딥러닝

Report_2

컴퓨터공학과

2019305059

이현수

문제 1> 난수 데이터 생성과 Pandas 관련 실습

①교재 P57의 tf.random.normal()을 사용하여 5행 3열의 난수 데이터 'df1'과 8행 3열의 난수 데이터 'df2' 를 만든다. 이때, 난수 평균은 '0', 분산은 '1'이다. 만들어진 2개의 난수 데이터를 print 출력한다.

```
import tensorflow as tf

df1 = tf.random.normal([5,3], 0, 1)
df2 = tf.random.normal([8,3], 0, 1)

print(df1)
print()
print(df2)
```

```
tf.Tensor(
[[-0.58988     1.5297867  -1.3960167 ]
 [ 0.50567186  0.70140934 -0.6182757 ]
 [ 0.2355524  -1.6138663  -0.28356633]
 [ 1.1137061   0.7245037  -0.790816  ]
 [-2.5348716   0.23427802  0.26397982]], shape=(5, 3), dtype=float32)

tf.Tensor(
[[-1.0320024   1.6562694   0.7351915 ]
 [-0.23623613  0.6130939   0.22843134]
 [-0.6431004   0.7982766  -1.4253228 ]
 [-0.86395234  0.25895515 -1.5702775 ]
 [ 0.31310076  1.0090969   1.3256186 ]
 [-1.2022506  -0.44673738 -2.5816028 ]
 [ 0.5065931  -0.29928714 -0.61861914]
 [ 0.57299644  0.72942436  0.47014192]], shape=(8, 3), dtype=float32)
```

②①의 df1,df2를 pd.DataFrame()을 사용하여 pandas 형식으로 만든다. pandas frame의 이름을 각각 'df3', 'df4'로 하고. 이때, 열(column)의 이름은 임의로 한다. 단, df1과 df2의 column이름은 동일해야 한다. 'df3', 'df4' 를 print 출력한다.

```
import pandas as pd

col = ['col1', 'col2', 'col3']
row1 = [1,2,3,4,5]
row2 = [6,7,8,9,10,11,12,13]

df3 = pd.DataFrame(df1.numpy(), columns = col, index=row1)
df4 = pd.DataFrame(df2.numpy(), columns = col, index=row2)

print(df3)
print()
print(df4)
```

```
       col1      col2      col3
1 -0.589880  1.529787 -1.396017
2  0.505672  0.701409 -0.618276
3  0.235552 -1.613866 -0.283566
4  1.113706  0.724504 -0.790816
5 -2.534872  0.234278  0.263980

       col1      col2      col3
6  -1.032002  1.656269  0.735192
7  -0.236236  0.613094  0.228431
8  -0.643100  0.798277 -1.425323
9  -0.863952  0.258955 -1.570277
10  0.313101  1.009097  1.325619
11 -1.202251 -0.446737 -2.581603
12  0.506593 -0.299287 -0.618619
13  0.572996  0.729424  0.470142
```

③ 교재 예제5.2를 참조하여 ②의 df3,df4 에 각각 column 2개를 추가하고 이름과 데이터는 임의로 한다.(즉, df3, df4는 각각 5열이 된다.) 'df3', 'df4'를 print 출력한다.

```
df3['type1'] = 10
df3['type2'] = 20
df4['type1'] = 30
df4['type2'] = 40

print(df3)
print()
print(df4)
```

```
       col1      col2      col3  type1  type2
1 -0.589880  1.529787 -1.396017     10     20
2  0.505672  0.701409 -0.618276     10     20
3  0.235552 -1.613866 -0.283566     10     20
4  1.113706  0.724504 -0.790816     10     20
5 -2.534872  0.234278  0.263980     10     20

        col1      col2      col3  type1  type2
6  -1.032002  1.656269  0.735192     30     40
7  -0.236236  0.613094  0.228431     30     40
8  -0.643100  0.798277 -1.425323     30     40
9  -0.863952  0.258955 -1.570277     30     40
10  0.313101  1.009097  1.325619     30     40
11 -1.202251 -0.446737 -2.581603     30     40
12  0.506593 -0.299287 -0.618619     30     40
13  0.572996  0.729424  0.470142     30     40
```

