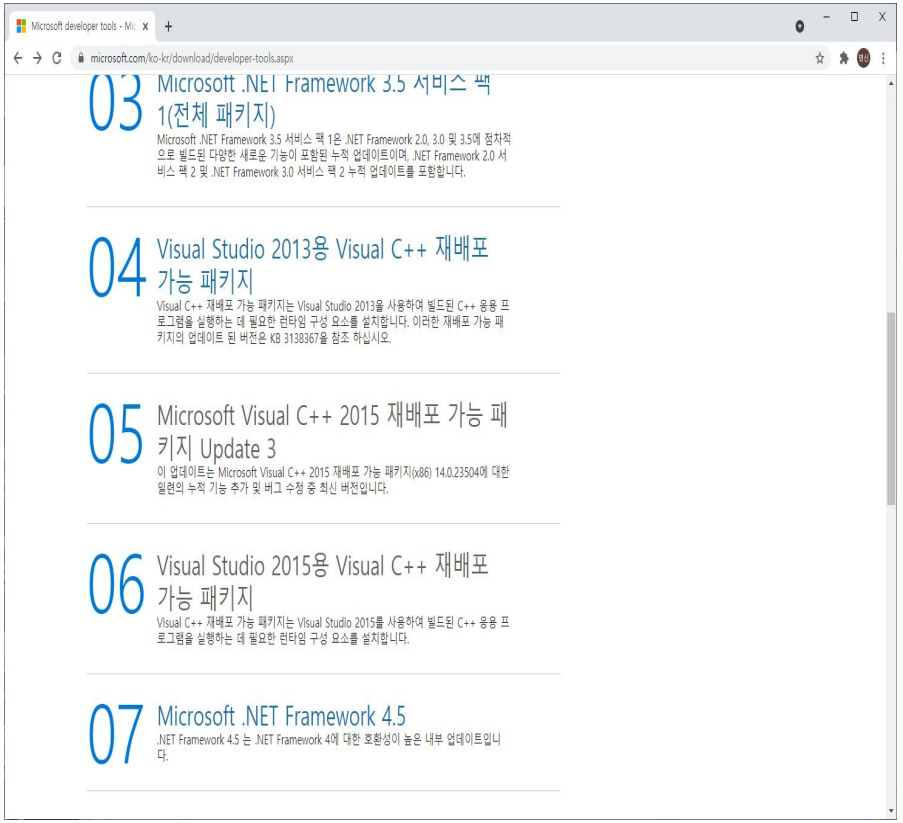
Tensorflow 설치

- https://www.microsoft.com/ko-kr

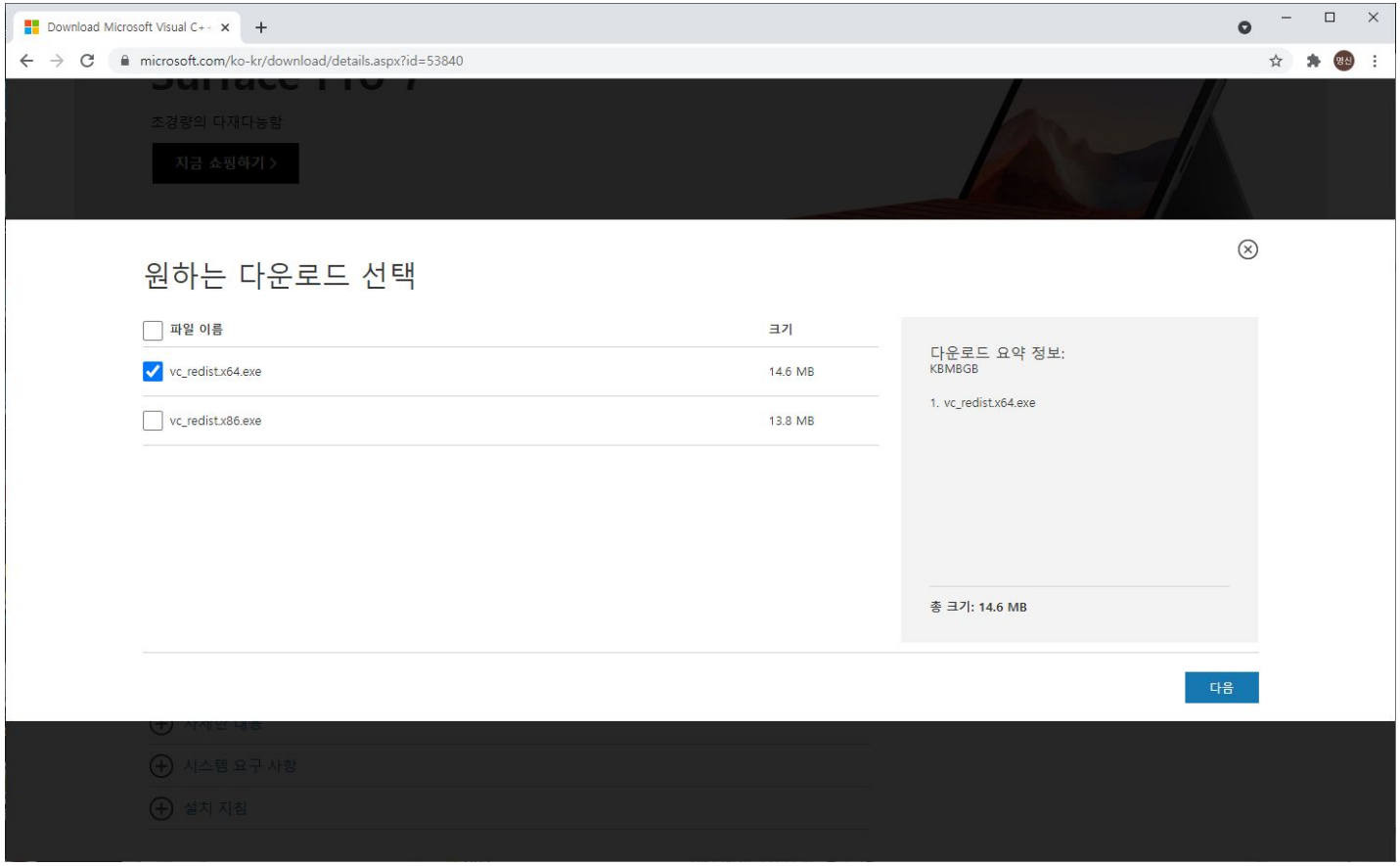


Tensorflow 설치

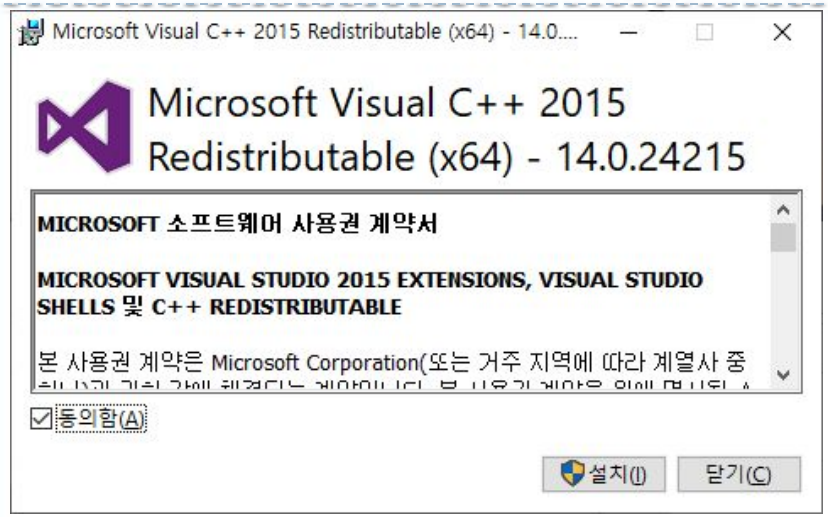


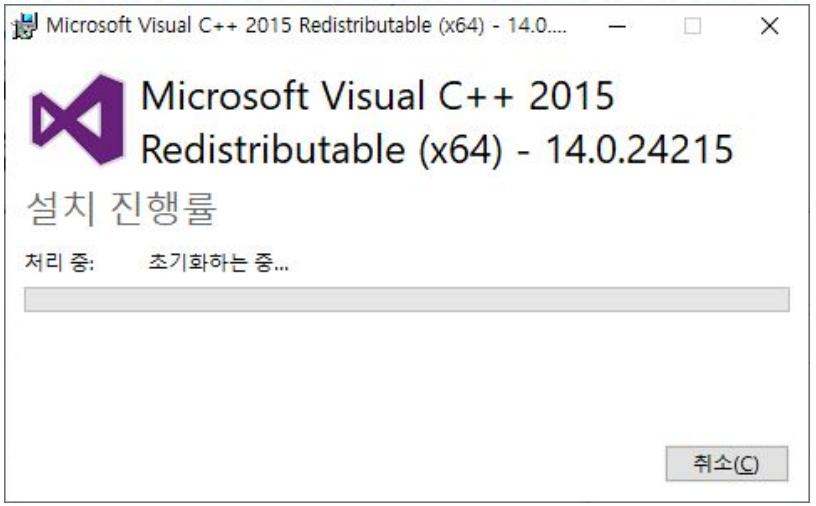


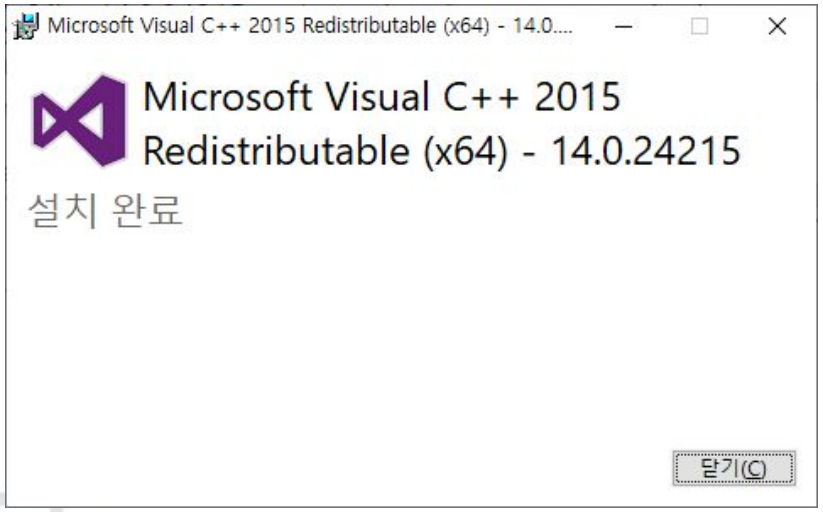
Tensorflow 설치



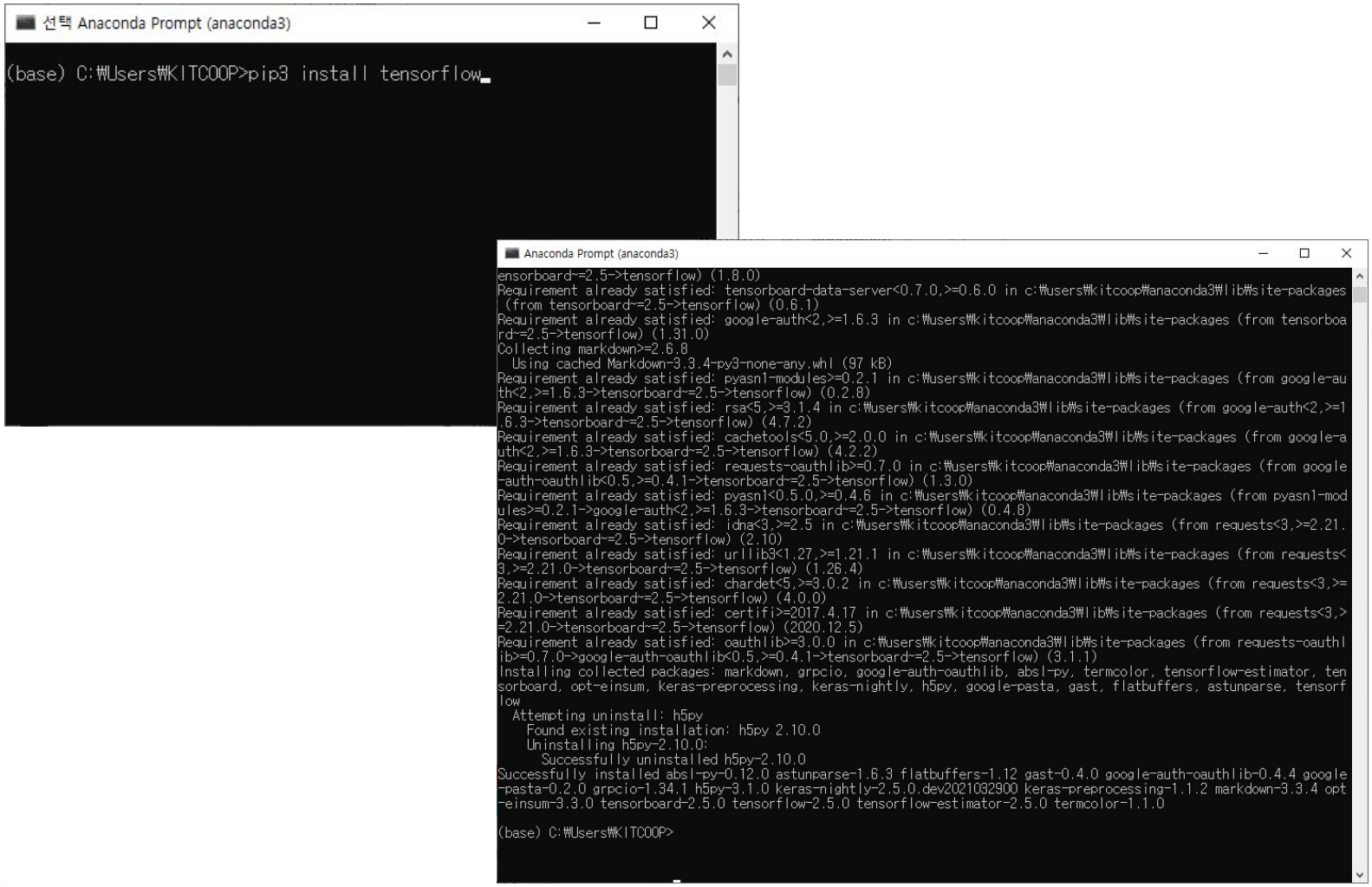
Tensorflow 설치







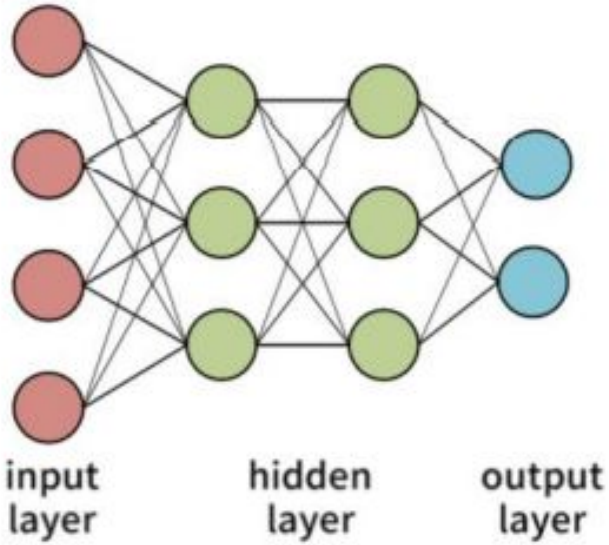
Tensorflow 설치



Deep Learning

Deep Neural Network을 이용한 Machine Learning 방법

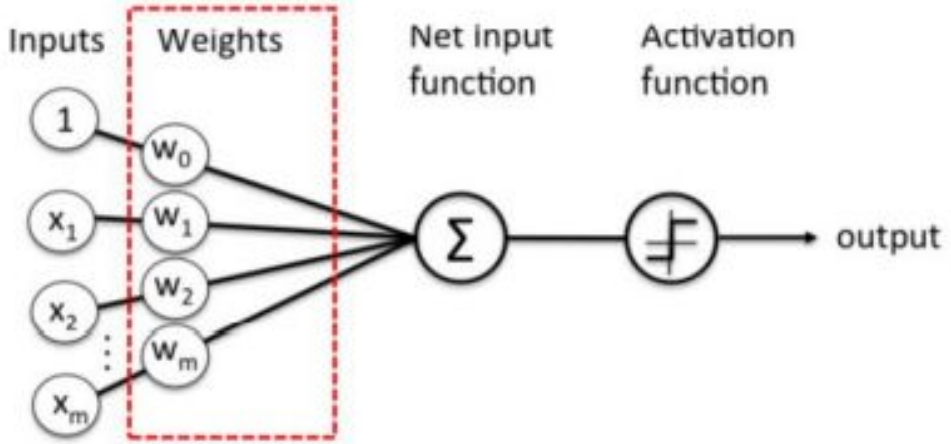
Deep Neural Network(DNN) : Hidden layer 여러 개의 은닉층들로 이루어진 인공신경망이다. 심층 신경망은 일반적인 인공신경망과 마찬가지로 복잡한 비선형
관계들을 모델링 할 수 있다



Weight 값의 초기화

원하는 Weight를 어떻게 찾을 수 있을까?
아무것도 모르는 상태에서 Weight의 초기값은 어떻게 설정할 것인가?
처음 시작은 Random한 값으로 시작한다.

Random
Initialization



Cost Function (=Loss Function)

Neural Network가 실제 값과 얼마나 비슷한지에 대한 척도

(뉴럴넷 예측값 – 정답값)^2

2가지 광고료 조합에 대하여 Neural Network은

예측한 판매량 : (100, 80)
실 제 판매량 : (105, 78)

→ L = (105-100)2 + (78-80)2 = 29

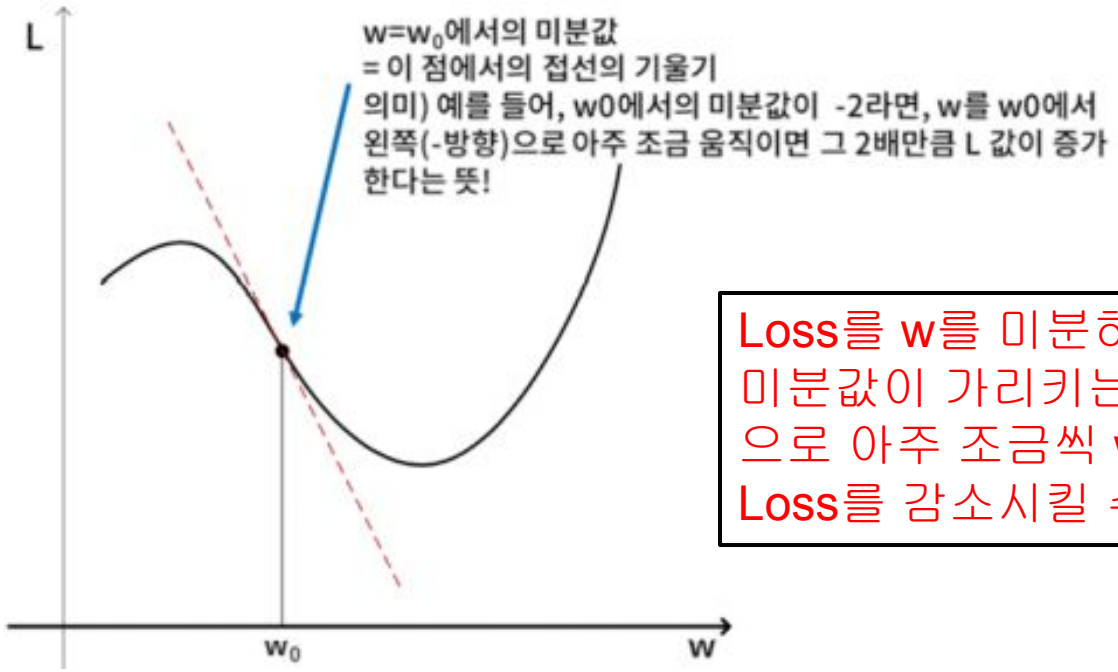
예측 값의 정확도를 높이기 위해 필요한 것은
Cost Function의 값이 줄어들도록 weight값들을 조금씩 바꾸는 것

Weight를 어떻게 바꾸어야 Cost Function값이 줄어들까?

→ 미분

Cost Function에서 최소값을 구하려면

미분이 갖는 의미 : L의 값이 가장 낮은 위치를 찾을 수 있다!



Loss를 w를 미분하고
미분값이 가리키는 방향의 반대방향으로 아주 조금씩 w를 바꿔나가면
Loss를 감소시킬 수 있다.

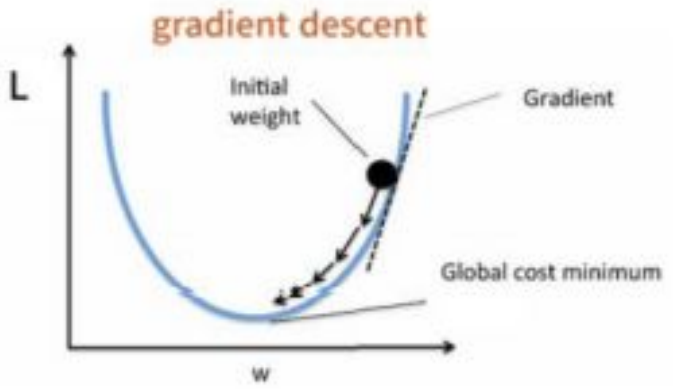
Gradient Descent (경사 하강법)

1차 근사값 발견용 최적화 알고리즘이다. 기본 아이디어는 함수의 기울기(경사)를 구하여 기울기가 낮은 쪽으로 계속 이동시켜서 극 값에 이를 때까지 반복한다.

$$w_{new} = w_{old} - \eta \nabla_w L$$

$$\text{Weight update} = w_{new} - w_{old} = -\eta \nabla_w L$$

- $-$: Loss를 감소시키는 방향 (Descent)
- η : 아주 조금씩 이동 (Learning Rate)
- $\nabla_w L$: 미분값 (Gradient)



Gradient Descent (경사 하강법)

1. 모든 Training Data에 대해
Neural Network의 출력과 실제 Lable을 비교하여 각각의 Loss(Cost)값을 계산한다.
2. 이 Loss를 Weight로 미분한 다음
그 미분값(Gradient)이 가리키는 방향의 반대 방향으로 weight값을 아주 조금씩 바꿔 나간다.

문제점 : 데이터의 양이 많은 경우 학습 속도가 느리다!

Batch/Stochastic/Mini-batch Gradient Descent

모든 training data에 대하여 Loss를 계산하면 시간이 너무 오래 걸림. 개선 방법

Batch Gradient Descent
