ECT HW9

前置處理

```
%tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

➔ 先把 tensorflow 版本指定為 1.x，因為目前預設的 2 版有一些問題

```
from keras.datasets import fashion_mnist
from keras.utils import np_utils
import matplotlib.pyplot as plt
(x_train,y_train),(x_test,y_test) = fashion_mnist.load_data()
```

➔ 把資料即從 fashion_mnist 中下載下來，並放進 traing、tesing 的變數中

```
print(x_train.shape)
print(y_train.shape)
```

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

➔ 用 Shape 可以觀察出資料類型為 28*28 的圖片

```
x_train = x_train.reshape(x_train.shape[0], 28*28).astype('float32')
x_test = x_test.reshape(x_test.shape[0], 28*28).astype('float32')
# normalize
x_train = x_train / 255
x_test = x_test / 255
```

➔ 把 input 的 28 * 28 維度的圖片，用 reshape 轉成 1 維的 784，並且同除以 255 來當作正規化 (因為每個點的值為 0~255)

```
# one-hot encodeing
y_train_categorical = np_utils.to_categorical(y_train)
y_test_categorical = np_utils.to_categorical(y_test)
```

➔ Testing data 則是進行 one-hot encoding，因為共有 10 個類別，所以除了類別的位置是 1，其他的位置會用 0 表示，變成 10 維

```
print(x_train.shape)
print(y_train.shape)
print(y_train_categorical.shape)
```

```
(60000, 784)
(60000,)
(60000, 10)
```

➔ 可以印出來檢查，input 的確變成 1 維，output 也變成 10 維

```
# CNN的input shape要重新調整
x_train_cnn = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')
x_test_cnn = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32')
```

➔ CNN 的 input 則要重新 reshape 成適當的形式

(a)

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from keras.optimizers import Adam

model = Sequential()
model.add(Dense(input_dim = 28*28, units = 512, activation="relu"))
model.add(BatchNormalization())
model.add(Dense(512, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(256, activation="relu"))
model.add(BatchNormalization())
model.add(Dense(256, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(10,activation='softmax'))

opti = Adam(lr=0.001,decay=0, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(loss='categorical_crossentropy', optimizer=opti, metrics=['accuracy'])
```

➔ 建立 NN 模型，共有 5 層 Dense

```
filter_size = 128
model_CNN = Sequential()
model_CNN.add(Conv2D(filters=filter_size, input_shape=(28,28,1), kernel_size=(3,3), strides = (1, 1), activation="relu"))
model_CNN.add(BatchNormalization(axis=1))
model_CNN.add(Conv2D(filter_size, (3,3), activation="relu"))
model_CNN.add(BatchNormalization(axis=1))
model_CNN.add(Dropout(0.3))

model_CNN.add(Conv2D(2*filter_size, (3,3), activation="relu"))
model_CNN.add(BatchNormalization(axis=1))
model_CNN.add(Conv2D(2*filter_size, (3,3), activation="relu"))
model_CNN.add(BatchNormalization(axis=1))
model_CNN.add(MaxPooling2D((2,2)))
model_CNN.add(Dropout(0.3))

model_CNN.add(Flatten())
model_CNN.add(Dense(256, activation="relu"))
model_CNN.add(BatchNormalization())
model_CNN.add(Dense(256, activation="relu"))
model_CNN.add(BatchNormalization())
model_CNN.add(Dropout(0.3))
model_CNN.add(Dense(output_dim=10, activation="softmax"))
opti = Adam(lr=0.001,decay=0, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model_CNN.compile(loss='categorical_crossentropy', optimizer=opti, metrics=['accuracy'])
```

➔ 建立 CNN 模型，共有一堆層...太多了

(b)

參數意義 For NN :

```
model = Sequential()
```

➔ 初始化 model

```
model.add(Dense(input_dim = 28*28, units = 512, activation="relu"))
```

➔ Add 函數用來增加新的層，Dense 就是最普通的神經元

- Input_dim: 代表 input 的 shape，可以對比前處理時的 784
- Units: 代表神經元的數量
- Activation: 代表經過 Sum(wx + b)之後要用哪個函數來進行非線性激發

```
model.add(BatchNormalization())
```

➔ 這是用來使每次神經元 output 之後，在 input 進下一層之前進行正規化

```
model.add(Dropout(0.3))
```

➔ 代表這一層訓練完之後，有 0.3 的 output 被拋棄，可以讓結果更加

General，有助於提高 test 的準確度，但 train 的準確度會暫時下降

```
model.add(Dense(10,activation='softmax'))
```