④ 교재 예제5.2를 참조하여 ③의 df3,df4 를 연결하여 하나의 프레임('df') 들고 예제5.6을 참조하여 랜덤하게 섞는다. 'df' 10줄(row)만 print 출력한다.

```
import numpy as np

df = pd.concat([df3, df4])

df = df.sample(frac=1)
print(df.head(10))
```

```
       col1      col2      col3  type1  type2
6  -1.032002  1.656269  0.735192     30     40
1  -0.589880  1.529787 -1.396017     10     20
5  -2.534872  0.234278  0.263980     10     20
9  -0.863952  0.258955 -1.570277     30     40
11 -1.202251 -0.446737 -2.581603     30     40
2   0.505672  0.701409 -0.618276     10     20
7  -0.236236  0.613094  0.228431     30     40
8  -0.643100  0.798277 -1.425323     30     40
13  0.572996  0.729424  0.470142     30     40
3   0.235552 -1.613866 -0.283566     10     20
```

문제 2> 교재 예제 3.23 관련 실습

①코드에서 i = 10 일때 실행 후 output과 error를 실행한 후 output과 error 값을 출력한다.

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

def sigmoid(x):
    return 1 / (1 + math.exp(-x))

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

print('w :', w)
print('b :', b);print()

for i in range(10):
    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
        if i==9:
           print('i=9일 때 j=',j,'일 경우','output :',output,'error :',error)
```

```
w : tf.Tensor([ 1.5808064 -1.6506009], shape=(2,), dtype=float32)
b : tf.Tensor([0.03616123], shape=(1,), dtype=float32)

i=9일 때 j= 0 일 경우 output : 0.49986220337799364 error : -0.49986220337799364
i=9일 때 j= 1 일 경우 output : 0.7779105332324534 error : 0.2220894667675466
i=9일 때 j= 2 일 경우 output : 0.1991995836388571 error : 0.8008004163611429
i=9일 때 j= 3 일 경우 output : 0.5231875241103796 error : -0.5231875241103796
```

②①번의 출력값이 어떻게 계산되어 얻어 졌는지 수식으로 설명하시오.

np.sum(x[j]*w) = [x1,x2] * [w1, w2] = (x1 * w1) + (x2 * w2) = temp1(실수)

b_x * b = 1 * b = temp2 (실수)

temp1 + temp2 = temp3 (실수)

output = sigmoid(temp3)

$$output = \frac{1}{(1+e^{-temp3})} = \text{temp\_output(실수)}$$

error = y[j][0] – output = y – temp_output = temp_error(실수)

w = w + x[j]*0.1*error = [w1, w2] + [x1,x2]*0.1*temp_error

= [w1,w2] + [x1,x2]*temp4(실수) (temp4=0.1*temp_error)

= [w1,w2] + [x1*temp4, x2*temp4] = [w1+x1*temp4, w2+x2*temp4] = [w3,w4]

b = b + b_x*0.1*error = b + 1*0.1*temp_error = temp_b(실수)

(i=0, j = 0 일 때) output = sigmoid(np.sum(x[j]*w)+ b_x*b) 이다

np.sum(x[0]*w) = [1,1] * [1.5808064, -1.6506009] = (1 * 1.5808064) + (1 * -1.6506009) = -0.0697945

b_x * b = 1 * 0.03616123 = 0.03616123 (실수)

np.sum(x[j]*w)+ b_x*b = -0.0697945 + 0.03616123 = -0.03363327

output = sigmoid(-0.03363327)

$$output = \frac{1}{(1+e^{-(-0.03363327)})} = 0.491592475$$

error = y[0][0] – output = 0 – 0.491592475 = -0.491592475

w = w + x[0]*0.1*error = [1.5808064, -1.6506009] + [1,1]*0.1*(-0.491592475)

= [1.5808064, -1.6506009] + [1,1]*(-0.0491592475)

= [1.5808064, -1.6506009] + [1*(-0.0491592475), 1*(-0.0491592475)]

=[1.5808064+ 1*(-0.0491592475), -1.6506009+ 1*(-0.0491592475)]

b = b + b_x*0.1*error = 0.03616123 + 1*0.1*(-0.491592475)=-0.0129980175

(이것을 j=1, j=2, j=3 일때 반복한다.) (그리고 j=0 ~ j=3 의 과정을 i=0 부터 i=9 까지 10 번 반복한다.)

문제 3> 교재 예제 4.4 관련 실습

①코드에서 주어진 X, Y, a, b 를 이용하여 compute_loss()를 1회 실행한 후 y_pred와 loss 값을 출력한다.

