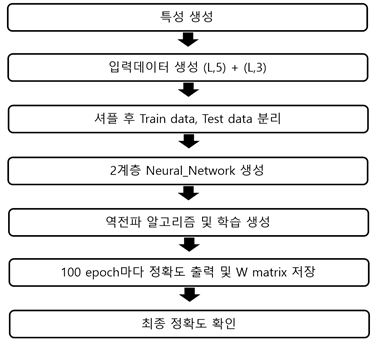
이번에는 Mnist 데이터들을 활용해서 직접 0,1,2를 구분하는 인공지능을 만들어보았다.

이를 확장해서 구현한다면 0~9까지 구분하는 인공지능을 라이브러리 없이 직접 만들어보는 것이 가능할 것이다.

이전 신경망까지 잘 따라왔다면 오히려 쉽다! 입력 후보군 5개와 0,1,2로 된 출력 후보군 3개를 넣어주기만 하면 된다.

먼저 흐름도를 통해 전체적인 흐름을 파악해보자.



[[배포용] MINIST Data.zip](https://blog.kakaocdn.net/dn/wGneQ/btrE6xZXIZ8/tmmWSR1ZWgLul5JhZa78Q1/%5B%EB%B0%B0%ED%8F%AC%EC%9A%A9%5D%20MINIST%20Data.zip?attach=1&knm=tfile.zip)

[0.83MB](https://blog.kakaocdn.net/dn/wGneQ/btrE6xZXIZ8/tmmWSR1ZWgLul5JhZa78Q1/%5B%EB%B0%B0%ED%8F%AC%EC%9A%A9%5D%20MINIST%20Data.zip?attach=1&knm=tfile.zip)

우선 일반적인 Minist 를 받아오는것이 아닌 기존에 했던것처럼 엑셀형식으로 가공된 파일이 있다. 이들을 활용해서 구현할 것이다.

출력은 0,1,2로 나오기 때문에 ONE-HOT-ENCODIG을 거친다면 100, 010, 001 3가지의 출력 후보군이 나올것이다. 그렇다면 입력 후보군들은 어떻게 선택해야 할까?

(1) 가로축 Projection => 확률밀도함수로 변환 => 기댓값

(2) 가로축 Projection => 확률밀도함수로 변환 => 분산

(3) 세로축 Projection => 확률밀도함수로 변환 => 기댓값

(4) 세로축 Projection => 확률밀도함수로 변환 => 분산

(5) Diagonal\* 원소 배열 추출 => 밀도함수로 변환 => 기댓값

\* Diagonal: 좌측 상단에서 우측 하단으로 내려오는 대각선

(6) Diagonal 원소 배열 추출 => 확률밀도함수로 변환 => 분산

(7) Diagonal 원소 배열 추출 => 0의 개수

(8) Anti-Diagonal\*\* 원소 배열 추출 => 밀도함수로 변환 => 기댓값

\*\* Anti\_Diagonal: 우측 상단에서 좌측 하단으로 내려오는 대각선

(9) Anti-Diagonal 원소 배열 추출 => 확률밀도함수로 변환 => 분산

(10) Anti-Diagonal 원소 배열 추출 => 0의 개수

위와같이 10개의 후보군이 있고 여기서 5개정도를 선택하여 입력으로 넣어주기만 하면 된다.

###############################

def feature\_1(input\_data):

# 특징 후보 1번 : 가로축 Projection => 확률밀도함수로 변환 => 기댓값

X = sum(input\_data)

input\_data = input\_data.T

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

output\_value = sum(X \* pdf)

return output\_value

#####################################################################################

def feature\_2(input\_data):

# 특징 후보 2번 : 가로축 Projection => 확률밀도함수로 변환 => 분산

X = sum(input\_data)

input\_data = input\_data.T

S = sum(X)

pdf = np.array([ sum(input\_data[i])/S for i in range(len(input\_data)) ])

E = sum(X \* pdf)

output\_value = sum( (X - E)\*\*2 \* pdf)

return output\_value

#####################################################################################

def feature\_3(input\_data):

# 특징 후보 3번 : 세로축 Projection => 확률밀도함수로 변환 => 기댓값

X = sum(input\_data.T)

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

output\_value = sum(X \* pdf)

return output\_value

#####################################################################################

def feature\_4(input\_data):

# 특징 후보 4번 : 세로축 Projection => 확률밀도함수로 변환 => 분산

X = sum(input\_data.T)

input\_data = input\_data

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

E = sum(X \* pdf)

output\_value = sum((X - E) \*\*2 \* pdf)

return output\_value

def feature\_5(input\_data):