전체 data중에 일부 data만 골라서, 이 data들이 전체 training data를 대표한다고 판단하고 이 data들로만 Loss를 계산하고, Gradient Descent를 행한다.

Stochastic Gradient Descent (랜덤하기 확률적으로 Data를 취한다.)
Data를 1개만 선택하고, 1개의 data에 대한 Loss를 이용하여 Gradient Descent를 행한다.

Mini-batch Gradient Descent
Batch/Stochastic의 중간 형태로 data를 n개 뽑고 그 n개의 data에 대한 Loss를 계산하여다 더한 뒤 이를 이용하여 Gradient Descent를 행한다.

Backpropagation(오차역전파법)

오차역전파법을 이해하는 방법은 2가지가 있다.
1. 수식을 통해 이해하는 방법
2. 계산그래프를 통하는 방법

수식을 사용한 설명은 정확하고 간결하다. 그러나 수많은 수식에 당황하는 일도 있다.
반면, 계산 그래프를 이용하면 시각적으로 이해하는데 도움이 된다.

계산 그래프는 계산 과정을 그래프로 나타낸 것이다.
그래프는 우리가 자주 사용하는 자료구조로 복수의 노드와 에지로 구성된다.
(노드 사이의 직선을 에지라고 볼 수 있다.)

수학으로 이해하는 오차역전파법

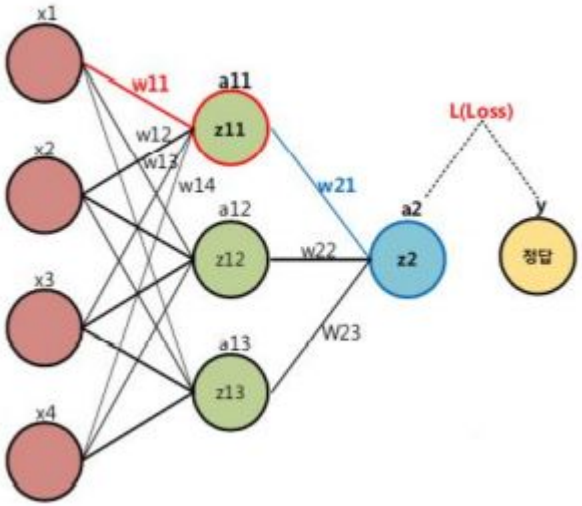
$$z11 = x1 \cdot w11 + x2 \cdot w12 + x3 \cdot w13 + x4 \cdot w14$$

$$a11 = \sigma(z11) = \frac{1}{1 + e^{-z11}}$$

$$z2 = a11 \cdot w21 + a12 \cdot w22 + a13 \cdot w23$$

$$a2 = z2$$

$$L = (y - a2)^2$$



수학으로 이해하는 오차역전파법

$$z11 = x1 \cdot w11 + x2 \cdot w12 + x3 \cdot w13 + x4 \cdot w14$$
$$a11 = \sigma(z11) = \frac{1}{1 + e^{-z11}}$$
$$z2 = a11 \cdot w21 + a12 \cdot w22 + a13 \cdot w23$$
$$a2 = z2$$
$$L = (y - a2)^2$$

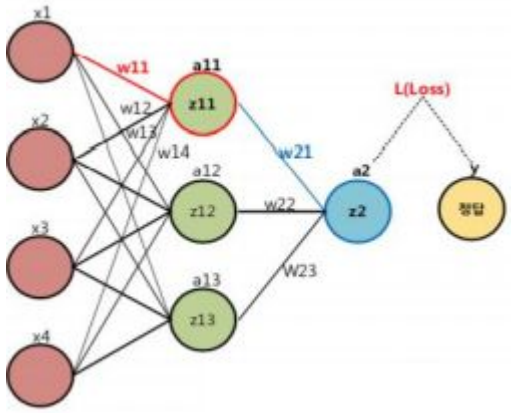
Loss부터 거꾸로 한 단계씩 미분을 해봅시다

$$\partial L/\partial a2 = 2(y - a2)$$
$$\partial a2/\partial z2 = 1$$
$$\partial z2/\partial a11 = w21$$
$$\partial a11/\partial z11 = \sigma(z11) \cdot \big(1 - \sigma(z11)\big)$$
$$\partial z11/\partial w11 = x1$$

이 미분들을 전부 각각 곱하면(chain rule),

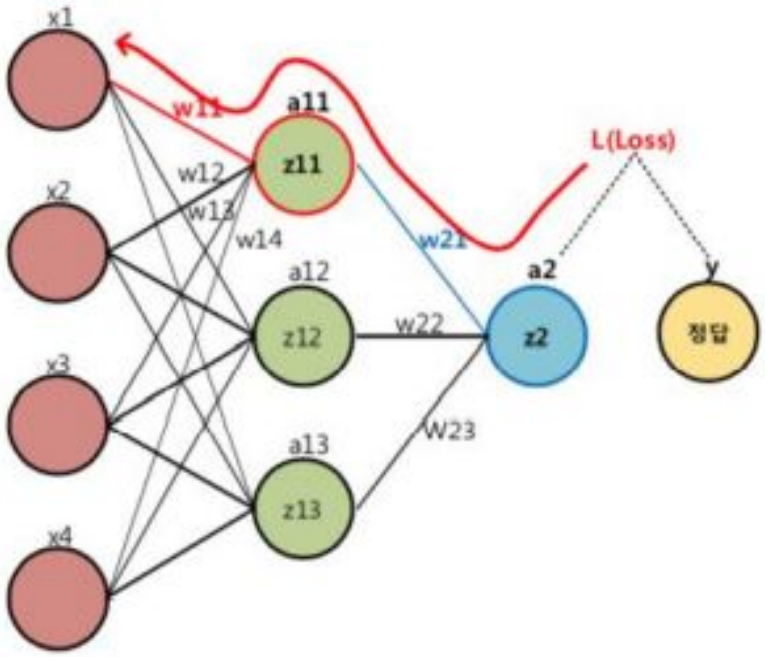
$$\frac{\partial L}{\partial a2} \cdot \frac{\partial a2}{\partial z2} \cdot \frac{\partial z2}{\partial a11} \cdot \frac{\partial a11}{\partial z11} \cdot \frac{\partial z11}{\partial w11} = \frac{\partial L}{\partial w11}$$

우리가 구하려고 했던 미분값



수학으로 이해하는 오차역전파법

Loss로부터 거꾸로 한 단계씩 미분 값을 구하고 이 값들을 chain rule에 의하여 곱해가면서 weight 에 대한 gradient를 구하는 방법



Overfitting

딥러닝의 목적

Deep Neural Network를 사용하여, 우리가 가진 데이터로 학습을 해서(Training), 학습에 사용하지 않았던 미지의 데이터가 들어왔을 때, 예측(Inference)하는 시스템을 설계하는 것이다.