```
import tensorflow as tf
import numpy as np
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 = tf.Variable(random.random())
b = tf.Variable(random.random())
print(a)
print(b)
print()

def compute_loss():
    y_pred = a*X + b
    loss = tf.reduce_mean((Y-y_pred)**2)
    print('y_pred : ', y_pred)
    print()
    print('loss : ', loss)
    return loss

optimizer = tf.optimizers.Adam(lr=0.07)
optimizer.minimize(compute_loss, var_list=[a,b])
```

1회실행

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=0.9729808>
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=0.11812344>

y_pred :  tf.Tensor(
[ 0.4100177  -0.64080155  1.3440793   0.14731286  1.1981322   0.35163882
 -0.11539194 -0.33917752 -0.6310718  -0.24187946 -0.7089102  -0.28079867
 -0.14458138  0.13758306 -0.62134194  2.7062526 ], shape=(16,), dtype=float32)

loss :  tf.Tensor(251.26926, shape=(), dtype=float32)
```

랜던값 a

랜던값 b

②①번의 출력값이 어떻게 계산되어 얻어 졌는지 수식으로 설명하시오.

우선 a와 b에 랜덤한 값으로 초기화 한다. 최적화함수를 Adam optimzer를 사용하고 학습률은 0.07으로 설정. optimizer.minimize(compute_loss, var_list=[a,b])함수를 한번 실행. 첫번째 인수 compute_loss는 최소화할 손실을 전달해주고, 두번째 인수 var_list에는 학습시킬 변수리스트 [a,b]를 전달했다.

compute_loss()함수는 잔차의 제곱의 평균을 반환하는 함수이다.

loss = tf.reduce_mean((Y-y_pred)2)**는 기대출력 Y에서 실제출력 y_pred를 뺀값(잔차)의 제곱을 모두 더해서 평균을 낸 값이다.

y_pred = a * X + b를 통해 y_pred 값이 사진과 같이 출력됐다.

y_pred = (0.9729808) * [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37,...생략] + 0.1181234

계산과정을 살펴보면 **0.9729808*0.3 +0.1181234 = 0.4100177** (y_pred의 첫번째 원소)

0.9729808*-0.78+0.1181234 = -0.64080155(y_pred의 두번째 원소) **.....반복**

loss = tf.reduce_mean((Y-y_pred)2)**는 기대출력 Y에서 실제출력 y_pred를 뺀값(잔차)의 제곱을 모두 더해서 평균을 낸 값이다.