# 특징 후보 5번 : Diagonal 원소배열 추출 => 밀도함수로 변환 => 기댓값

input\_data = np.diag(input\_data.T)

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

output\_value = sum(input\_data \* pdf)

return output\_value

def feature\_6(input\_data):

# 특징 후보 6번 : Diagonal 원소배열 추출 => 밀도함수로 변환 => 분산

input\_data = np.diag(input\_data.T)

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

E = sum(input\_data \* pdf)

output\_value = sum((input\_data - E) \*\*2 \* pdf)

return output\_value

def feature\_7(input\_data):

# 특징 후보 7번 : Diagonal 원소배열 추출 => 0의 개수

input\_data = np.diag(input\_data.T)

cnt=0

for i in input\_data:

if i ==0:

cnt +=1

return cnt

def feature\_8(input\_data):

# 특징 후보 8번 : Anti-Diagonal 원소배열 추출 => 밀도함수로 변환 => 기댓값

input\_data = np.diag(np.fliplr(input\_data))

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

output\_value = sum(input\_data \* pdf)

return output\_value

def feature\_9(input\_data):

# 특징 후보 9번 : Anti-Diagonal 원소배열 추출 => 밀도함수로 변환 => 분산

input\_data = np.diag(np.fliplr(input\_data))

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

E = sum(input\_data \* pdf)

output\_value = sum((input\_data - E) \*\*2 \* pdf)

return output\_value

def feature\_10(input\_data):

# 특징 후보 10번 : Anti-Diagonal 원소배열 추출 => 0의 개수

input\_data = np.diag(np.fliplr(input\_data))

cnt=0

for i in input\_data:

if i ==0:

cnt +=1

return cnt

feature = [0,feature\_1,feature\_2,feature\_3,feature\_4,feature\_5,feature\_6,feature\_7,feature\_8,feature\_9,feature\_10]

해당 후보군들을 실제로 나타낸다면 위코드들을 통해서 나타낼 수 있다.

이 코드들을 활용해서 실제 모델을 만들어보면 아래코드와 같다.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

class perceptron: # perceptron 클래스 구현

def \_\_init\_\_(self, w):

self.w = w

def output(self, x):

return np.dot(np.append(x, 1), self.w)

def sigmoid(x): # 시그모이드 함수

return 1 / (1 + np.exp(-x))

def one\_hot\_encoding(array, size):

lst = [] # one\_hot\_encoding 과정

for i in array:

tmp = [0] \* size

tmp[i -1] +=1

lst.append(tmp)

return np.array(lst)

class Neural\_Network: # 2계층 신경망 구현

def \_\_init\_\_(self, hidden\_layer\_size, Input, Output, learning\_rate,Test\_set):

self.hidden\_layer\_size = hidden\_layer\_size # 은닉층 노드 개수

self.Input\_size = Input.shape[1] # 입력층 사이즈

self.Output\_size = Output.shape[1] # 출력층 사이즈

self.X = Input # 트레이닝 셋 입력

self.Y = Output # 트레이닝 셋 출력(정답)

self.learning\_rate = learning\_rate # Learning rate

self.Create\_Weight\_Matrix() # W0,W1매트릭스 임의 생성

self.Test\_set = Test\_set # 테스트셋 저장

def Create\_Weight\_Matrix(self): # Weight 를 만드는 함수

self.W0 = np.random.randn(self.Input\_size +1, self.hidden\_layer\_size)

self.W1 = np.random.randn(self.hidden\_layer\_size +1, self.Output\_size)

def Set\_Hidden\_layer\_Node\_size(self, size): # 히든Node수 설정

self.hidden\_layer\_size = size

self.Create\_Weight\_Matrix()

def Check\_Input\_Output\_size(self): # Input,Output 체크함수

print('Input 속성 수 ====>', self.Input\_size)

print('Output 속성 수 ===>', self.Output\_size)

def predict(self, x): # y예측 함수

INPUT\_LAYER = perceptron(self.W0)

OUTPUT\_LAYER = perceptron(self.W1)

self.sigmoid\_input = sigmoid(INPUT\_LAYER.output(x))

self.H = np.append(self.sigmoid\_input, 1)

return sigmoid(OUTPUT\_LAYER.output(self.sigmoid\_input))

def Back\_propagation(self):

lr =self.learning\_rate # Learning rate

for i in range(len(self.X)):

Y\_pred =self.predict(self.X[i]) # Y 예측값

Input = np.append(self.X[i], 1) # 입력층 + 1 추가

# 역전파를 1단계부터 시행하면 W가 업데이트되어 2단계부터 시행

for j in range(self.Input\_size +1): # 역전파 2단계

for k in range(self.hidden\_layer\_size):