학습을 잘 하는 것 = Training Error를 줄이는 것 미지의 데이터 대해 예측을 잘하자 = Test Error를 줄이는 것

Training Error를 줄이면 Test Error도 줄어들까?

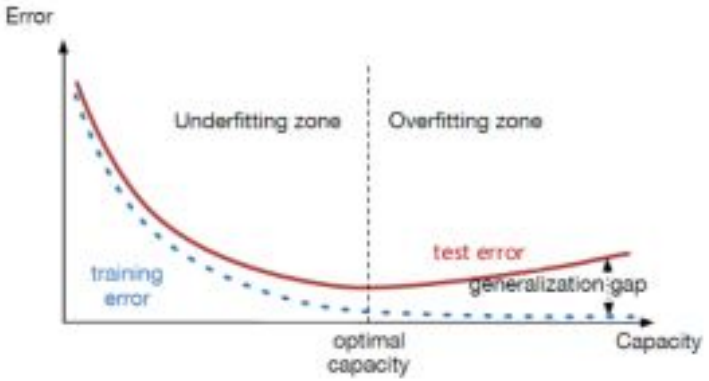
Underfitting과 Overfitting

Model capacity가 너무 작으면,
- 아무리 학습을 해도 성능이 안나옴

→underfitting

Model capacity가 너무 크면,
- Generalization gap이 커짐

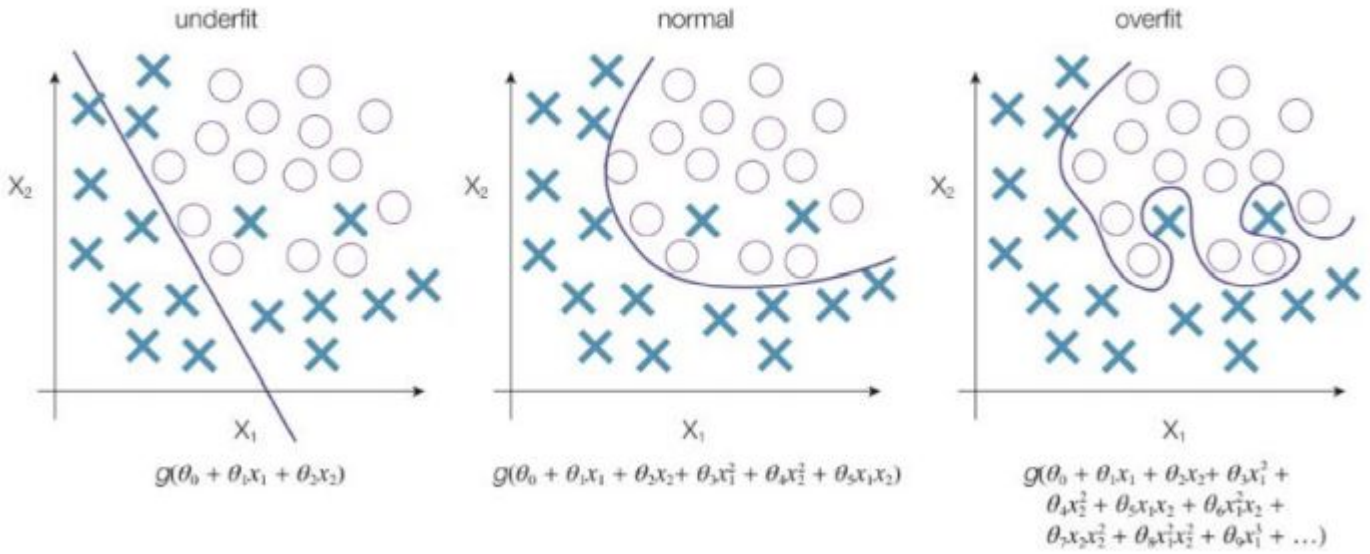
→overfitting



Deep Learning에 사용하는 model은 capacity가 대부분 크기 때문에 overfitting에 취약함

Test Error = Training error + generalization gap

Overfitting



Overfitting

열심히 Neural Network에게 고양이 를 가르쳤더 니

색깔이 이상해 서
고양이 아 님

뚱뚱해서
고양이 아 님

귀가 쳐져 서
고양이 아 님

내가 아는 지식의 세상의 전부 인것처럼 융통성이 없어지는 것

Overfitting을 방지하는 방법

1. 데이터량을 늘린다.

2. **Regularization 방법을 사용 한다.**
   L1 / L2 Regularization(Weight Decay)
   Dropout
   Batch Normalization

텐서플로의 기본 이해

최근에 가장 선호하는 딥러닝 프레임워크

- Tensorflow 2.0 (텐서플로 + 케라스)

텐서플로의 특징

1. 전세계적으로 활발한 커뮤니티
2. 텐서보드를 이용한 편리한 시각화
3. 단일 데스크톱, 대량의 서버 클러스터, 모바일 디바이스까지 지원하는 광범위한 이식성
4. Keras와 같은 High Level API를 혼용해서 사용가능

텐서플로 기초

- 구글 코랩 운영체제 확인

```
import platform
print(platform.platform())
```

```
Linux-4.19.112+-x86_64-with-Ubuntu-18.04-bionic
```

```
# 3.5 텐서플로우 불러오기, 버전 확인
import tensorflow as tf
print(tf.__version__)
```

```
2.4.1
```

텐서플로 기초

- 랜덤한 수 생성

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

- 랜덤한 수 여러 개 얻기 : 균일 분포

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

```
tf.Tensor([0.4000026 0.02589869 0.97795093 0.37491727], shape=(4,), dtype=float32)
```

- 랜덤함 수 여러 개 얻기 : 정규 분포

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

```
tf.Tensor([-0.9041287 -1.4643867 0.05388802 0.16180076], shape=(4,), dtype=float32)
```

텐서플로 기초

- 경사 하강법을 이용한 뉴런 학습

```
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
x = 1
y = 0
w = tf.random.normal([1],0,1)
output = sigmoid(x * w)
print(output)
for i in range(1000):
    output = sigmoid(x * w)
    error = y - output
    w = w + x * 0.1 * error
if i % 100 == 99:
        print(i, error, output)
```

텐서플로 기초

- X=0,Y=1을 얻는 뉴런 학습에 편향을 더함

```
x = 0
y = 1
w = tf.random.normal([1],0,1)
b = tf.random.normal([1],0,1)

for i in range(1000):
    output = sigmoid(x * w + 1 * b)
    error = y - output
    w = w + x * 0.1 * error
    b = b + 1 * 0.1 * error

    if i % 100 == 99:
        print(i, error, output)
```

텐서플로 기초

- 신경망 네트워크 : AND

```
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [0], [0], [0]])
w = tf.random.normal([2],0,1)
b = tf.random.normal([1],0,1)
b_x = 1
for i in range(2000):
    error_sum = 0
    for j in range(4):
        output =
sigmoid(np.sum(x[j]*w)+b_x*b)
        error = y[j][0] - output
        w = w + x[j] * 0.1 * error
        b = b + b_x * 0.1 * error
        error_sum += error
if i % 200 == 199:
        print(i, error_sum)
```

텐서플로 기초

- 신경망 네트워크 : AND 평가

```
for i in range(4):
    print('X:', x[i], 'Y:', y[i], 'Output:', sigmoid(np.sum(x[i]*w)+b))
```

- 결과

X: [1 1] Y: [1] Output: 0.9644071525096253
X: [1 0] Y: [0] Output: 0.02521980267807997 4
X: [0 1] Y: [0] Output: 0.025297886216117057
X: [0 0] Y: [0] Output: 2.4782131349425264e-05

텐서플로 기초

- 신경망 네트워크 : OR

```
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [1], [1], [0]])
w = tf.random.normal([2],0,1)
b = tf.random.normal([1],0,1)
b_x = 1
for i in range(2000):
    error_sum = 0
    for j in range(4):
        output =
sigmoid(np.sum(x[j]*w)+b_x*b)
        error = y[j][0] - output
        w = w + x[j] * 0.1 * error
        b = b + b_x * 0.1 * error
        error_sum += error
if i % 200 == 199:
        print(i, error_sum)
```

텐서플로 기초

- 신경망 네트워크 : OR 평가

```
for i in range(4):
    print('X:', x[i], 'Y:', y[i], 'Output:', sigmoid(np.sum(x[i]*w)+b))
```

- 결과

X: [1 1] Y: [1] Output: 0.9999971214271413
X: [1 0] Y: [1] Output: 0.9896895055860089
X: [0 1] Y: [1] Output: 0.9896669235974469
X: [0 0] Y: [0] Output: 0.02578182732149576 4

텐서플로 기초

- 신경망 네트워크 : XOR

```
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])
w = tf.random.normal([2],0,1)
b = tf.random.normal([1],0,1)
b_x = 1
for i in range(2000):
    error_sum = 0
    for j in range(4):
        output =
sigmoid(np.sum(x[j]*w)+b_x*b)
        error = y[j][0] - output
        w = w + x[j] * 0.1 * error
        b = b + b_x * 0.1 * error
        error_sum += error
if i % 200 == 199:
        print(i, error_sum)
```

텐서플로 기초

- 신경망 네트워크 : XOR 평가

```
for i in range(4):
    print('X:', x[i], 'Y:', y[i], 'Output:', sigmoid(np.sum(x[i]*w)+b))
```

- 결과

X: [1 1] Y: [0] Output: 0.5128176286712095
X: [1 0] Y: [1] Output: 0.512817630532630 5
X: [0 1] Y: [1] Output: 0.4999999990686774
X: [0 0] Y: [0] Output: 0.5000000009313226

텐서플로 기초

- tf.keras 를 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.optimizer
s.SGD(lr=0.3), loss='mse')

model.summary()
```

텐서플로 기초

- tf.keras 를 이용한 XOR 네트워크 학습

```
history = model.fit(x, y, epochs=2000, batch_size=1)
```

- tf.keras 를 이용한 XOR 네트워크 평가

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

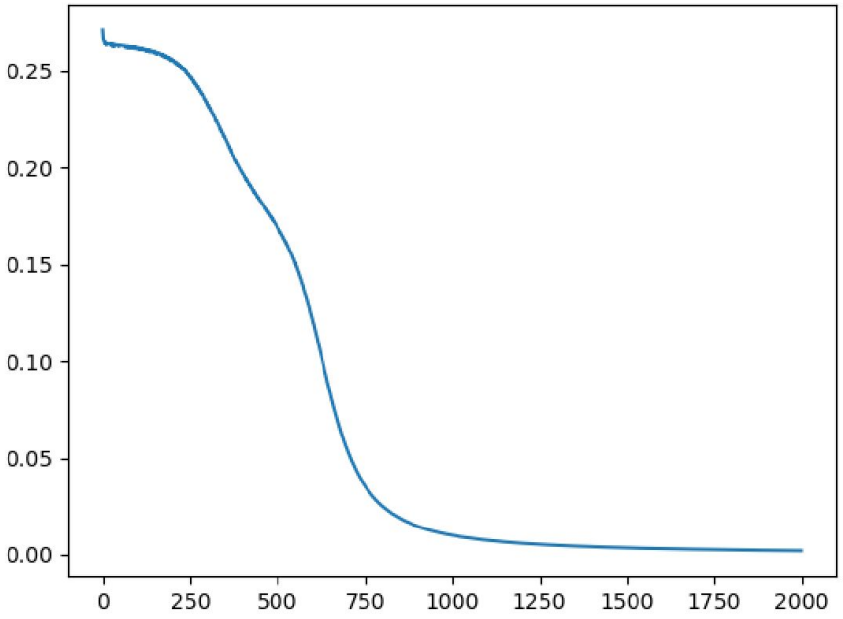
- 결과

```
array([[0.5591498 ],
       [0.548841  ],
       [0.57169133],
       [0.56192833]], dtype=float32)
```

텐서플로 기초

- 2-레이어 XOR 네트워크의 loss 변화를 선 그래프로 표시

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```



선형 회귀 (Linear Regression)

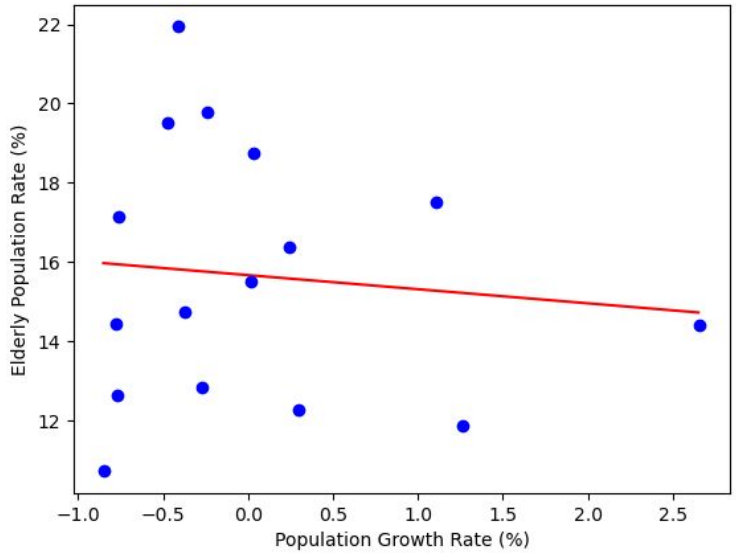
- 인구증가율과 고령인구비율 시각화:최소제곱법

```
import numpy as np
import matplotlib.pyplot as plt
X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85, -0.41, -0.27, 0.02, -0.76,
2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21.94, 12.83,
15.51, 17.14, 14.42]

# X, Y의 평균을 구합니다.
x_bar = sum(X) / len(X)
y_bar = sum(Y) / len(Y)

# 최소제곱법으로 a, b를 구합니다.
a = sum([(y - y_bar) * (x - x_bar) for y, x in list(zip(Y, X))])
a /= sum([(x - x_bar) ** 2 for x in X])
b = y_bar - a * x_bar
print('a:', a, 'b:', b)

# 그래프를 그리기 위해 회귀선의 x, y 데이터를 구합니다.
line_x = np.arange(min(X), max(X), 0.01)
line_y = a * line_x + b
```

선형 회귀 (Linear Regression)

- 인구증가율과 고령인구비율 시각화 :최소제곱법