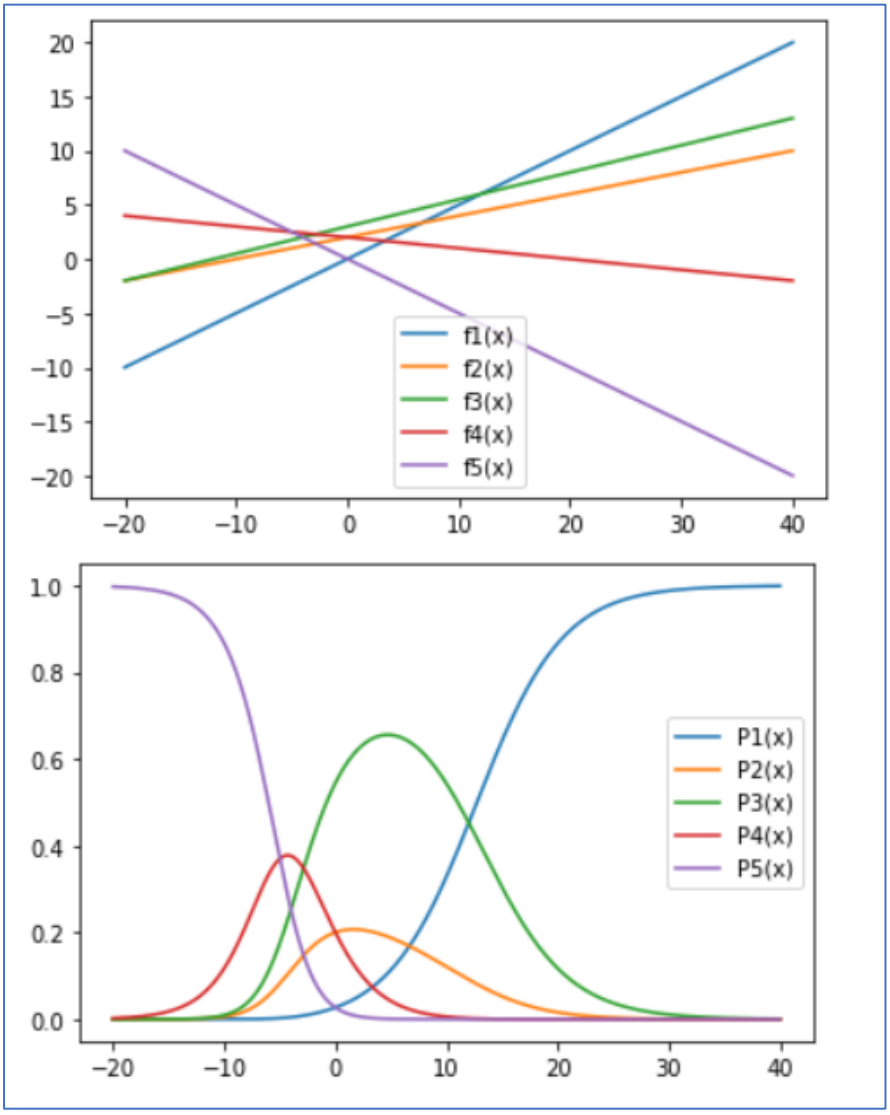
잔차의 제곱의 합 =sum((Y-y_pred)2) =sum(([12.27, 14.44 ....] – [0.4100177, -0.64080155 ...])**2)**

계산해보면 SUM = $(12.27 - 0.4100177)^2 + (14.44 - (-0.64080155))^2$ + ...나머지 14개생략

loss = (SUM) / 16 = 251.26926

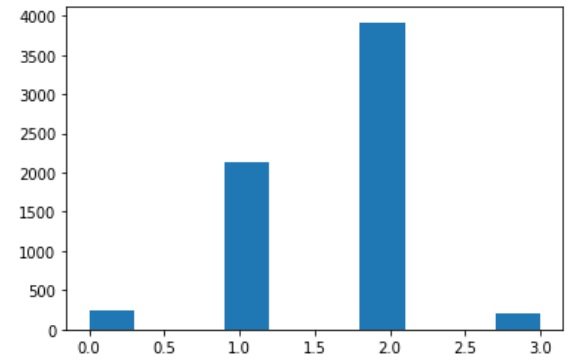
문제 4> 교재 P117의 소프트맥스 변환을 관찰하기 위해 5개의 임의의 직선의 식을 만든 후 소프트맥스로 변환을 한다. 직선의 식과 변환된 소프트맥스 함수를 하나의 그래프 그림으로 plot하여 출력한다. (이를 위한 코드를 임의로 작성한다. x축이 직선의 입력값이 된다)

```
import numpy as np
import matplotlib.pyplot as plt

def softmax(x):
    exp_x = np.exp(x)
    return exp_x / exp_x.sum(axis=0)

line_x = np.arange(-20, 40, 0.01)

line_y1 = 0.5*line_x
line_y2 = 0.2*line_x+2
line_y3 = 0.25*line_x+3
line_y4 = -0.1*line_x+2
line_y5 = -0.5*line_x

plt.plot(line_x, line_y1)
plt.plot(line_x, line_y2)
plt.plot(line_x, line_y3)
plt.plot(line_x, line_y4)
plt.plot(line_x, line_y5)

plt.legend(['f1(x)','f2(x)','f3(x)','f4(x)','f5(x)'])
plt.show()

scores = [line_y1, line_y2, line_y3,line_y4, line_y5]
plt.plot(line_x, softmax(scores).T)

plt.legend(['P1(x)','P2(x)','P3(x)','P4(x)','P5(x)'])
plt.show()
```