Etotal\_h\_diff =0

for q in range(self.Output\_size):

Etotal\_h\_diff +=-2 \* (self.Y[i][q] - Y\_pred[q]) \* Y\_pred[q] \* (1 - Y\_pred[q]) \*self.W1[k][q]

h\_z\_diff =self.H[k] \* (1 -self.H[k])

z\_w\_diff = Input[j]

Etotal\_w = Etotal\_h\_diff \* h\_z\_diff \* z\_w\_diff

self.W0[j][k] =self.W0[j][k] - lr \* Etotal\_w # W0업데이트

for j in range(self.hidden\_layer\_size +1): # 역전파 1단계

for k in range(self.Output\_size):

E\_o\_diff =-2 \* (self.Y[i][k] - Y\_pred[k])

o\_z\_diff = Y\_pred[k] \* (1 - Y\_pred[k])

z\_w\_diff =self.H[j]

Etotal\_w = E\_o\_diff \* o\_z\_diff \* z\_w\_diff

self.W1[j][k] =self.W1[j][k] - lr \* Etotal\_w # W1 업데이트

def train(self, epoch):

self.epoch = epoch # epoch 저장

self.MSEs = [] # MSE그래프를 그리기 위함

self.Accuaracys = [] # 정확도 그래프를 그리기 위함

for i in range(epoch):

data = np.concatenate([self.X, self.Y], 1) # 셔플과정

np.random.shuffle(data) # 매 에폭마다 섞어주는 과정

self.X, none, self.Y = np.hsplit(data, (self.Input\_size, self.Input\_size))

self.Back\_propagation() # 역전파 과정으로 W업데이트

if i % 100 ==0:

tmp\_mse = [] # 정확도 계산

cnt =0

for j in range(len(self.X)):

Y\_pred =self.predict(self.X[j])

tmp\_mse.append(np.mean((self.Y[j] - Y\_pred) \*\*2))

maxindex = np.argmax(self.predict(self.X[j])) # 가장큰 index가져오기

tmp = np.array([0] \*self.Output\_size)

tmp[maxindex] =1

if np.array\_equal(tmp, self.Y[j]): # 정답과 비교

cnt +=1

Accuracy = cnt /len(self.X)

self.Accuaracys.append(Accuracy)

MSE = np.mean(tmp\_mse)

self.MSEs.append(MSE)

test\_X, none, test\_Y = np.hsplit(Test\_set, (self.Input\_size, self.Input\_size))

cnt =0

for j in range(len(test\_X)):

maxindex = np.argmax(self.predict(test\_X[j]))

tmp = np.array([0] \*self.Output\_size)

tmp[maxindex] =1

if np.array\_equal(tmp, test\_Y[j]):

cnt +=1

test\_Accuracy = cnt /len(test\_X)

print(f 'EPOCH {i} ===> MSE : {MSE} , Accuracy : {Accuracy} ,Test\_Accuracy : {test\_Accuracy}')

# W 저장 ( W폴더를 하나 만들어야 저장이 가능함 )

# df0 = pd.DataFrame(self.W0)

# df0.to\_csv(f'W\\{i}epoch\_W0.csv',index=False,header='None')

# df1 = pd.DataFrame(self.W1)

# df1.to\_csv(f'W\\{i}epoch\_W1.csv', index=False,header='None')

#####################################################################################

def feature\_1(input\_data):

# 특징 후보 1번 : 가로축 Projection => 확률밀도함수로 변환 => 기댓값

X = sum(input\_data)

input\_data = input\_data.T

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

output\_value = sum(X \* pdf)

return output\_value

#####################################################################################

def feature\_2(input\_data):

# 특징 후보 2번 : 가로축 Projection => 확률밀도함수로 변환 => 분산

X = sum(input\_data)

input\_data = input\_data.T

S = sum(X)

pdf = np.array([ sum(input\_data[i])/S for i in range(len(input\_data)) ])

E = sum(X \* pdf)

output\_value = sum( (X - E)\*\*2 \* pdf)

return output\_value

#####################################################################################

def feature\_3(input\_data):

# 특징 후보 3번 : 세로축 Projection => 확률밀도함수로 변환 => 기댓값

X = sum(input\_data.T)

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

output\_value = sum(X \* pdf)

return output\_value

#####################################################################################

def feature\_4(input\_data):