```
# 붉은색 실선으로 회귀선을 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()
```



선형 회귀 (Linear Regression)

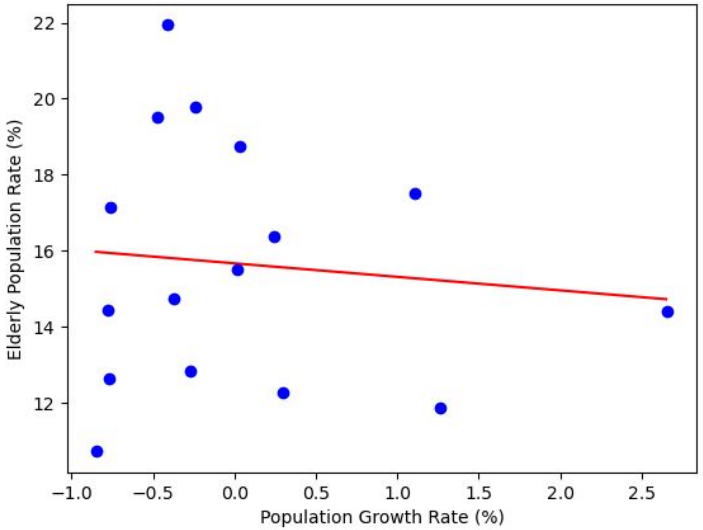
- 인구증가율과 고령인구비율 시각화:텐서플로이용

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85, -0.41, -0.27, 0.02, -0.76,
2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21.94, 12.83,
15.51, 17.14, 14.42]

# a와 b를 랜덤한 값으로 초기화합니다.
a = tf.Variable(random.random())
b = tf.Variable(random.random())

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

선형 회귀 (Linear Regression)

- 인구증가율과 고령인구비율 시각화 :텐서플로이용

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

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

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()
```



다항 회귀

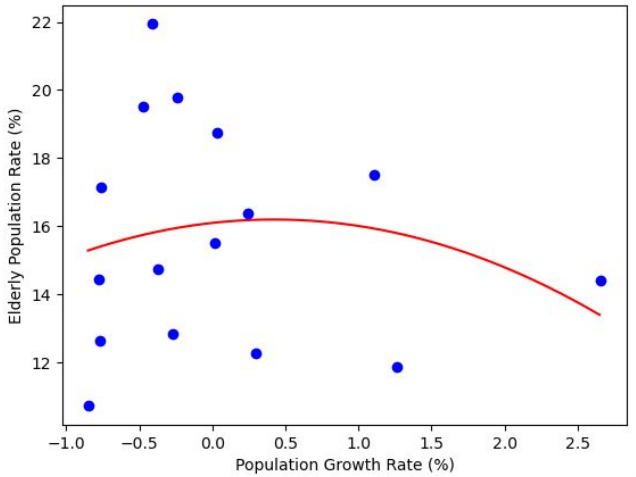
- 텐서플로우를 이용해서 2차 함수 회귀선 구하기

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85, -0.41, -0.27, 0.02,
-0.76, 2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21.94,
12.83, 15.51, 17.14, 14.42]

# a, b, c를 랜덤한 값으로 초기화합니다.
a = tf.Variable(random.random())
b = tf.Variable(random.random())
c = tf.Variable(random.random())

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

다항 회귀

- 텐서플로우를 이용해서 2차 함수 회귀선 구하기

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

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

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

# 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()
```



다항 회귀

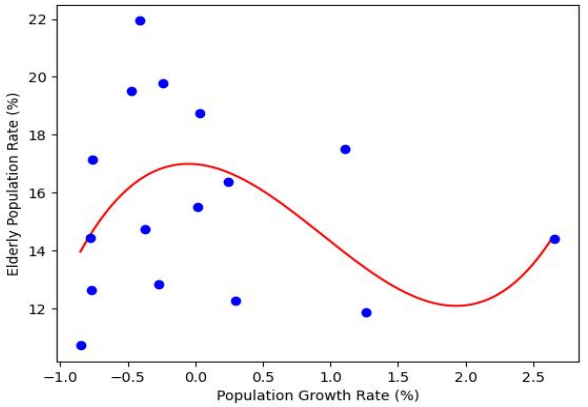
- 텐서플로우를 이용해서 3차 함수 회귀선 구하기

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85, -0.41, -0.27, 0.02,
-0.76, 2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21.94,
12.83, 15.51, 17.14, 14.42]

# a, b, c, d를 랜덤한 값으로 초기화합니다.
a = tf.Variable(random.random())
b = tf.Variable(random.random())
c = tf.Variable(random.random())
d = tf.Variable(random.random())

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

다항 회귀

- 텐서플로우를 이용해서 3차 함수 회귀선 구하기

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

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

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

# 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()
```



딥러닝 네트워크를 이용한 회귀

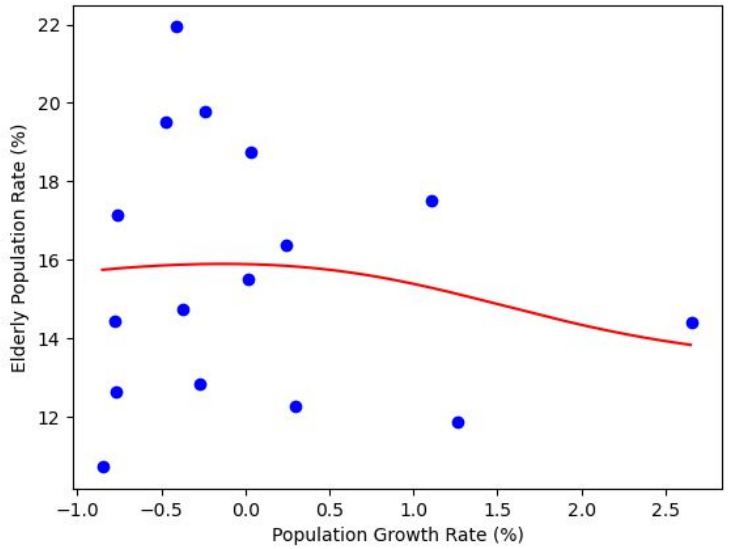
- 딥러닝 네트워크를 이용한 회귀

```
import tensorflow as tf
import numpy as np

X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85, -0.41, -0.27, 0.02,
-0.76, 2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21.94,
12.83, 15.51, 17.14, 14.42]

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()

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

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

딥러닝 네트워크를 이용한 회귀

- 딥러닝 네트워크를 이용한 회귀

```
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-')
plt.plot(X,Y,'bo')
plt.xlabel('Population Growth Rate (%)')
plt.ylabel('Elderly Population Rate (%)')
plt.show()
```



보스턴 주택 가격 데이터 예측

- 데이터 불러 오기

```
from tensorflow.keras.datasets import boston_housing
(train_X, train_Y), (test_X, test_Y) = boston_housing.load_data()

print(len(train_X), len(test_X))
print(train_X[0])
print(train_Y[0])
```

보스턴 주택 가격 데이터 예측

- 데이터 전처리(정규화)

```
x_mean = train_X.mean(axis=0)
x_std = train_X.std(axis=0)
train_X -= x_mean
train_X /= x_std
test_X -= x_mean
test_X /= x_std

y_mean = train_Y.mean(axis=0)
y_std = train_Y.std(axis=0)
train_Y -= y_mean
train_Y /= y_std
test_Y -= y_mean
test_Y /= y_std

print(train_X[0])
print(train_Y[0])
```

보스턴 주택 가격 데이터 예측

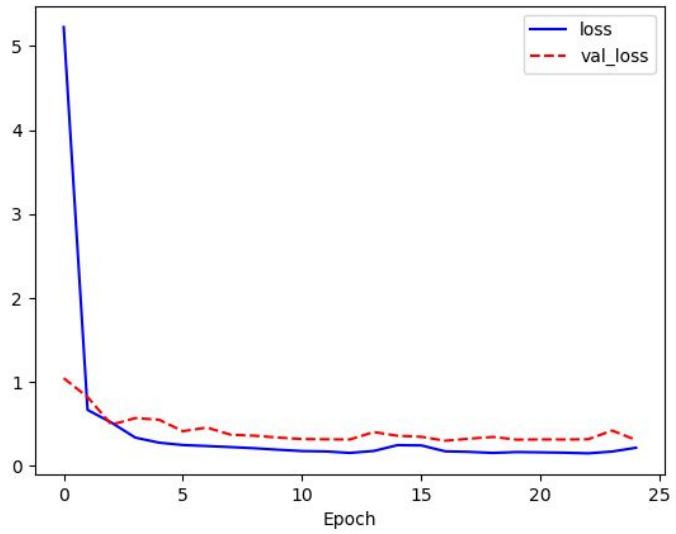
- 회귀 모델 생성

```
model = tf.keras.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()
```

보스턴 주택 가격 데이터 예측

- 회귀 모델 학습 및 학습 결과 시각화

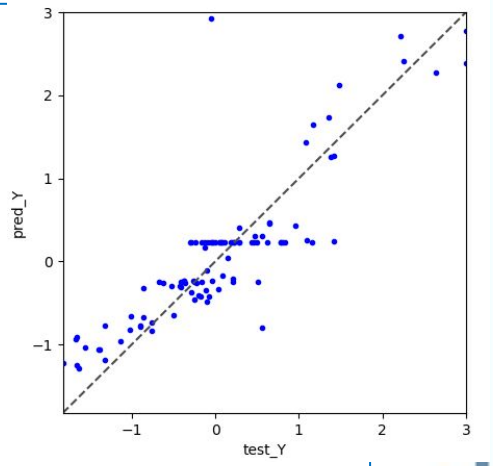
```
history = model.fit(train_X, train_Y, epochs=25, batch_size=32,
validation_split=0.25)


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()
```



보스턴 주택 가격 데이터 예측

- 회귀 모델평가 및 실제 가격과 예측 가격 시각화

```
model.evaluate(test_X, test_Y)

import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(5,5))
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=".3")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```



보스턴 주택 가격 데이터 예측

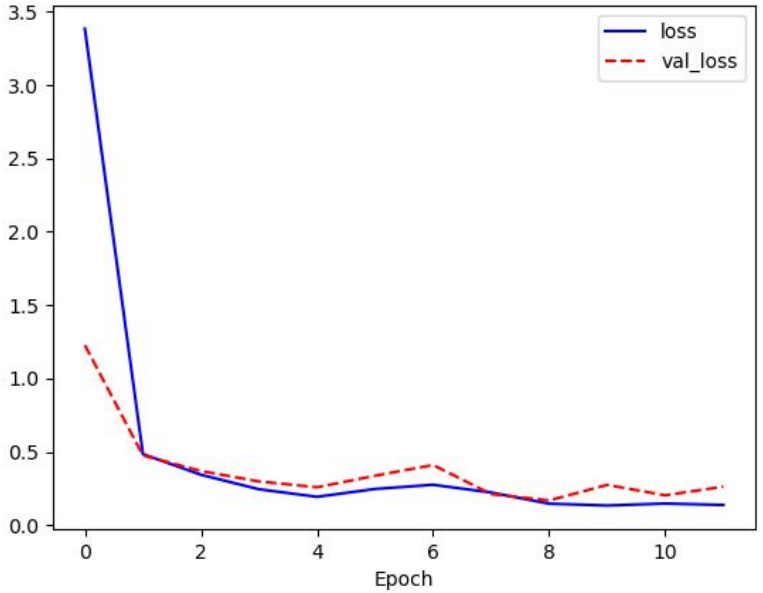
- 모델 재정의 및 학습

```
model = tf.keras.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')

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')])
```

보스턴 주택 가격 데이터 예측

- 모델 학습 결과 시각화

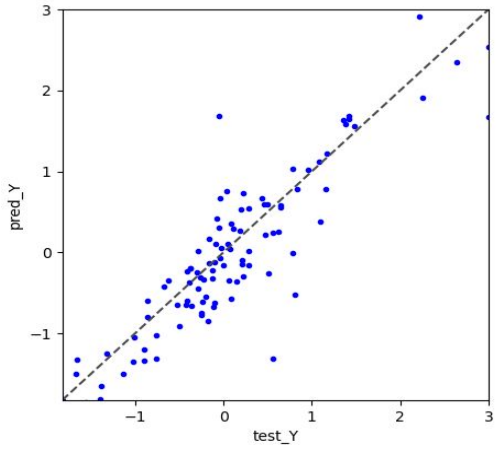
```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



보스턴 주택 가격 데이터 예측

- 모델평가 및 실제 주택 가격과 예측가격 시각화

```
model.evaluate(test_X, test_Y)

import matplotlib.pyplot as plt

pred_Y = model.predict(test_X)

plt.figure(figsize=(5,5))
plt.plot(test_Y, pred_Y, 'b.')
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--",
c=".3")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```



이항 분류

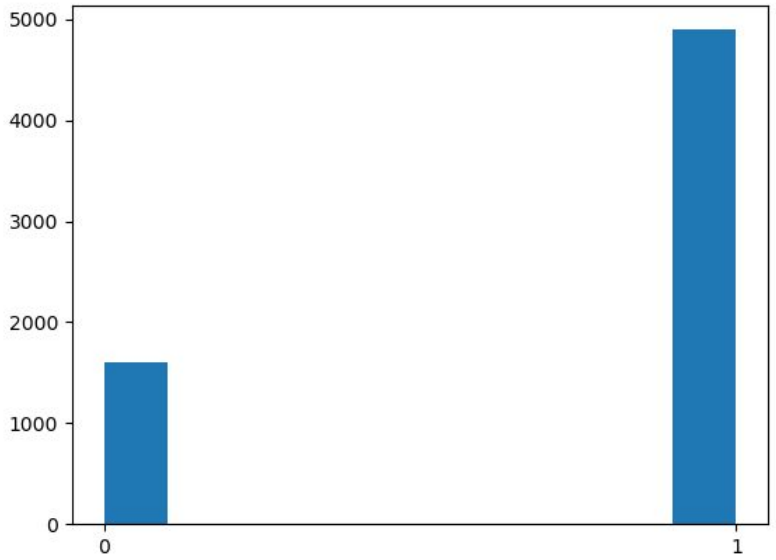
- 와인 데이터셋

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

# 와인 데이터셋 합치기
red['type'] = 0
white['type'] = 1
print(red.head(2))
print(white.head(2))

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

이항 분류

- 히스토그램으로 출력하기

```
import matplotlib.pyplot as plt
plt.hist(wine['type'])
plt.xticks([0, 1])
plt.show()

print(wine['type'].value_counts())
print(wine.info())
```



이항 분류

- 데이터 정규화

```
wine_norm = (wine - wine.min()) / (wine.max() - wine.min())
print(wine_norm.head())
print(wine_norm.describe())
```

- 데이터 섞은 후 numpy array로 변환

```
import numpy as np
wine_shuffle = wine_norm.sample(frac=1)
print(wine_shuffle.head())
wine_np = wine_shuffle.to_numpy()
print(wine_np[:5])
```

이항 분류

- train 데이터와 test 데이터 분리

```
import tensorflow as tf
train_idx = int(len(wine_np) * 0.8)
train_X, train_Y = wine_np[:train_idx, :-1], wine_np[:train_idx, -1]
test_X, test_Y = wine_np[train_idx:, :-1], wine_np[train_idx:, -1]
print(train_X[0])
print(train_Y[0])
print(test_X[0])
print(test_Y[0])
train_Y = tf.keras.utils.to_categorical(train_Y, num_classes=2)
test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=2)
print(train_Y[0])
print(test_Y[0])
```

