Multi-Layer Perceptron (MLP) classifier

HW4

7109056202 資工碩一 陳彥儒

I. 目的

本次作業為設計Multi-Layer Perceptron分類器來分辨我們作業一自己造的data set。

II. 實作

a. Load Data

一開始先把在作業一已經分好的訓練集跟測試集載入進來，那他們的資料組成為: 訓練集是兩個class的前5000筆組成，測試集為兩個class的後5000筆組成。程式碼如下:

```
if __name__ =="__main__":
    #load data
    train_data = np.load('./data/train_data.npy')
    test_data  = np.load('./data/test_data.npy')
```

b. Load Data

因為我們要告訴電腦我們餵進去的data是哪一個類別的，所以我們需要為data設label。那class 1為0，class 2為 1。
因為我們希望model訓練出來為二維的結果，就是它會顯示class 1的機率是多少，class2的機率為多少，所以我們在這邊train_label、test_label都是二維的。我以train_label舉例，

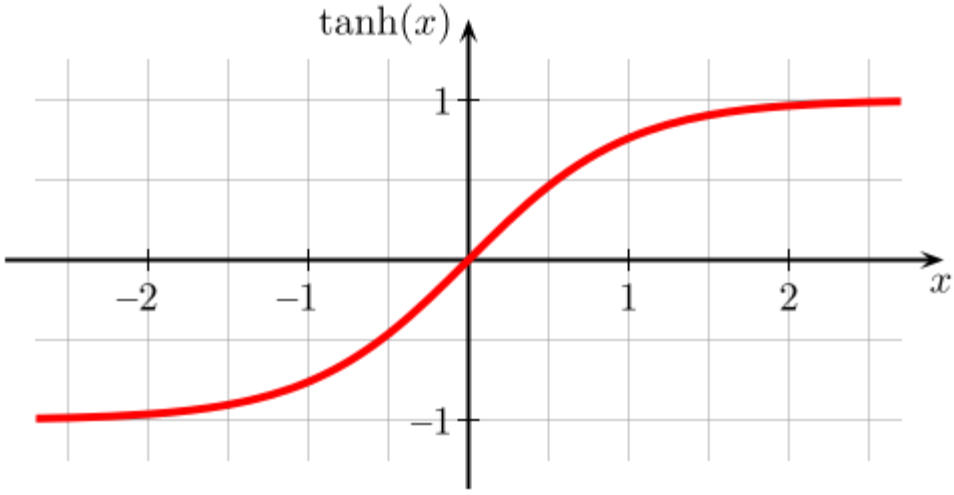
```
train_label
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

可以看到訓練集因為前5000筆是class 1 的data，所以都為[1 , 0]，那後5000筆是class2的data，所以都為[0 , 1]。程式碼如下:

```
def label_data():
    class_1 = [[0]]*5000
    class_2 = [[1]]*5000

    train_label = np.concatenate((class_1,class_2),axis=0)
    test_label = np.concatenate((class_1,class_2),axis=0)
    train_label = to_categorical(train_label,num_classes=2)
    test_label = to_categorical(test_label,num_classes=2)
    print('train_label')
    print(train_label)
    return train_label,test_label
```

c. Build model

首先，我總共建了四層的model(含輸出層)，每一層的activation都是使用tanh。tanh是個常用於分類問題的activation function，輸出範圍介於[-1, 1]，輸出範圍會有正有負，也是個嚴格遞增函數，他的微分是f'(x) = 1 - f^2(x)。



層與層之間都會加BatchNormalization和Dropout，加入這兩個function都是為了避免訓練結果overfitting。overfitting就是看似訓練資料的預測結果很好，可是實際用測試資料的預測結果卻很差。Loss function選擇BinaryCrossentropy，Binary Crossentropy常用於『二元分類』，BinaryCrossentropy公式為

$$\text{BCELoss}(\text{O},\text{T}) = -\frac{1}{n}\sum_{i}\left((T[i] * \log(O[i])) + (1 - T[i]) * \log(1 - O[i])\right)$$

，因為我們的輸出結果只有兩類，所以適合用這個loss function。optimizer選擇adam。model.fit的參數調整就用以下表格呈現:

batch_size	32
epochs	500
shuffle	True
validation_split	0.2

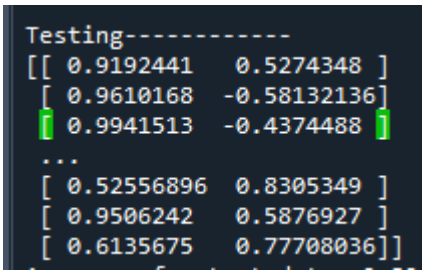
以下為整個model的截圖:

```
model=Sequential()
#layer 1
model.add(Dense(256, input_dim=50, activation='tanh'))
model.add(BatchNormalization())
model.add(Dropout(0.25))
#layer 2
model.add(Dense(64, activation='tanh'))
model.add(BatchNormalization())
model.add(Dropout(0.25))
#layer 3
model.add(Dense(32, activation='tanh'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(2,activation='tanh'))


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

history = model.fit(train_data, train_label, batch_size=24, epochs=500,
                    shuffle=True,
                    validation_split=0.2)
```

d. predict model

接下來就是用測試集資料來看我們的model成效。我們會用到model.predict()，把test_data丟進去，出來的結果為二維矩陣

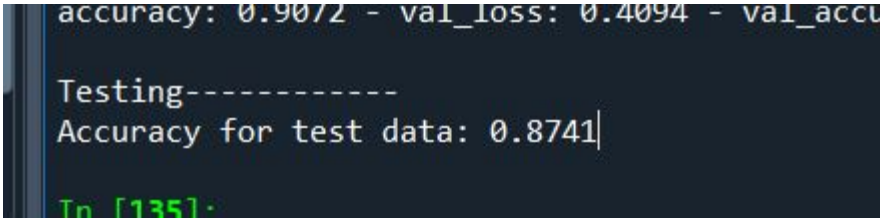
，如右圖:



，可以看到我們第一筆資料判給class 1的機率為91%，而判給class 2的機率為52%，因為class 1的機率比較大，那我就是把第一筆資料判給class 1並且把第一筆資料改成[1 , 0]，這樣就可以方便跟我們test_data的label來逐一做比對，看哪筆資料預測錯，進而來算我們的準確率。以下為程式碼截圖:

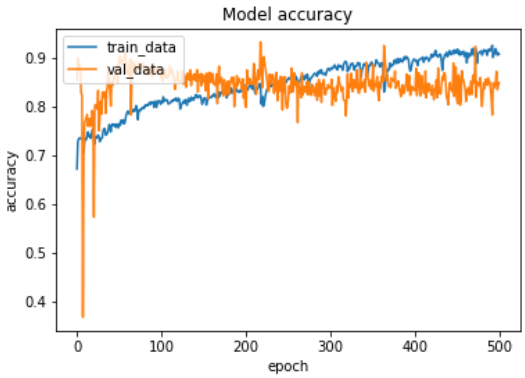
```
print('\nTesting----------')
y_pred = model.predict(test_data)
print(y_pred)
for i in range(10000):
    if y_pred[i][0]>y_pred[i][1] :
        y_pred[i][0]=1
        y_pred[i][1]=0
    elif y_pred[i][0]<y_pred[i][1]:
        y_pred[i][0]=0
        y_pred[i][1]=1

counter = 0
for i in range(10000):