# 특징 후보 4번 : 세로축 Projection => 확률밀도함수로 변환 => 분산

X = sum(input\_data.T)

input\_data = input\_data

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

E = sum(X \* pdf)

output\_value = sum((X - E) \*\*2 \* pdf)

return output\_value

def feature\_5(input\_data):

# 특징 후보 5번 : Diagonal 원소배열 추출 => 밀도함수로 변환 => 기댓값

input\_data = np.diag(input\_data.T)

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

output\_value = sum(input\_data \* pdf)

return output\_value

def feature\_6(input\_data):

# 특징 후보 6번 : Diagonal 원소배열 추출 => 밀도함수로 변환 => 분산

input\_data = np.diag(input\_data.T)

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

E = sum(input\_data \* pdf)

output\_value = sum((input\_data - E) \*\*2 \* pdf)

return output\_value

def feature\_7(input\_data):

# 특징 후보 7번 : Diagonal 원소배열 추출 => 0의 개수

input\_data = np.diag(input\_data.T)

cnt=0

for i in input\_data:

if i ==0:

cnt +=1

return cnt

def feature\_8(input\_data):

# 특징 후보 8번 : Anti-Diagonal 원소배열 추출 => 밀도함수로 변환 => 기댓값

input\_data = np.diag(np.fliplr(input\_data))

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

output\_value = sum(input\_data \* pdf)

return output\_value

def feature\_9(input\_data):

# 특징 후보 9번 : Anti-Diagonal 원소배열 추출 => 밀도함수로 변환 => 분산

input\_data = np.diag(np.fliplr(input\_data))

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

E = sum(input\_data \* pdf)

output\_value = sum((input\_data - E) \*\*2 \* pdf)

return output\_value

def feature\_10(input\_data):

# 특징 후보 10번 : Anti-Diagonal 원소배열 추출 => 0의 개수

input\_data = np.diag(np.fliplr(input\_data))

cnt=0

for i in input\_data:

if i ==0:

cnt +=1

return cnt

feature = [0,feature\_1,feature\_2,feature\_3,feature\_4,feature\_5,feature\_6,feature\_7,feature\_8,feature\_9,feature\_10]

Training\_X = np.array([],dtype ='float32') #입력 데이터 가공

Training\_X=np.resize(Training\_X,(0,5))

Training\_Y = np.array([],dtype ='float32')

Training\_Y = np.resize(Training\_X,(0,3))

for i in range(3):

for j in range(1,501):

tmp\_name = f '[배포용] MINIST Data\\{i}\_{j}.csv'

tmp\_img = pd.read\_csv(tmp\_name,header =None).to\_numpy(dtype ='float32')

x0 = feature[2](tmp\_img)

x1 = feature[4](tmp\_img)

x2 = feature[6](tmp\_img)

x3 = feature[7](tmp\_img)

x4 = feature[9](tmp\_img)

X = np.array([[x0,x1,x2,x3,x4]],dtype ='float32')

Y = np.array([[0] \*3])

Y[0][i] =1

Training\_X = np.concatenate((Training\_X,X),axis =0)

Training\_Y = np.concatenate((Training\_Y,Y),axis =0)

#데이터 셔플 및 Train,Test set 구분

Train\_data = np.concatenate([Training\_X,Training\_Y],1)

np.random.shuffle(Train\_data)

Traning\_set = Train\_data[:1200]

Test\_set = Train\_data[1200:]

X,none,Y = np.hsplit(Traning\_set,(5,5))

Network = Neural\_Network(hidden\_layer\_size =4,Input =X,Output =Y,learning\_rate =0.007,Test\_set =Test\_set)

Network.Check\_Input\_Output\_size()

Network.train(5000)

print('W0 :',Network.W0) #학습된 W0,W1 출력

print('W1 :',Network.W1)

plt.xlabel('Epoch')

plt.ylabel("MSE")

plt.grid()

plt.plot(range(len(Network.MSEs)),Network.MSEs,color ='blue')

plt.show() #MSE 변화 출력

plt.xlabel('Epoch')

plt.ylabel("Accuracy")

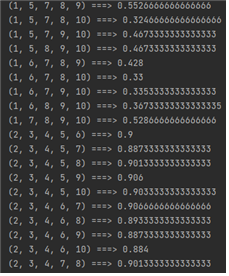
plt.grid()

plt.plot(range(len(Network.Accuaracys)),Network.Accuaracys,color ='blue')

plt.show() #정확도 변화 출력

그렇다면 어떻게 구성해야 잘 나올까??

10개의 후보군들중에서 5개를 골라야 하므로 순열을 활용해서 해볼수도 있을 것이다.