이항 분류

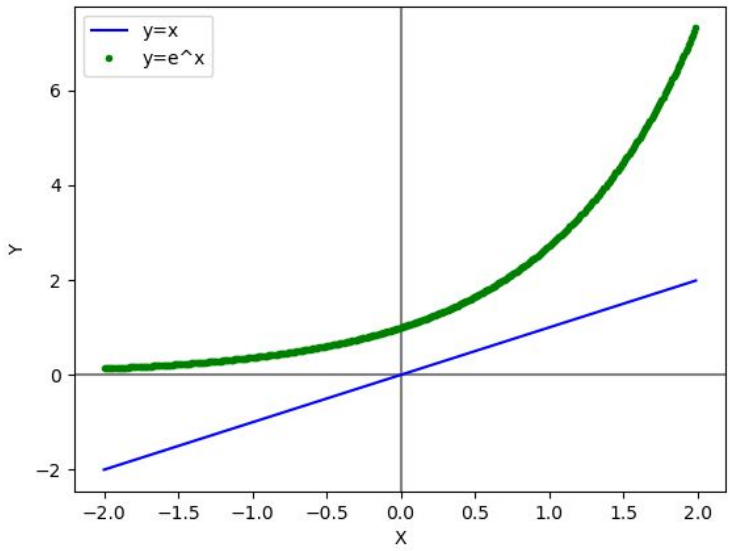
- 와인 데이터셋 분류 모델 생성

```
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=48, activation='relu', input_shape=(12,)),
    tf.keras.layers.Dense(units=24, activation='relu'),
    tf.keras.layers.Dense(units=12, activation='relu'),
    tf.keras.layers.Dense(units=2, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07),
loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

이항 분류

- 시각화 하기

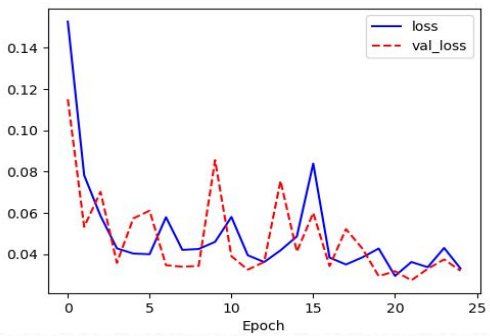
```
import matplotlib.pyplot as plt
import math
import numpy as np
x = np.arange(-2, 2, 0.01)
e_x = math.e ** x

plt.axhline(0, color='gray')
plt.axvline(0, color='gray')
plt.plot(x, x, 'b-', label='y=x')
plt.plot(x, e_x, 'g.', label='y=e^x')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

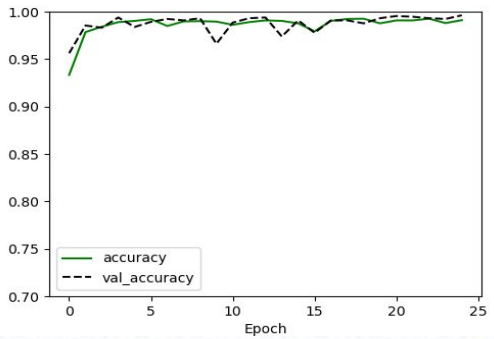


이항 분류

- 와인 데이터셋 분류 모델 학습 및 결과 시각화

```
history = model.fit(train_X, train_Y, epochs=25, batch_size=32,
validation_split=0.25)
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accur
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
plt.show()
model.evaluate(test_X, test_Y)
```





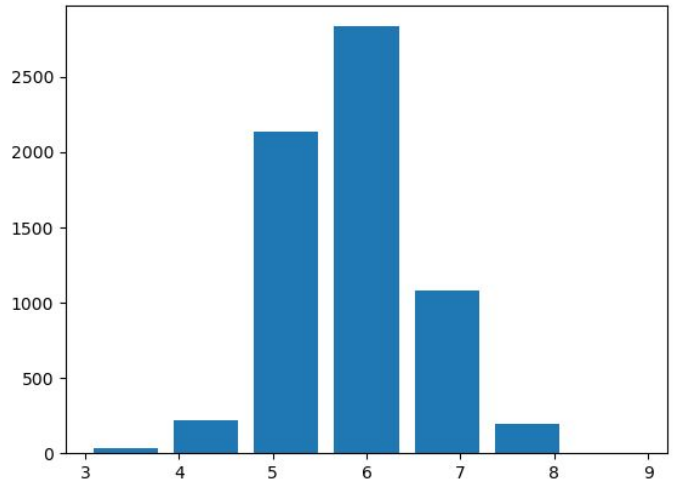
다항 분류

- 품질 데이터 확인

```
print(wine['quality'].describe())
print(wine['quality'].value_counts())
```

- 품질 히스토그램으로 시각화

```
import matplotlib.pyplot as plt
plt.hist(wine['quality'], bins=7, rwidth=0.8)
plt.show()
```



다항 분류

- 품질을 3개의 범주(좋음, 보통, 나쁨)로 재분류

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

print(wine['new_quality'].describe())
print(wine['new_quality'].value_counts())
```

- 데이터 정규화

```
del wine['quality']
wine_backup = wine.copy()
wine_norm = (wine - wine.min()) / (wine.max() - wine.min())
wine_norm['new_quality'] = wine_backup['new_quality']
wine_shuffle = wine_norm.sample(frac=1)
wine_np = wine_shuffle.to_numpy()
```

다항 분류

- train, test 데이터 분리

```
train_idx = int(len(wine_np) * 0.8)
train_X, train_Y = wine_np[:train_idx, :-1], wine_np[:train_idx, -1]
test_X, test_Y = wine_np[train_idx:, :-1], wine_np[train_idx:, -1]
train_Y = tf.keras.utils.to_categorical(train_Y, num_classes=3)
test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=3)
```

- 와인 데이터셋 다항 분류 모델 생성 및 학습

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=48, activation='relu', input_shape=(12,)),
    tf.keras.layers.Dense(units=24, activation='relu'),
    tf.keras.layers.Dense(units=12, activation='relu'),
    tf.keras.layers.Dense(units=3, activation='softmax') ])
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.003),
loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_X, train_Y, epochs=25, batch_size=32,
validation_split=0.25)
```

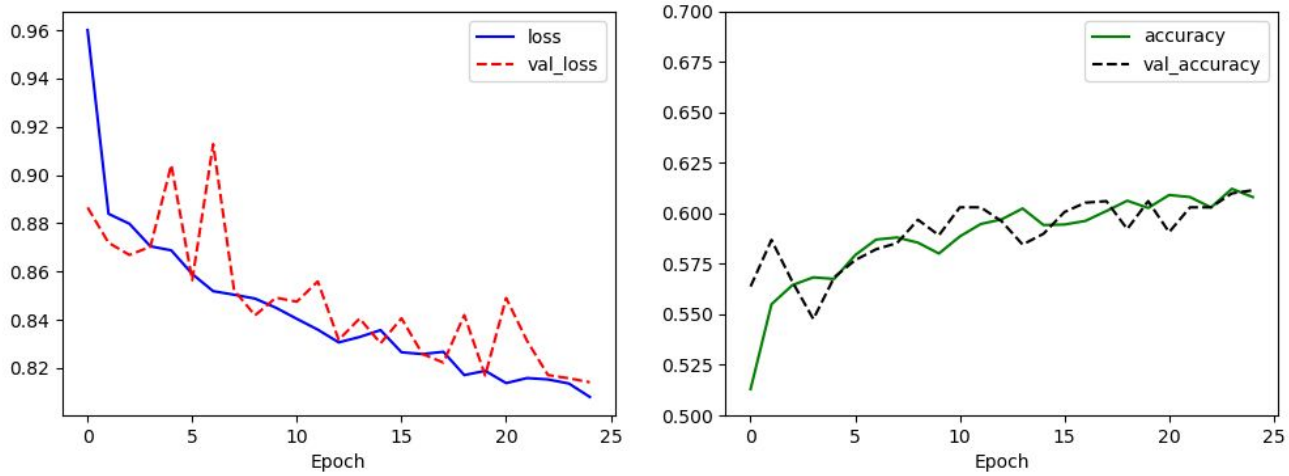
다항 분류

- 다항 분류 모델 학습 결과 시각화

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.5, 0.7)
plt.legend()

plt.show()
```

다항 분류

- 다항 분류 모델 학습 결과 시각화



- 다항 분류 모델 평가

```
model.evaluate(test_X, test_Y)
```

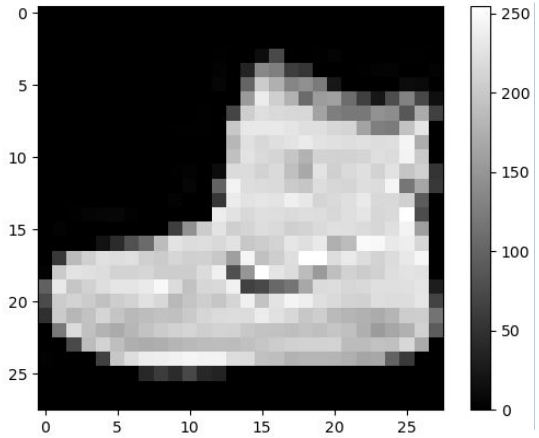
Fashion MNIST

- Fashion MNIST 데이터셋 불러오기

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()

print(len(train_X), len(test_X))
```

- 데이터 확인

```
import matplotlib.pyplot as plt
plt.imshow(train_X[0], cmap='gray')
plt.colorbar()
plt.show()

print(train_Y[0])
```



Fashion MNIST

- 데이터 정규화

```
train_X = train_X / 255.0
test_X = test_X / 255.0

print(train_X[0])
```

- Fashion MNIST 분류 모델

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

Fashion MNIST

- Fashion MNIST 분류 모델 학습

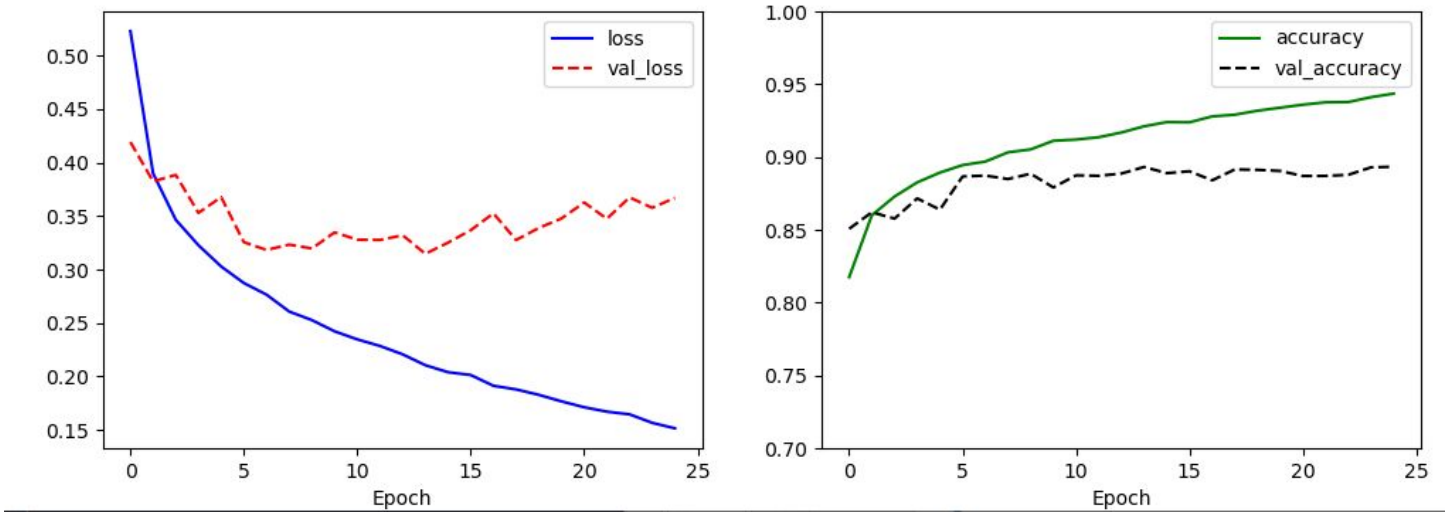
```
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

- Fashion MNIST 분류 모델 학습 결과 시각화

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
plt.show()
```

Fashion MNIST

- Fashion MNIST 분류 모델 학습 결과 시각화



- Fashion MNIST 분류 모델 평가

```
model.evaluate(test_X, test_Y)
```

컨볼루션 신경망

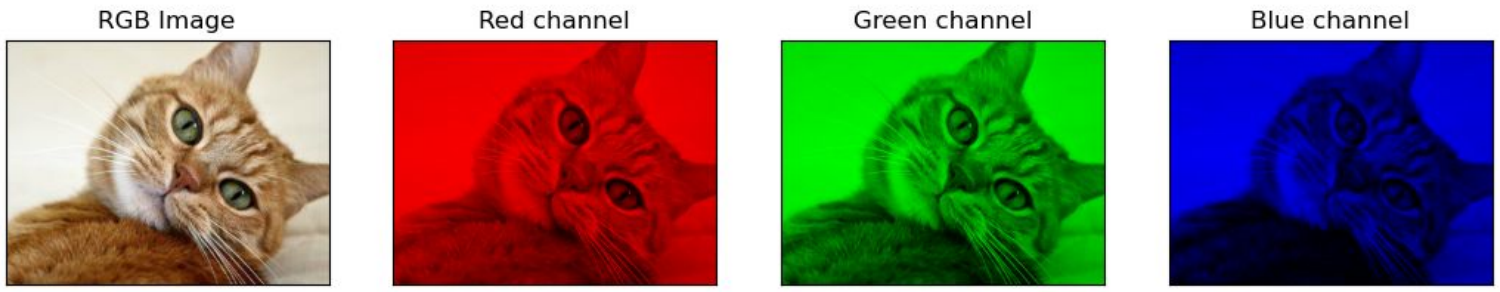
- 고양이 이미지 출력

```
import matplotlib.pyplot as plt

image_path = tf.keras.utils.get_file('cat.jpg', 'http://bit.ly/33U6mH9')
image = plt.imread(image_path)

titles = ['RGB Image', 'Red channel', 'Green channel', 'Blue channel']
cmaps = [None, plt.cm.Reds_r, plt.cm.Greens_r, plt.cm.Blues_r]

from numpy import array, zeros_like
def channel(image, color):
    if color not in (0, 1, 2): return image
    c = image[..., color]
    z = zeros_like(c)
    return array([(c, z, z), (z, c, z), (z, z, c)][color]).transpose(1,2,0)
```