    if y_pred[i][0]!=test_label[i][0] or y_pred[i][1]!=test_label[i][1]:
        counter+=1
```

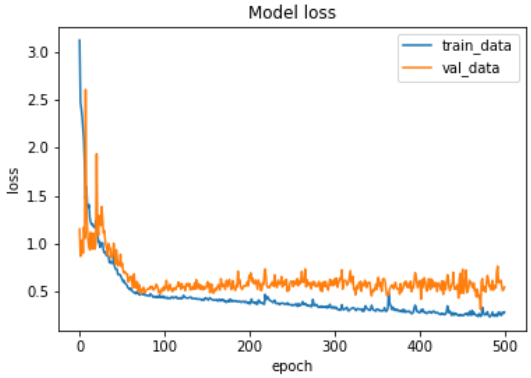
最後可以看到我們的準確率來到:87.41%



e. draw picture

那我透過matplotlib.pyplot來繪製訓練模型中train data、val_data的accurancy和loss。以下為結果圖:





透過圖我們可以看到val_data在訓練初期的準確率不好，那loss也是非常高，透過我們不停訓練模型後val_data的準確率穩定在80%-90%之間，loss也進入穩定。

III. 討論

那這次的作業也是非常有趣，自己Build model來預測分類我們的兩種資料。不過在建model時真的需要很多domain knowledge，像是最基本的每層的神經元個數就是一個課題，本來是嘗試漸增方式來處理，可是準確率只有79%左右，後來是換漸減方式使得準確率有效提升。如果未來想要再讓model預測準確率提升，還需要再多多吸收相關知識和花時間再去train model。

謝謝老師的教導和助教辛苦的批閱。

```python
import os
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import time

from  tensorflow.keras.models import Sequential
from  tensorflow.keras.layers import Dense, SpatialDropout2D, Dropout, Flatten
from  tensorflow.keras.layers import Conv2D, MaxPooling2D,BatchNormalization
from  tensorflow.keras.optimizers import SGD,RMSprop
from  tensorflow.keras.callbacks import EarlyStopping
from  tensorflow.keras.losses import BinaryCrossentropy
from  tensorflow.keras.utils import to_categorical
from  tensorflow.keras.optimizers import Adam

def label_data():
    class_1 = [[0]]*5000
    class_2 = [[1]]*5000

    train_label = np.concatenate((class_1,class_2),axis=0)
    test_label = np.concatenate((class_1,class_2),axis=0)
    train_label = to_categorical(train_label,num_classes=2)
    test_label = to_categorical(test_label,num_classes=2)
    print('train_label')
    print(train_label)
    return train_label,test_label
def loss_pic(History):
    history = History

    history = History
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train_data', 'val_data'], loc='upper left')

    timestr = time.strftime("%Y%m%d_%H%M%S")
    plt.savefig('./Model accuracy_{}.png'.format(timestr))
    plt.cla()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train_data', 'val_data'], loc='upper right')
    timestr = time.strftime("%Y%m%d_%H%M%S")
    plt.savefig('./Model loss_{}.png'.format(timestr))
    plt.cla()
    #timestr = time.strftime("%Y%m%d_%H%M%S")


```

```python
def Build_model(train_data, test_data, train_label, test_label):
    model=Sequential()
    #layer 1
    model.add(Dense(256, input_dim=50, activation='tanh'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    #layer 2
    model.add(Dense(64, activation='tanh'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    #layer 3
    model.add(Dense(32, activation='tanh'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(2,activation='tanh'))


    model.compile(loss='BinaryCrossentropy', optimizer='adam',metrics=['accuracy'])
    model.summary()
    #24 85%
    #32 87%
    #48 86%
    history = model.fit(train_data, train_label, batch_size=32, epochs=500,
                        shuffle=True,
                        validation_split=0.2)
    History = history

    #model.evaluate(test_data,test_label, batch_size=24)
    #predict
    print('\nTesting------------')
    y_pred = model.predict(test_data)
    print(y_pred)
    for i in range(10000):
        if y_pred[i][0]>y_pred[i][1] :
            y_pred[i][0]=1
            y_pred[i][1]=0
        elif y_pred[i][0]<y_pred[i][1]:
            y_pred[i][0]=0
            y_pred[i][1]=1

    counter = 0
    for i in range(10000):

        if y_pred[i][0]!=test_label[i][0] or y_pred[i][1]!=test_label[i][1]:
            counter+=1


    print('Accuracy for test data:',(10000-counter)/10000)
    #draw loss picture
    loss_pic(History)
    #timestr = time.strftime("%Y%m%d_%H%M%S")
    #model.save('./model/model_{}.h5'.format(timestr))
```

```python
if __name__ =="__main__":
    #load data
    train_data = np.load('./data/train_data.npy')
    test_data  = np.load('./data/test_data.npy')
    print(train_data[0].shape)
    #label
    train_label,test_label = label_data()
    #Build model

    Build_model(train_data ,test_data,train_label,test_label)
```