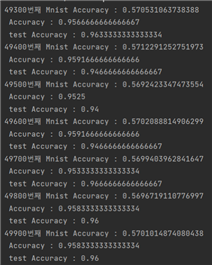


이런 방법은 가장 확실하지만 시간이 굉장히 오래걸린다 (필자는 20시간정도)

아무튼 결과적으로 특성간의 차이가 큰 후보일수록 학습이 잘된다는것을 확인할 수 있었다.

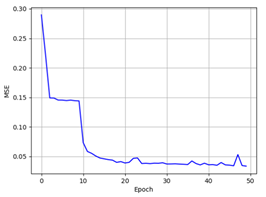
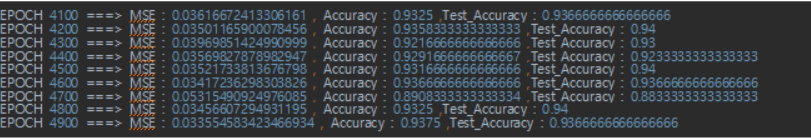
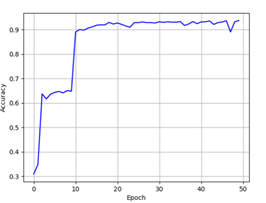
|  |  |  |  |
| --- | --- | --- | --- |
| 은닉층 노드 수 | learning rate | 특성후보 조합 | epoch |
| 5 | 0.001 | 2,3,4,7,10 | 50000 |

위세팅처럼 두었을때 정확도가 가장 잘나왔는데, 96%정도 나왔다.



만약 설정값을 조금 다르게 둔다면

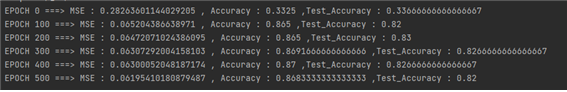
|  |  |  |  |
| --- | --- | --- | --- |
| 은닉층 노드 수 | learning rate | 특성후보 조합 | epoch |
| 4 | 0.007 | 2,3,4,7,10 | 5000 |

MSE정확도

위와같이 Best Case만큼은 아니지만 학습이 잘 된것을 알 수 있다.

만약 은닉층 노드수를 극단적으로 늘리면 어떻게 될까

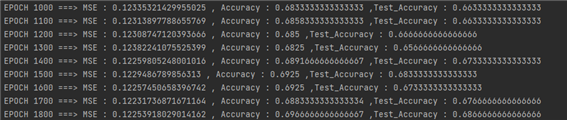
|  |  |  |  |
| --- | --- | --- | --- |
| 은닉층 노드 수 | learning rate | 특성후보 조합 | epoch |
| 40 | 0.007 | 2,3,4,7,10 | 5000 |



우선 시간이 정말 오래걸린다. 은닉층이 많아짐에 따라 계산량이 많아지기때문 또한 이경우 Validation 정확도는 높아지더라도 위와같이 Test\_Accuracy가 낮아지는 Over fitting 현상이 잘 일어나게 된다.

learning rate를 낮게주면 가끔 Local minimum에 걸리는 현상도 관찰할 수 있었다.

|  |  |  |  |
| --- | --- | --- | --- |
| 은닉층 노드 수 | learning rate | 특성후보 조합 | epoch |
| 4 | 0.001 | 2,3,4,7,10 | 5000 |



정확도가 70프로를 넘어가지 않는데, 초기 W에 따라서 같은 설정값으로 해도 90%가까이 될때가 있었다. 즉 Learning rate가 작아 초기 w에 따라서 local minimum 현상이 일어남을 알 수 있다.

**W 저장**

해당 코드를 보면 학습에 따라 W가 엑셀파일로 저장되는것을 알 수 있다.

이를 가져와서 테스트하는 파일을 통해 검사해보자.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import tensorflow\_datasets

import warnings

warnings.filterwarnings('ignore')

class perceptron: # perceptron 클래스 구현

def \_\_init\_\_(self, w):

self.w = w

def output(self, x):

return np.dot(np.append(x, 1), self.w)

def sigmoid(x): # 시그모이드 함수

x=np.float64(x)

return 1 / (1 + np.exp(-x))

def one\_hot\_encoding(array, size):

lst = [] # one\_hot\_encoding 과정

for i in array:

tmp = [0] \* size

tmp[i -1] +=1

lst.append(tmp)

return np.array(lst)

class Neural\_Network: # 2계층 신경망 구현

def \_\_init\_\_(self, hidden\_layer\_size, Input, Output, learning\_rate):