컨볼루션 신경망

- 고양이 이미지 출력

```
colors = range(-1, 3)
fig, axes = plt.subplots(1, 4, figsize=(13,3))
objs = zip(axes, titles, colors)
for ax, title, color in objs:
    ax.imshow(channel(image, color))
    ax.set_title(title)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```



컨볼루션 신경망

- Conv2D 레이어 생성 코드

```
conv1=tf.keras.layers.Conv2D(kernel_size=(3,3),strides=(2,2),padding='valid'
,filters=16)
```

- MaxPool2D 레이어 생성 코드

```
pool1 = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2))
```

- Dropout 레이어 생성 코드

```
pool1 = tf.keras.layers.Dropout(rate=0.3)
```

- Fashion MNIST 데이터셋 불러오기 및 정규화

```
import tensorflow as tf

fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()
train_X = train_X / 255.0
test_X = test_X / 255.0
```

컨볼루션 신경망

- 데이터를 채널을 가진 이미지 형태(3차원)으로 변경

```
print(train_X.shape, test_X.shape)

train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

# reshape 이후
print(train_X.shape, test_X.shape)
```

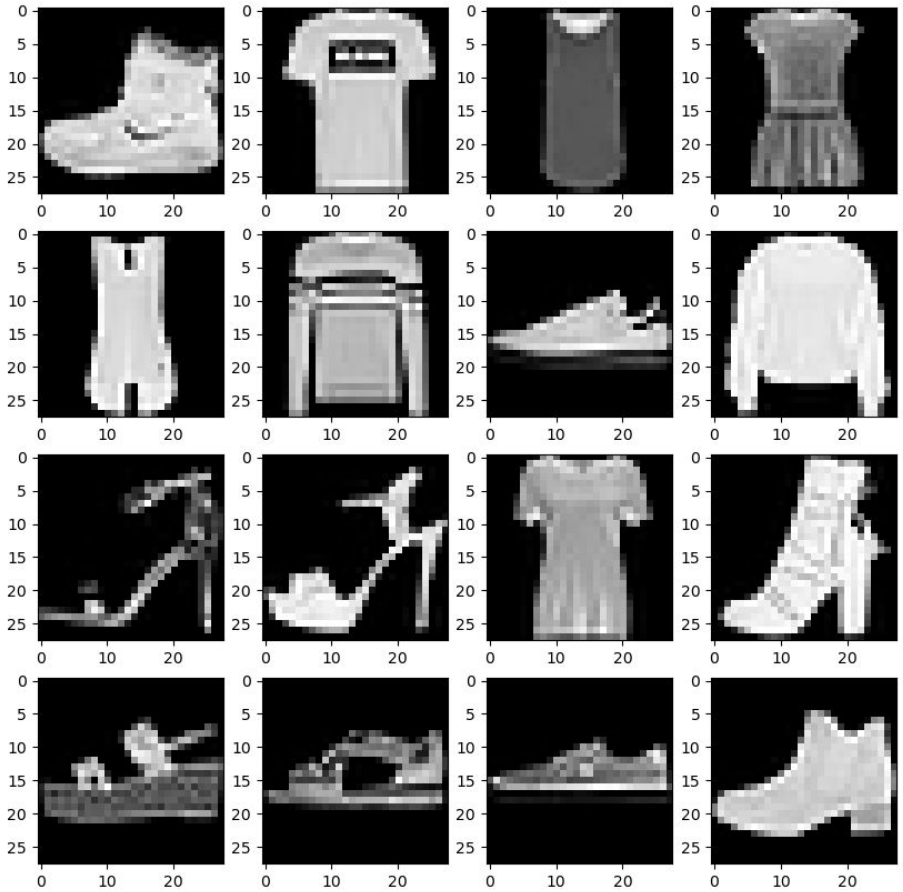
컨볼루션 신경망

- 데이터 확인

```
import matplotlib.pyplot as plt
# 전체 그래프의 사이즈를 width=10, height=10 으로 지정합니다.
plt.figure(figsize=(10, 10))
for c in range(16):
# 4행 4열로 지정한 grid 에서 c+1 번째의 칸에 그래프를 그립니다.
# 1~16 번째 칸을 채우게 됩니다.
    plt.subplot(4,4,c+1)
    plt.imshow(train_X[c].reshape(28,28), cmap='gray')
plt.show()
# train 데이터의 첫번째 ~ 16번째 까지의 라벨을 프린트합니다.
print(train_Y[:16])
```

컨볼루션 신경망

- 데이터 확인



컨볼루션 신경망

- Fashion MNIST 분류 컨볼루션 신경망 모델 정의

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3),
filters=16),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=32),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

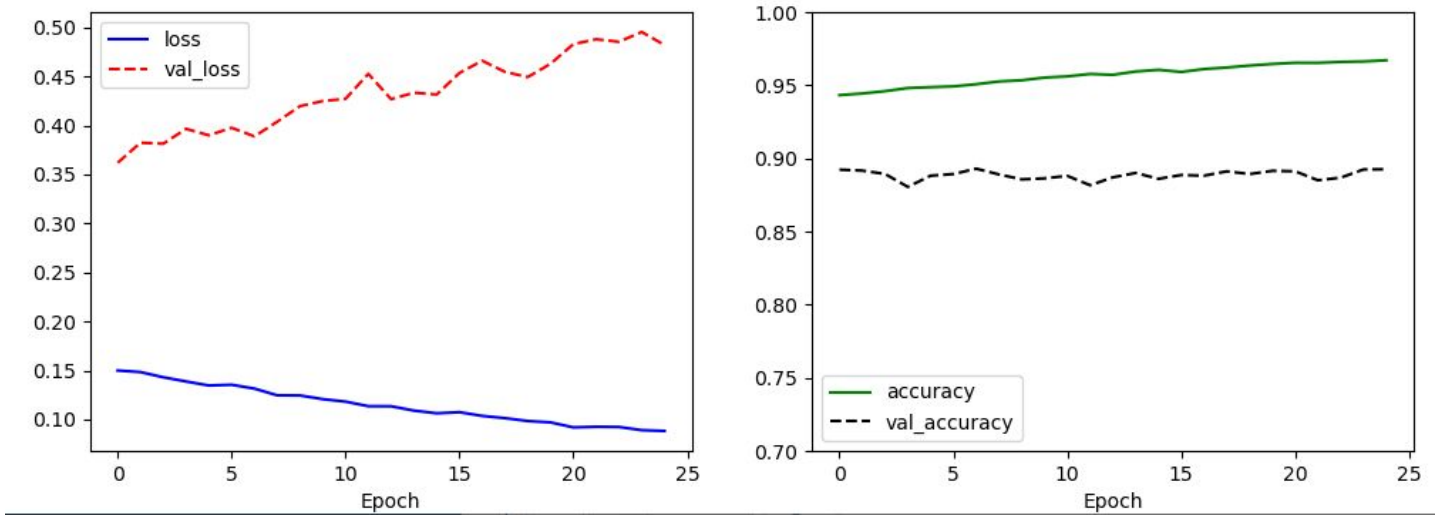
컨볼루션 신경망

- Fashion MNIST 분류 컨볼루션 신경망 모델 학습

```
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
plt.show()
model.evaluate(test_X, test_Y, verbose=0)
```

컨볼루션 신경망

- Fashion MNIST 분류 컨볼루션 신경망 모델 학습



컨볼루션 신경망

- Fashion MNIST 분류 컨볼루션 신경망 모델 정의
  - 풀링 레이어, 드랍아웃 레이어 추가

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3),
filters=32),
    tf.keras.layers.MaxPool2D(strides=(2,2)),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64),
    tf.keras.layers.MaxPool2D(strides=(2,2)),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

컨볼루션 신경망

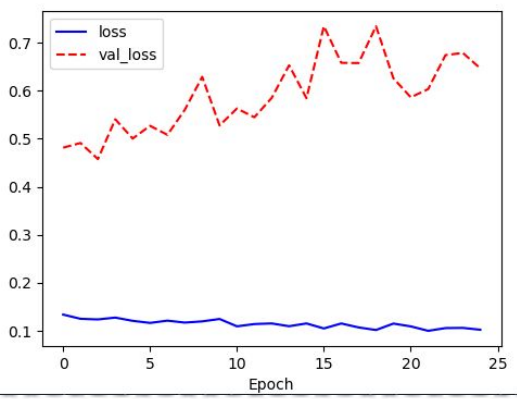
- Fashion MNIST 분류 컨볼루션 신경망 모델 학습
  - 풀링 레이어, 드랍아웃 레이어 추가

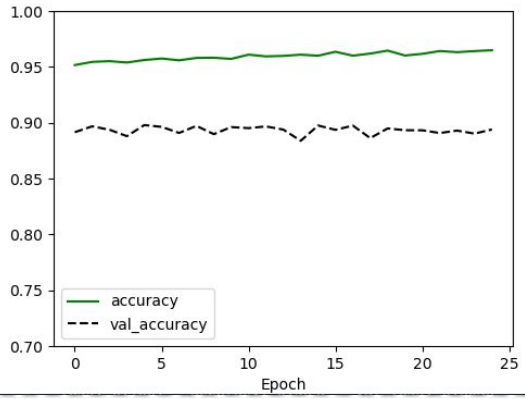
```
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
```

컨볼루션 신경망

- Fashion MNIST 분류 컨볼루션 신경망 모델 학습
  - 풀링 레이어, 드랍아웃 레이어 추가

```
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
plt.show()
model.evaluate(test_X, test_Y, verbose=0)
```





컨볼루션 신경망

- VGGNet 스타일의 Fashion MNIST 분류 컨볼루션 신경망 모델 정의 (퍼포먼스 높이기 향상)

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, padding='same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=256, padding='valid', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

컨볼루션 신경망

- VGGNet 스타일의 Fashion MNIST 분류 컨볼루션 신경망 모델 정의 (퍼포먼스 높이기 향상)

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

컨볼루션 신경망

- VGGNet 스타일의 Fashion MNIST 분류 컨볼루션 신경망 모델 학습

```
history = model.fit(train_X, train_Y, epochs=25,
validation_split=0.25)
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```

컨볼루션 신경망

- Image Augmentation 데이터 표시

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

# 이 ImageDataGenerator 코드 부분은 다음 링크에서 참조했습니다.
#
https://github.com/franneck94/MNIST-Data-Augmentation/blob/master/mnist.py
# rotation, zoom, shift, flip 등을 지정합니다.
image_generator = ImageDataGenerator(
        rotation_range=10,
        zoom_range=0.10,
        shear_range=0.5,
        width_shift_range=0.10,
        height_shift_range=0.10,
        horizontal_flip=True,
        vertical_flip=False)

augment_size = 100
```

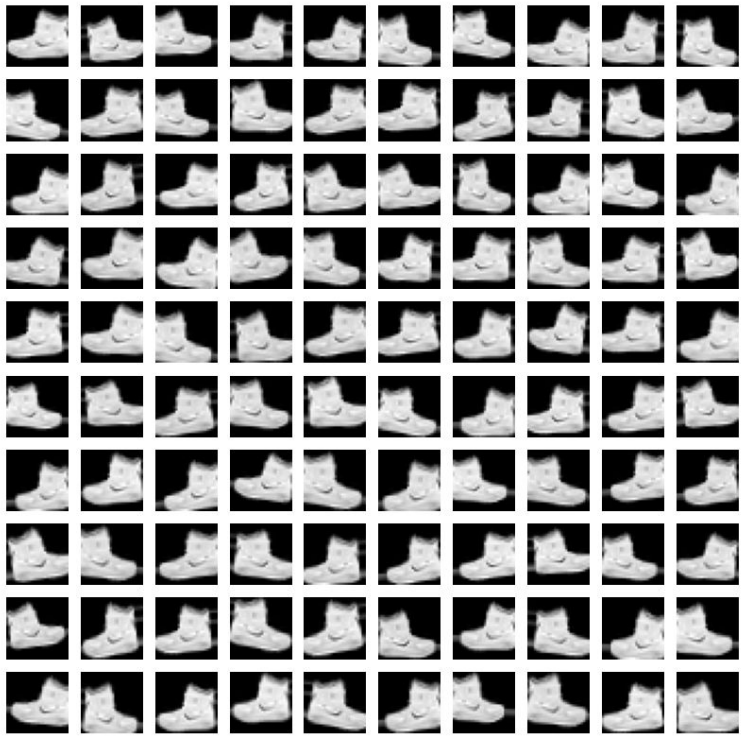
컨볼루션 신경망

- Image Augmentation 데이터 표시

```
x_augmented =
image_generator.flow(np.tile(train_X[0].reshape(28*28),100).reshape(-1,28,28,
1), np.zeros(augment_size), batch_size=augment_size, shuffle=False).next()[0]

# 새롭게 생성된 이미지 표시
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for c in range(100):
    plt.subplot(10,10,c+1)
    plt.axis('off')
    plt.imshow(x_augmented[c].reshape(28,28), cmap='gray')
plt.show()
```

컨볼루션 신경망

- Image Augmentation 데이터 표시



컨볼루션 신경망

- Image Augmentation