문제 5> 교재 5.2에서 wine의 quality를 4종류로 나누어 4항 분류로 신경망을 바꾸고 p128을 참조하여 분류 성적이 70% 이상이 되도록 학습하고 결과를 p129와 같이 시각화 하시오. loss 와 plot을 출력한다. (이때, 분류 성적이 잘 나오도록 epochs, batch_size, validation_split를 임의로 한다.)

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

red = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv', sep=';')
white = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv', sep=';')
red['type']=0
white['type']=1

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

wine.loc[wine['quality']<=7, 'new_quality'] = 2
wine.loc[wine['quality']<=4, 'new_quality'] = 0
wine.loc[wine['quality']==5, 'new_quality'] = 1
wine.loc[wine['quality']>=8, 'new_quality'] = 3

plt.hist(wine['new_quality'])
plt.show()
print(wine['new_quality'].value_counts())
```

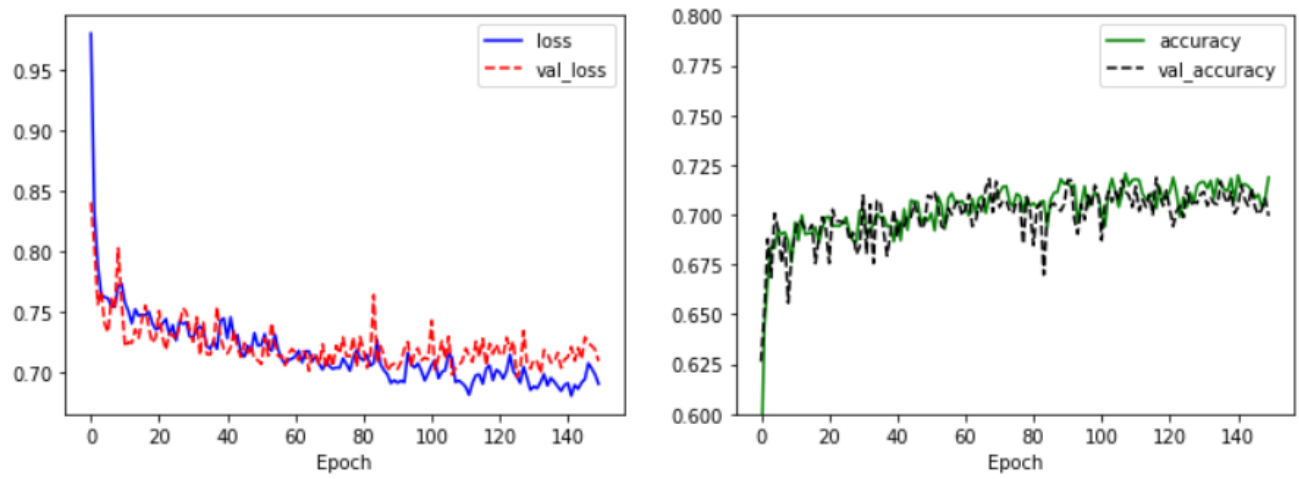



```
2.0    3915
1.0    2138
0.0     246
3.0     198
Name: new_quality, dtype: int64
```

4 항분류를 위 사진처럼 나눴다.

```
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()

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=4)
test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=4)

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=4, activation='softmax')
])

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

history = model.fit(train_X, train_Y, epochs=150, batch_size=250, validation_split=0.3)
```

```
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(0.6, 0.8)
plt.legend()

plt.show()
```




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

```
41/41 [==============================] - 0s 969us/step - loss: 0.7157 - accuracy: 0.7015
[0.7157419323921204, 0.7015384435653687]
```

정확도가 70.15%가 나왔다.