self.hidden\_layer\_size = hidden\_layer\_size # 은닉층 노드 개수

self.Input\_size = Input.shape[1] # 입력층 사이즈

self.Output\_size = Output.shape[1] # 출력층 사이즈

self.X = Input # 트레이닝 셋 입력

self.Y = Output # 트레이닝 셋 출력(정답)

self.learning\_rate = learning\_rate # Learning rate

self.Create\_Weight\_Matrix() # W0,W1매트릭스 임의 생성

def Create\_Weight\_Matrix(self): # Weight 를 만드는 함수

self.W0 = np.random.randn(self.Input\_size +1, self.hidden\_layer\_size)

self.W1 = np.random.randn(self.hidden\_layer\_size +1, self.Output\_size)

def Set\_Hidden\_layer\_Node\_size(self, size): # 히든Node수 설정

self.hidden\_layer\_size = size

self.Create\_Weight\_Matrix()

def Check\_Input\_Output\_size(self): # Input,Output 체크함수

print('Input 속성 수 ====>', self.Input\_size)

print('Output 속성 수 ===>', self.Output\_size)

def predict(self, x): # y예측 함수

INPUT\_LAYER = perceptron(self.W0)

OUTPUT\_LAYER = perceptron(self.W1)

self.sigmoid\_input = sigmoid(INPUT\_LAYER.output(x))

self.H = np.append(self.sigmoid\_input, 1)

return sigmoid(OUTPUT\_LAYER.output(self.sigmoid\_input))

def Back\_propagation(self):

lr =self.learning\_rate # Learning rate

for i in range(len(self.X)):

Y\_pred =self.predict(self.X[i]) # Y 예측값

Input = np.append(self.X[i], 1) # 입력층 + 1 추가

# 역전파를 1단계부터 시행하면 W가 업데이트되어 2단계부터 시행

for j in range(self.Input\_size +1): # 역전파 2단계

for k in range(self.hidden\_layer\_size):

Etotal\_h\_diff =0

for q in range(self.Output\_size):

Etotal\_h\_diff +=-2 \* (self.Y[i][q] - Y\_pred[q]) \* Y\_pred[q] \* (1 - Y\_pred[q]) \*self.W1[k][q]

h\_z\_diff =self.H[k] \* (1 -self.H[k])

z\_w\_diff = Input[j]

Etotal\_w = Etotal\_h\_diff \* h\_z\_diff \* z\_w\_diff

self.W0[j][k] =self.W0[j][k] - lr \* Etotal\_w # W0업데이트

for j in range(self.hidden\_layer\_size +1): # 역전파 1단계

for k in range(self.Output\_size):

E\_o\_diff =-2 \* (self.Y[i][k] - Y\_pred[k])

o\_z\_diff = Y\_pred[k] \* (1 - Y\_pred[k])

z\_w\_diff =self.H[j]

Etotal\_w = E\_o\_diff \* o\_z\_diff \* z\_w\_diff

self.W1[j][k] =self.W1[j][k] - lr \* Etotal\_w # W1 업데이트

def train(self, epoch):

self.epoch = epoch # epoch 저장

self.MSEs = [] # MSE그래프를 그리기 위함

self.Accuaracys = [] # 정확도 그래프를 그리기 위함

for i in range(epoch):

data = np.concatenate([self.X, self.Y], 1) # 셔플과정

np.random.shuffle(data) # 매 에폭마다 섞어주는 과정

self.X, none, self.Y = np.hsplit(data, (self.Input\_size, self.Input\_size))

self.Back\_propagation() # 역전파 과정으로 W업데이트

if i % 100 ==0:

tmp\_mse = [] # 정확도 계산

cnt =0

for j in range(len(self.X)):

Y\_pred =self.predict(self.X[j])

tmp\_mse.append(np.mean((self.Y[j] - Y\_pred) \*\*2))

maxindex = np.argmax(self.predict(self.X[j])) # 가장큰 index가져오기

tmp = np.array([0] \*self.Output\_size)

tmp[maxindex] =1

if np.array\_equal(tmp, self.Y[j]): # 정답과 비교

cnt +=1

Accuracy = cnt /len(self.X)

self.Accuaracys.append(Accuracy)

MSE = np.mean(tmp\_mse)

self.MSEs.append(MSE)

testcnt =0

for j in range(len(data\_x)):