```
image_generator = ImageDataGenerator(
        rotation_range=10,
        zoom_range=0.10,
        shear_range=0.5,
        width_shift_range=0.10,
        height_shift_range=0.10,
        horizontal_flip=True,
        vertical_flip=False)
augment_size = 30000
randidx = np.random.randint(train_X.shape[0], size=augment_size)
x_augmented = train_X[randidx].copy()
y_augmented = train_Y[randidx].copy()
x_augmented = image_generator.flow(x_augmented,
np.zeros(augment_size),
                                   batch_size=augment_size, shuffle=False).next()[0]
# 원래 데이터인 x_train 에 Image Augmentation 된 x_augmented 를
추가합니다.
train_X = np.concatenate((train_X, x_augmented))
train_Y = np.concatenate((train_Y, y_augmented))
```

컨볼루션 신경망

- VGGNet style 네트워크 + Image Augmentation 학습

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32,
padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, padding='same',
activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128, padding='same',
activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=256, padding='valid',
activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

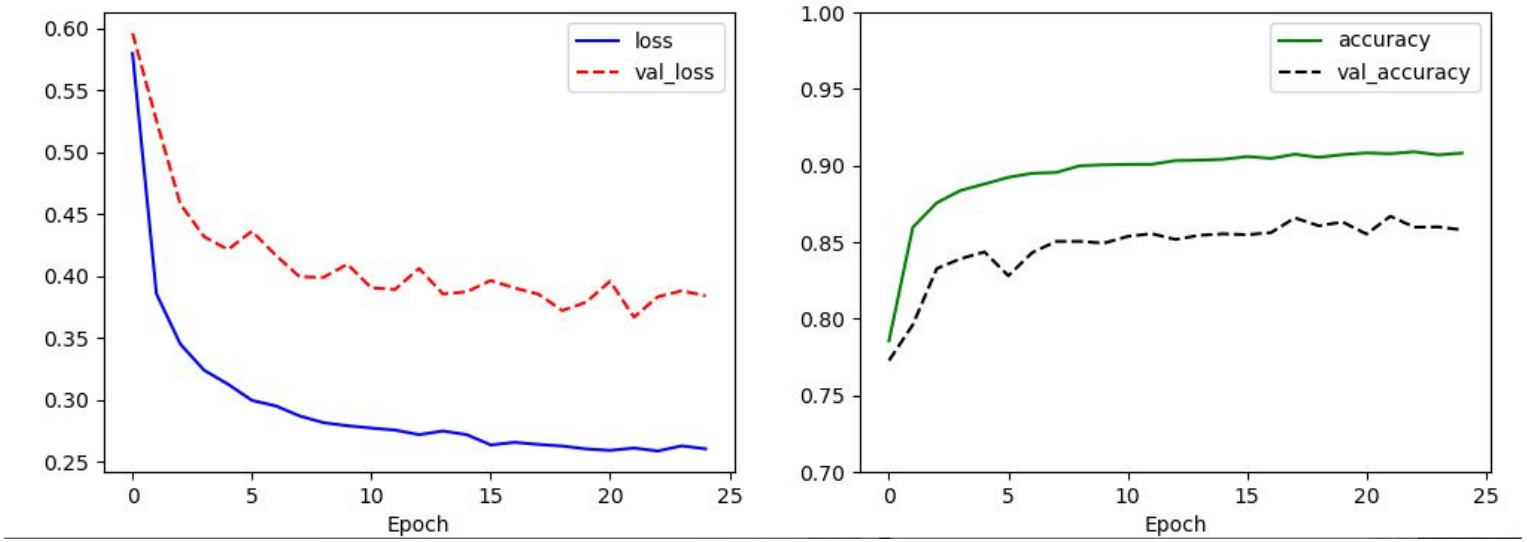
컨볼루션 신경망

- VGGNet style 네트워크 + Image Augmentation 학습

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
plt.show()
model.evaluate(test_X, test_Y, verbose=0)
```

컨볼루션 신경망

- VGGNet style 네트워크 + Image Augmentation 학습



순환 신경망- SimpleRNN 레이어

- SimpleRNN 레이어 생성

```
rnn1 = tf.keras.layers.SimpleRNN(units=1, activation='tanh',
return_sequences=True)
```

- 시퀀스 예측 데이터 생성

```
X = []
Y = []
for i in range(6):
    # [0,1,2,3], [1,2,3,4] 같은 정수의 시퀀스를 만듭니다.
    lst = list(range(i,i+4))
    # 위에서 구한 시퀀스의 숫자들을 각각 10으로 나눈 다음 저장합니다.
   # SimpleRNN 에 각 타임스텝에 하나씩 숫자가 들어가므로 하나씩 분리 배열 저장
    X.append(list(map(lambda c: [c/10], lst)))
    # 정답에 해당하는 4, 5 등의 정수를 역시 위처럼 10으로 나눠서 저장합니다.
    Y.append((i+4)/10)

X = np.array(X)
Y = np.array(Y)
for i in range(len(X)):
    print(X[i], Y[i])
```

순환 신경망- SimpleRNN 레이어

- 시퀀스 예측 모델 정의

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=10, return_sequences=False, input_shape=[4,1]),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

- 네트워크 훈련 및 결과 확인

```
model.fit(X, Y, epochs=100, verbose=0)
print(model.predict(X))
```

- 학습되지 않은 시퀀스에 대한 예측 결과

```
print(model.predict(np.array([[[0.6],[0.7],[0.8],[0.9]]])))
print(model.predict(np.array([[[-0.1],[0.0],[0.1],[0.2]]])))
```

순환 신경망-LSTM 레이어

곱셈 문제 데이터 생성

```
X = []
Y = []
for i in range(3000):
    # 0~1 사이의 랜덤한 숫자 100 개를 만듭니다.
    lst = np.random.rand(100)
    # 마킹할 숫자 2개의 인덱스를 뽑습니다.
    idx = np.random.choice(100, 2, replace=False)
    # 마킹 인덱스가 저장된 원-핫 인코딩 벡터를 만듭니다.
    zeros = np.zeros(100)
    zeros[idx] = 1
    # 마킹 인덱스와 랜덤한 숫자를 합쳐서 X 에 저장합니다.
    X.append(np.array(list(zip(zeros, lst))))
    # 마킹 인덱스가 1인 값들만 서로 곱해서 Y 에 저장합니다.
    Y.append(np.prod(lst[idx]))

print(X[0], Y[0])
```

순환 신경망-LSTM 레이어

- SimpleRNN 레이어를 사용한 곱셈 문제 모델 정의

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=30, return_sequences=True, input_shape=[100,2]),
    tf.keras.layers.SimpleRNN(units=30),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

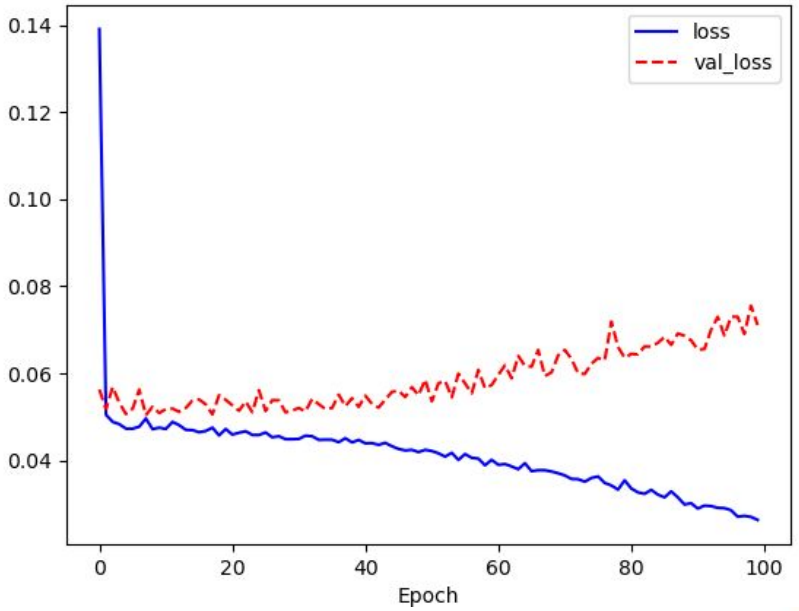
- SimpleRNN 네트워크 학습

```
X = np.array(X)
Y = np.array(Y)
# 2560개의 데이터만 학습시킵니다. validation 데이터는 20% 로 지정합니다.
history = model.fit(X[:2560], Y[:2560], epochs=100, validation_split=0.2)
```

순환 신경망-LSTM 레이어

- SimpleRNN 네트워크 학습 결과 확인

```
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()
```



순환 신경망-LSTM 레이어

- Test 데이터에 대한 예측 정확도 확인

```
model.evaluate(X[2560:], Y[2560:])
prediction = model.predict(X[2560:2560+5])
# 5개 테스트 데이터에 대한 예측을 표시합니다.
for i in range(5):
    print(Y[2560+i], '\t', prediction[i][0], '\tdiff:', abs(prediction[i][0] - Y[2560+i]))

prediction = model.predict(X[2560:])
fail = 0
for i in range(len(prediction)):
if abs(prediction[i][0] - Y[2560+i]) > 0.04: # 오차가 0.04 이상이면 오답입니다.
        fail += 1
print('correctness:', (440 - fail) / 440 * 100, '%')
```

순환 신경망-LSTM 레이어

- LSTM 레이어를 사용한 곱셈 문제 모델 정의

```
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(units=30, return_sequences=True,
input_shape=[100,2]),
    tf.keras.layers.LSTM(units=30),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

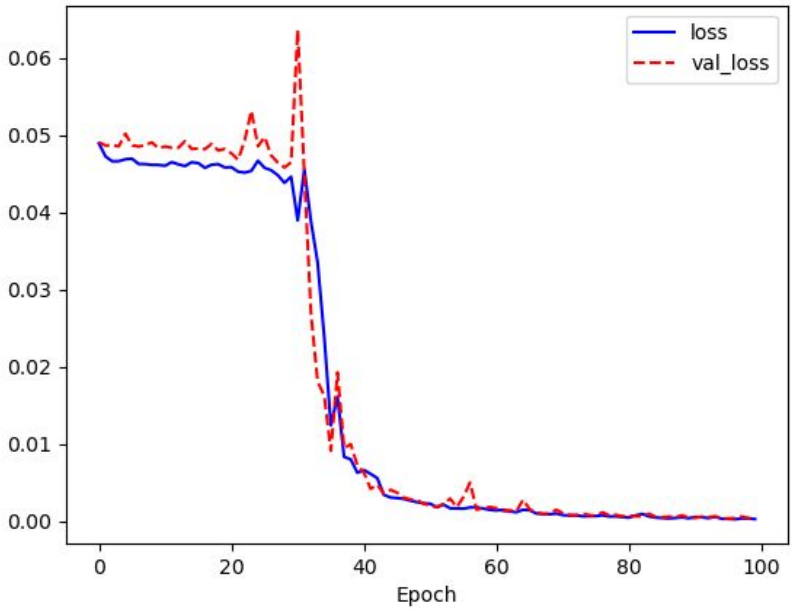
- LSTM 네트워크 학습

```
X = np.array(X)
Y = np.array(Y)
history = model.fit(X[:2560], Y[:2560], epochs=100, validation_split=0.2)
```

순환 신경망-LSTM 레이어

- LSTM 네트워크 학습 결과 확인

```
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()
```



순환 신경망-LSTM 레이어

- Test 데이터에 대한 예측 정확도 확인

```
model.evaluate(X[2560:], Y[2560:])
prediction = model.predict(X[2560:2560+5])
for i in range(5):
    print(Y[2560+i], '\t', prediction[i][0], '\tdiff:', abs(prediction[i][0] - Y[2560+i]))

prediction = model.predict(X[2560:])
cnt = 0
for i in range(len(prediction)):
    if abs(prediction[i][0] - Y[2560+i]) > 0.04:
        cnt += 1
print('correctness:', (440 - cnt) / 440 * 100, '%')
```

순환 신경망-GRU 레이어

- GRU 레이어를 사용한 곱셈 문제 모델 정의

```
model = tf.keras.Sequential([
    tf.keras.layers.GRU(units=30, return_sequences=True,
input_shape=[100,2]),
    tf.keras.layers.GRU(units=30),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

- GRU 네트워크 학습

```
X = np.array(X)
Y = np.array(Y)
history = model.fit(X[:2560], Y[:2560], epochs=100, validation_split=0.2)
```

순환 신경망-GRU 레이어

- GRU 네트워크 학습 결과 확인

```
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()
```

- Test 데이터에 대한 예측 정확도 확인

```
model.evaluate(X[2560:], Y[2560:])
prediction = model.predict(X[2560:2560+5])
for i in range(5):
    print(Y[2560+i], '\t', prediction[i][0], '\tdiff:', abs(prediction[i][0] - Y[2560+i]))

prediction = model.predict(X[2560:])
cnt = 0
for i in range(len(prediction)):
    if abs(prediction[i][0] - Y[2560+i]) > 0.04:
        cnt += 1
print('correctness:', (440 - cnt) / 440 * 100, '%')
```

순환 신경망- 긍정, 부정 감성 분석

- Naver Sentiment Movie Corpus v1.0 다운로드

```
path_to_train_file = tf.keras.utils.get_file('train.txt',
'https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt')
path_to_test_file = tf.keras.utils.get_file('test.txt',
'https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt')
```

- 데이터 로드 및 확인

```
train_text = open(path_to_train_file, 'rb').read().decode(encoding='utf-8')
test_text = open(path_to_test_file, 'rb').read().decode(encoding='utf-8')

# 텍스트가 총 몇 자인지 확인합니다.
print('Length of text: {} characters'.format(len(train_text)))
print('Length of text: {} characters'.format(len(test_text)))
print()
# 처음 300 자를 확인해봅니다.
print(train_text[:300])
```

순환 신경망- 긍정, 부정 감성 분석

- 학습을 위한 정답 데이터(Y) 만들기

```
train_Y = np.array([[int(row.split('\t')[2])] for row in train_text.split('\n')[1:] if
row.count('\t') > 0])
test_Y = np.array([[int(row.split('\t')[2])] for row in test_text.split('\n')[1:] if
row.count('\t') > 0])
print(train_Y.shape, test_Y.shape)
print(train_Y[:5])
```

순환 신경망- 긍정, 부정 감성 분석

- train 데이터의 입력(X)에 대한 정제(Cleaning)

```
import re
# From https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
def clean_str(string):
    string = re.sub(r"[^가-힣A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    string = re.sub(r"\'{2,}", "\'", string)
    string = re.sub(r"\'", "", string)
return string.lower()
```

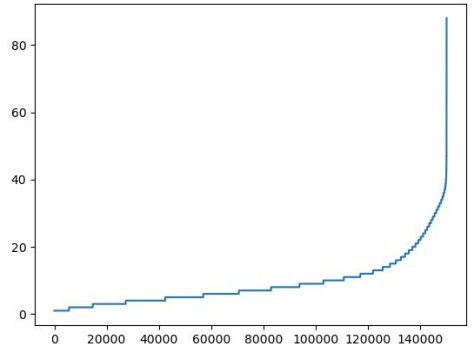
순환 신경망- 긍정, 부정 감성 분석

- train 데이터의 입력(X)에 대한 정제(Cleaning)

```
train_text_X = [row.split('\t')[1] for row in train_text.split('\n')[1:] if row.count('\t')
> 0]
train_text_X = [clean_str(sentence) for sentence in train_text_X]
# 문장을 띄어쓰기 단위로 단어 분리
sentences = [sentence.split(' ') for sentence in train_text_X]
for i in range(5):
    print(sentences[i])
```

- 각 문장의 단어 길이 확인

```
import matplotlib.pyplot as plt
sentence_len = [len(sentence) for sentence in sentences
sentence_len.sort()
plt.plot(sentence_len)
plt.show()
print(sum([int(l<=25) for l in sentence_len]))
```



순환 신경망- 긍정, 부정 감성 분석

- 단어 정제 및 문장 길이 줄임

```
sentences_new = []
for sentence in sentences:
    sentences_new.append([word[:5] for word in sentence][:25])
sentences = sentences_new
for i in range(5):
    print(sentences[i])
```