Y\_pred = Network.predict(data\_x[j])

maxindex = np.argmax(Y\_pred) # 가장큰 index가져오기

tmp = np.array([0] \*3)

tmp[maxindex] =1

if np.array\_equal(tmp, label\_list[j]): # 정답과 비교

testcnt +=1

testAccuracy = testcnt /len(data\_x)

print(f 'EPOCH {i} ===> MSE : {MSE} , Accuracy : {Accuracy} testAccuracy : {testAccuracy}')

df0 = pd.DataFrame(self.W0)

df0.to\_csv(f'W\\{i}epoch\_W0.csv',index =False)

df1 = pd.DataFrame(self.W1)

df1.to\_csv(f'W\\{i}epoch\_W1.csv', index =False)

def InputWs(self,W0,w1):

self.W0 = W0

self.W1 = w1

#####################################################################################

def feature\_1(input\_data):

# 특징 후보 1번 : 가로축 Projection => 확률밀도함수로 변환 => 기댓값

X = sum(input\_data)

input\_data = input\_data.T

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

output\_value = sum(X \* pdf)

return output\_value

#####################################################################################

def feature\_2(input\_data):

# 특징 후보 2번 : 가로축 Projection => 확률밀도함수로 변환 => 분산

X = sum(input\_data)

input\_data = input\_data.T

S = sum(X)

pdf = np.array([ sum(input\_data[i])/S for i in range(len(input\_data)) ])

E = sum(X \* pdf)

output\_value = sum( (X - E)\*\*2 \* pdf)

return output\_value

#####################################################################################

def feature\_3(input\_data):

# 특징 후보 3번 : 세로축 Projection => 확률밀도함수로 변환 => 기댓값

X = sum(input\_data.T)

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

output\_value = sum(X \* pdf)

return output\_value

#####################################################################################

def feature\_4(input\_data):

# 특징 후보 4번 : 세로축 Projection => 확률밀도함수로 변환 => 분산

X = sum(input\_data.T)

input\_data = input\_data

S = sum(X)

pdf = np.array([sum(input\_data[i]) / S for i in range(len(input\_data))])

E = sum(X \* pdf)

output\_value = sum((X - E) \*\*2 \* pdf)

return output\_value

def feature\_5(input\_data):

# 특징 후보 5번 : Diagonal 원소배열 추출 => 밀도함수로 변환 => 기댓값

input\_data = np.diag(input\_data.T)

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

output\_value = sum(input\_data \* pdf)

return output\_value

def feature\_6(input\_data):

# 특징 후보 6번 : Diagonal 원소배열 추출 => 밀도함수로 변환 => 분산

input\_data = np.diag(input\_data.T)

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

E = sum(input\_data \* pdf)

output\_value = sum((input\_data - E) \*\*2 \* pdf)

return output\_value

def feature\_7(input\_data):

# 특징 후보 7번 : Diagonal 원소배열 추출 => 0의 개수

input\_data = np.diag(input\_data.T)

cnt=0

for i in input\_data:

if i ==0:

cnt +=1

return cnt

def feature\_8(input\_data):

# 특징 후보 8번 : Anti-Diagonal 원소배열 추출 => 밀도함수로 변환 => 기댓값

input\_data = np.diag(np.fliplr(input\_data))

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

output\_value = sum(input\_data \* pdf)

return output\_value

def feature\_9(input\_data):

# 특징 후보 9번 : Anti-Diagonal 원소배열 추출 => 밀도함수로 변환 => 기댓값

input\_data = np.diag(np.fliplr(input\_data))

S = sum(input\_data)

pdf = np.array([input\_data[i] / S for i in range(len(input\_data))])

E = sum(input\_data \* pdf)

output\_value = sum((input\_data - E) \*\*2 \* pdf)

return output\_value

def feature\_10(input\_data):

# 특징 후보 10번 : Anti-Diagonal 원소배열 추출 => 0의 개수

input\_data = np.diag(np.fliplr(input\_data))

cnt=0

for i in input\_data:

if i ==0:

cnt +=1

return cnt

feature = [0,feature\_1,feature\_2,feature\_3,feature\_4,feature\_5,feature\_6,feature\_7,feature\_8,feature\_9,feature\_10]

# raw\_data = pd.read\_csv(f'[배포용] MINIST Data\\0\_1.csv',encoding='utf-8', engine = 'python',header=None).to\_numpy()

# for i in range(1,11):

# print(i,':' , feature[i](raw\_data))

##########mnist

mnist = tensorflow\_datasets.load('mnist')

train\_set = mnist["train"]

# (train\_images, train\_labels), (test\_images, test\_labels) = mnist

data\_list = []

label\_list = []

one = []

two = []

zero=[]

for d in train\_set:

if d["label"] ==0:

zero.append(d["image"])

elif d["label"] ==1:

one.append(d["image"])

elif d["label"] ==2:

two.append(d["image"])

data\_list = zero + one + two

data\_list = np.array(data\_list)

print(data\_list.shape)

label\_list = []

one\_index =len(one)

two\_index =len(two)

zero\_index =len(zero)

print(one\_index,two\_index,zero\_index)

t = [[1,0,0], [0,1,0], [0,0,1]]

for i in range(zero\_index):

label\_list.append(t[0])

for i in range(one\_index):

label\_list.append(t[1])

for i in range(two\_index):

label\_list.append(t[2])

label\_list = np.array(label\_list)

data\_x=np.array([])

data\_x=np.resize(data\_x,(0,5))

for d in data\_list:

d=d.T[0].T

x0 = feature[2](d)

x1 = feature[3](d)

x2 = feature[4](d)

x3 = feature[7](d)

x4 = feature[10](d)

X = np.array([[x0,x1,x2,x3,x4]],dtype ='float32')

data\_x = np.concatenate((data\_x,X),axis =0)

print(data\_x.shape)

Training\_X = np.array([],dtype ='float64')

Training\_X=np.resize(Training\_X,(0,5))

Training\_Y = np.array([],dtype ='float64')

Training\_Y = np.resize(Training\_X,(0,3))

for i in range(3):

for j in range(1,501):

tmp\_name = f '[배포용] MINIST Data\\{i}\_{j}.csv'

tmp\_img = pd.read\_csv(tmp\_name,header =None).to\_numpy(dtype ='float32')

x0 = feature[2](tmp\_img)

x1 = feature[3](tmp\_img)

x2 = feature[4](tmp\_img)

x3 = feature[7](tmp\_img)

x4 = feature[10](tmp\_img)

X = np.array([[x0,x1,x2,x3,x4]],dtype ='float32')

Y = np.array([[0] \*3])

Y[0][i] =1

Training\_X = np.concatenate((Training\_X,X),axis =0)

Training\_Y = np.concatenate((Training\_Y,Y),axis =0)

Train\_data = np.concatenate([Training\_X,Training\_Y],1)

np.random.shuffle(Train\_data)

Traning\_set = Train\_data[:1200]

Test\_set = Train\_data[1200:]

X,none,Y = np.hsplit(Traning\_set,(5,5))

TX,none,TY = np.hsplit(Test\_set,(5,5))

Network = Neural\_Network(hidden\_layer\_size =4,Input =Training\_X,Output =Training\_Y,learning\_rate =0.007)

Network.Check\_Input\_Output\_size()

# Network.train(10000)

for i in range(100,50000,100):

w\_hidden = pd.read\_csv(f 'W\_234710\\{i}epoch\_W0.csv').to\_numpy()

w\_output = pd.read\_csv(f 'W\_234710\\{i}epoch\_W1.csv').to\_numpy()

Network.InputWs(w\_hidden, w\_output)

# 정확도 계산

cnt =0

for j in range(len(data\_x)):

Y\_pred = Network.predict(data\_x[j])

maxindex = np.argmax(Y\_pred) # 가장큰 index가져오기

tmp = np.array([0] \*3)

tmp[maxindex] =1

if np.array\_equal(tmp, label\_list[j]): # 정답과 비교

cnt +=1

Accuracy = cnt /len(data\_x)

print(f '{i}번째 Mnist Accuracy : {Accuracy} ')

cnt =0

for j in range(len(X)):

Y\_pred = Network.predict(X[j])

maxindex = np.argmax(Y\_pred) # 가장큰 index가져오기

tmp = np.array([0] \*3)

tmp[maxindex] =1

if np.array\_equal(tmp, Y[j]): # 정답과 비교

cnt +=1

Accuracy = cnt /len(X)

print(f ' Accuracy : {Accuracy} ')

cnt =0

for j in range(len(TX)):

Y\_pred = Network.predict(TX[j])

maxindex = np.argmax(Y\_pred) # 가장큰 index가져오기

tmp = np.array([0] \*3)

tmp[maxindex] =1

if np.array\_equal(tmp, TY[j]): # 정답과 비교

cnt +=1

Accuracy = cnt /len(TX)

print(f ' test Accuracy : {Accuracy} ')

해당 코드를 확인하면 저장된 w들을 가져와서 테스트셋과 비교해서 정확도를 다시 알아볼 수도 있고, 실제 Minist 데이터를 18000개정도 가져와서 형태를 맞춰 본 후에 정확도를 검사할 수도 있다.

학습이 90%넘게 되었지만 해당 데이터셋의 한계로 실제 Minist 에선 60%정도 정확도가 나오면 학습이 잘 된것이다!