- Tokenizer와 pad_sequences를 사용한 문장 전처리

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(num_words=20000)
tokenizer.fit_on_texts(sentences)
train_X = tokenizer.texts_to_sequences(sentences)
train_X = pad_sequences(train_X, padding='post')

print(train_X[:5])
```

순환 신경망- 긍정, 부정 감성 분석

- Tokenizer의 동작 확인

```
print(tokenizer.index_word[19999])
print(tokenizer.index_word[20000])
temp = tokenizer.texts_to_sequences(['#$#$#', '경우는', '잊혀질', '연기가'])
print(temp)
temp = pad_sequences(temp, padding='post')
print(temp)
```

- 감성 분석을 위한 모델 정의

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(20000, 300, input_length=25),
    tf.keras.layers.LSTM(units=50),
    tf.keras.layers.Dense(2, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.summary()
```

순환 신경망- 긍정, 부정 감성 분석

- 감성 분석 모델 학습

```
history = model.fit(train_X, train_Y, epochs=5, batch_size=128,
validation_split=0.2)
```

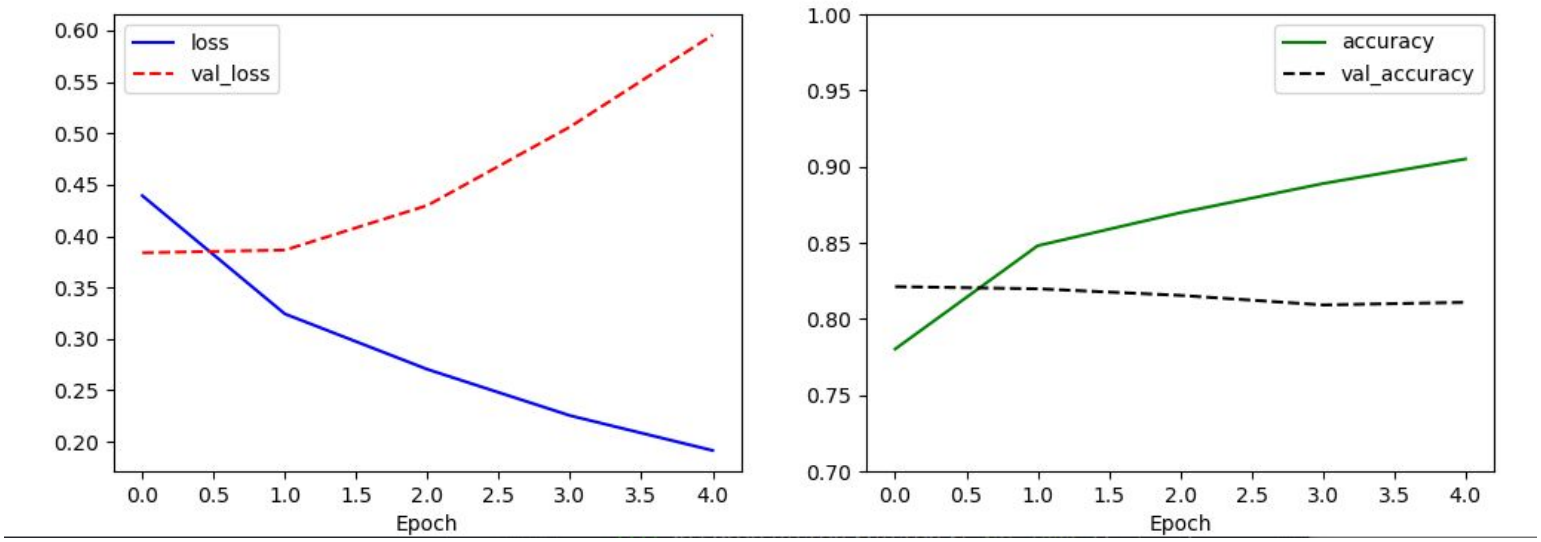
- 감성 분석 모델 학습 결과 확인

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
```

순환 신경망- 긍정, 부정 감성 분석

- 감성 분석 모델 학습 결과 확인

```
plt.ylim(0.7, 1)
plt.legend()

plt.show()
```



순환 신경망- 긍정, 부정 감성 분석

- 테스트 데이터 평가

```
test_text_X = [row.split('\t')[1] for row in test_text.split('\n')[1:] if row.count('\t') > 0]
test_text_X = [clean_str(sentence) for sentence in test_text_X]
sentences = [sentence.split(' ') for sentence in test_text_X]
sentences_new = []
for sentence in sentences:
    sentences_new.append([word[:5] for word in sentence][:25])
sentences = sentences_new

test_X = tokenizer.texts_to_sequences(sentences)
test_X = pad_sequences(test_X, padding='post')

model.evaluate(test_X, test_Y, verbose=0)
```

순환 신경망- 긍정, 부정 감성 분석

- 임의의 문장 감성 분석 결과 확인

```
test_sentence = '재미있을 줄 알았는데 완전 실망했다. 너무 졸리고 돈이 아까웠다.'
test_sentence = test_sentence.split(' ')
test_sentences = []
now_sentence = []
for word in test_sentence:
    now_sentence.append(word)
    test_sentences.append(now_sentence[:])

test_X_1 = tokenizer.texts_to_sequences(test_sentences)
test_X_1 = pad_sequences(test_X_1, padding='post', maxlen=25)
prediction = model.predict(test_X_1)
for idx, sentence in enumerate(test_sentences):
    print(sentence)
    print(prediction[idx])
```

순환 신경망 - 자연어 생성

- 조선왕조실록 데이터 파일 다운로드

```
path_to_file = tf.keras.utils.get_file('input.txt', 'http://bit.ly/2Mc3SOV')
```

- 데이터 로드 및 확인

```
train_text = open(path_to_file, 'rb').read().decode(encoding='utf-8')

# 텍스트가 총 몇 자인지 확인합니다.
print('Length of text: {} characters'.format(len(train_text)))
print()

# 처음 100 자를 확인해봅니다.
print(train_text[:100])
```

순환 신경망 - 자연어 생성

- 훈련 데이터 입력 정제

```
import re
# From
https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
def clean_str(string):
    string = re.sub(r"[^가-힣A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", "", string)
    string = re.sub(r"\)", "", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    string = re.sub(r"\'{2,}", "\'", string)
    string = re.sub(r"\'", "", string)

    return string
```

순환 신경망 - 자연어 생성

- 훈련 데이터 입력 정제

```
train_text = train_text.split('\n')
train_text = [clean_str(sentence) for sentence in train_text]
train_text_X = []
for sentence in train_text:
    train_text_X.extend(sentence.split(' '))
    train_text_X.append('\n')

train_text_X = [word for word in train_text_X if word != '']

print(train_text_X[:20])
```

순환 신경망 - 자연어 생성

- 단어 토큰화

```
vocab = sorted(set(train_text_X))
vocab.append('UNK')
print ('{} unique words'.format(len(vocab)))

# vocab list를 숫자로 맵핑하고, 반대도 실행합니다.
word2idx = {u:i for i, u in enumerate(vocab)}
idx2word = np.array(vocab)

text_as_int = np.array([word2idx[c] for c in train_text_X])

# word2idx 의 일부를 알아보기 쉽게 print 해봅니다.
print('{')
for word,_ in zip(word2idx, range(10)):
    print('  {:4s}: {:3d},'.format(repr(word), word2idx[word]))
print('  ...\n}')

print('index of UNK: {}'.format(word2idx['UNK']))
```

순환 신경망 - 자연어 생성

- 토큰 데이터 확인

```
print(train_text_X[:20])
print(text_as_int[:20])
```

- 기본 데이터셋 만들기

```
seq_length = 25
examples_per_epoch = len(text_as_int) // seq_length
sentence_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

sentence_dataset = sentence_dataset.batch(seq_length+1,
drop_remainder=True)
for item in sentence_dataset.take(1):
    print(idx2word[item.numpy()])
    print(item.numpy())
```

순환 신경망 - 자연어 생성

- 학습 데이터셋 만들기

```
def split_input_target(chunk):
    return [chunk[:-1], chunk[-1]]

train_dataset = sentence_dataset.map(split_input_target)
for x,y in train_dataset.take(1):
    print(idx2word[x.numpy()])
    print(x.numpy())
    print(idx2word[y.numpy()])
    print(y.numpy())
```

- 데이터셋 shuffle, batch 설정

```
BATCH_SIZE = 512
steps_per_epoch = examples_per_epoch // BATCH_SIZE
BUFFER_SIZE = 10000
train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE,
drop_remainder=True)
```

순환 신경망 - 자연어 생성

- 단어 단위 생성 모델 정의

```
total_words = len(vocab)
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(total_words, 100, input_length=seq_length),
    tf.keras.layers.LSTM(units=100, return_sequences=True),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(units=100),
    tf.keras.layers.Dense(total_words, activation='softmax')
])

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

순환 신경망 - 자연어 생성

단어 단위 생성 모델 학습 – 오류 발생

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
def testmodel(epoch, logs):
    if epoch % 5 != 0 and epoch != 49:
        return
    test_sentence = train_text[0]

    next_words = 100
    for _ in range(next_words):
        test_text_X = test_sentence.split(' ')[-seq_length:]
        test_text_X = np.array([word2idx[c] if c in word2idx else word2idx['UNK'] for c in test_text_X])
        test_text_X = pad_sequences([test_text_X], maxlen=seq_length, padding='pre',
value=word2idx['UNK'])

        output_idx = model.predict_classes(test_text_X)
        test_sentence += ' ' + idx2word[output_idx[0]]

    print()
    print(test_sentence)
    print()

testmodelcb = tf.keras.callbacks.LambdaCallback(on_epoch_end=testmodel)
history = model.fit(train_dataset.repeat(), epochs=50, steps_per_epoch=steps_per_epoch,
callbacks=[testmodelcb], verbose=2)
```

순환 신경망 - 자연어 생성

- 임의의 문장을 사용한 생성 결과 확인

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
test_sentence = '동헌에 나가 공무를 본 후 활 십오 순을 쏘았다'

next_words = 100
for _ in range(next_words):
    test_text_X = test_sentence.split(' ')[-seq_length:]
    test_text_X = np.array([word2idx[c] if c in word2idx else word2idx['UNK'] for c
in test_text_X])
    test_text_X = pad_sequences([test_text_X], maxlen=seq_length,
padding='pre', value=word2idx['UNK'])

    output_idx = model.predict_classes(test_text_X)
    test_sentence += ' ' + idx2word[output_idx[0]]

print(test_sentence)
```

순환 신경망 - 자연어 생성

- jamotools 설치

```
get_ipython().system('pip install jamotools')
```

- 자모 분리 테스트

```
import jamotools

train_text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
s = train_text[:100]
print(s)

# 한글 텍스트를 자모 단위로 분리해줍니다. 한자 등에는 영향이 없습니다.
s_split = jamotools.split_syllables(s)
print(s_split)
```

순환 신경망 - 자연어 생성

- 자모 결합 테스트

```
s2 = jamotools.join_jamos(s_split)
print(s2)
print(s == s2)
```

- 자모 토큰화-시간이 걸림.

```
train_text_X = jamotools.split_syllables(train_text)
vocab = sorted(set(train_text_X))
vocab.append('UNK')
print ('{} unique characters'.format(len(vocab)))

# vocab list를 숫자로 맵핑하고, 반대도 실행합니다.
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

text_as_int = np.array([char2idx[c] for c in train_text_X])
```

순환 신경망 - 자연어 생성

- 자모 토큰화-시간이 걸림.

```
# word2idx 의 일부를 알아보기 쉽게 print 해봅니다.
print('{')
for char,_ in zip(char2idx, range(10)):
    print('  {:4s}: {:3d},'.format(repr(char), char2idx[char]))
print('  ...\n}')

print('index of UNK: {}'.format(char2idx['UNK']))
```

- 토큰 데이터 확인

```
print(train_text_X[:20])
print(text_as_int[:20])
```

순환 신경망 - 자연어 생성

- 학습 데이터세트 생성

```
seq_length = 80
examples_per_epoch = len(text_as_int) // seq_length
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

char_dataset = char_dataset.batch(seq_length+1, drop_remainder=True)
for item in char_dataset.take(1):
    print(idx2char[item.numpy()])
    print(item.numpy())

def split_input_target(chunk):
    return [chunk[:-1], chunk[-1]]

train_dataset = char_dataset.map(split_input_target)
```

순환 신경망 - 자연어 생성

- 학습 데이터세트 생성

```
for x,y in train_dataset.take(1):
    print(idx2char[x.numpy()])
    print(x.numpy())
    print(idx2char[y.numpy()])
    print(y.numpy())

BATCH_SIZE = 256
steps_per_epoch = examples_per_epoch // BATCH_SIZE
BUFFER_SIZE = 10000

train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE,
drop_remainder=True)
```

순환 신경망 - 자연어 생성

- 자소 단위 생성 모델 정의

```
total_chars = len(vocab)
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(total_chars, 100, input_length=seq_length),
    tf.keras.layers.LSTM(units=400),
    tf.keras.layers.Dense(total_chars, activation='softmax')
])

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

순환 신경망 - 자연어 생성

- 자소 단위 생성 모델 학습

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
def testmodel(epoch, logs):
    if epoch % 5 != 0 and epoch != 99:
        return
   test_sentence = train_text[:48]
    test_sentence = jamotools.split_syllables(test_sentence)
    next_chars = 300
    for _ in range(next_chars):
        test_text_X = test_sentence[-seq_length:]
        test_text_X = np.array([char2idx[c] if c in char2idx else char2idx['UNK'] for
c in test_text_X])
        test_text_X = pad_sequences([test_text_X], maxlen=seq_length,
padding='pre', value=char2idx['UNK'])
        output_idx = model.predict_classes(test_text_X)
        test_sentence += idx2char[output_idx[0]]
    print()
    print(jamotools.join_jamos(test_sentence))
    print()
```

순환 신경망 - 자연어 생성

- 자소 단위 생성 모델 학습

```
testmodelcb = tf.keras.callbacks.LambdaCallback(on_epoch_end=testmodel)

history = model.fit(train_dataset.repeat(), epochs=100,
steps_per_epoch=steps_per_epoch, callbacks=[testmodelcb], verbose=2)
```

순환 신경망 - 자연어 생성

- 임의의 문장을 사용한 생성 결과 확인

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
test_sentence = '동헌에 나가 공무를 본 후 활 십오 순을 쏘았다'
test_sentence = jamotools.split_syllables(test_sentence)

next_chars = 300
for _ in range(next_chars):
    test_text_X = test_sentence[-seq_length:]
    test_text_X = np.array([char2idx[c] if c in char2idx else char2idx['UNK'] for c
in test_text_X])
    test_text_X = pad_sequences([test_text_X], maxlen=seq_length,
padding='pre', value=char2idx['UNK'])

    output_idx = model.predict_classes(test_text_X)
    test_sentence += idx2char[output_idx[0]]


print(jamotools.join_jamos(test_sentence))
```