➔ 這一層是我的最後一層，也就是真正的 output。

- 使用 softmax 函數可使 output 總合為 1。他是用每一個 output 總合當

  分母，各自 output 當分子來運算的，因此適合用於分類問題

```
opti = Adam(lr=0.001,decay=0, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

➔ 這是設定 optimizer，Adam 是一種可以自動調整 learning rate 的方式

```
model.compile(loss='categorical_crossentropy', optimizer=opti, metrics=['accuracy'])
```

➔ Compile 會把上面的設定都編譯起來

■ Loss 就是 loss funciton，metrics 代表要印出的資訊

```
train_history = model.fit(x_train, y_train_categorical, batch_size=64, epochs=50, validation_data=(x_test, y_test_categorical))
```

➔ Fit 就是真的開始訓練

■ 前兩個參數就是 training 的 input、output

■ Batch_size: 代表每次要訓練幾筆資料

■ Epochs: 代表要訓練幾回合

■ Validation_data: 用自己準備的 data 當作 Validation data，若用 validation_split 則會從 training data 中切割

```
model.summary()
```

➔ 可以看整個 NN 大致有哪些參數

參數意義 For CNN:

```
filter_size = 128
```

➔ 這代表 CNN 中 filter 的數量，我初始化最小的數量之後都用這個的倍數

```
model_CNN = Sequential()
```

➔ 初始化模型

```
model_CNN.add(Conv2D(filters=filter_size, input_shape=(28,28,1), kernel_size=(3,3), strides = (1, 1), activation="relu"))
```

➔ CNN 其實跟 NN 一樣，只是中間的隱藏層是使用 Conv2D (Convolutional layer)也就是卷積層

■ Filters: 代表 filter 有多少個，用來找特徵

■ Input_shape: 代表 input 的 shape 長怎樣，每張圖片都是 28 * 28 的 pixel，每種圖片只有單 1 色調，因此 shape = (28,28,1)

- Kernel_size: 代表抓特徵時，要用多大的 filter 來抓取，我用 3 X 3
- Strides: 代表每次 filter 滾動的步數，(1, 1)就是指滾動一步
- Activation: 就是 activation function

```
model_CNN.add(BatchNormalization(axis=1))
```

➔ 這跟 NN 一樣，用來正規化 output 再成為下一層的 input

```
model_CNN.add(Dropout(0.3))
```

➔ 一樣有 Dropout，來拋棄部分 output

```
model_CNN.add(MaxPooling2D((2,2)))
```

➔ 這會縮小圖片，Max 代表會抓取選定範圍內最大的值來代表它，其餘皆捨棄，我設定(2,2)就會從 2 X 2 的方格中，選一個最大的保留，其餘 3 個捨棄

```
model_CNN.add(Flatten())
```

➔ 用來攤平卷積後的結果，為了當作普通 Dense 的 input

```
model_CNN.add(Dense(256, activation="relu"))
model_CNN.add(BatchNormalization())
```

➔ Dense 部分跟 NN 相同，不多作介紹

```
model_CNN.add(Dense(output_dim=10, activation="softmax"))
```

➔ Output 也是跟 NN 相同

```
opti = Adam(lr=0.001,decay=0, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model_CNN.compile(loss='categorical_crossentropy', optimizer=opti, metrics=['accuracy'])
```

➔ 優化器和編譯部分也跟 NN 相同，不加以贅述

```
train_history_cnn = model_CNN.fit(x_train_cnn, y_train_categorical, batch_size=64, epochs=50, validation_data=(x_test_cnn, y_test_categorical))
```

➔ 然後一樣用 fit 丟進去訓練

```
model_CNN.summary()
```

➔ 用 summary 可以看各種參數和各層的關係

(c)

For NN:

```
scores = model.evaluate(x_test, y_test_categorical)
scores[1]
```

```
10000/10000 [==============================] - 1s 76us/step
0.8840000033378601
```

➔ 使用 evaluate 函數，把 input、output 都換成 testing data 放進去做評估，可以發現準確度約為 88.4%

For CNN:

```
scores = model_CNN.evaluate(x_test_cnn, y_test_categorical)
scores[1]
```

```
10000/10000 [==============================] - 3s 264us/step
0.9294999837875366
```

➔ 整體準確度上升至 92.9%，比起 NN 進步許多

(d)

For NN:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                2570
=================================================================
Total params: 870,410
Trainable params: 867,338
Non-trainable params: 3,072
```

➔ Model: 代表它的名字

➔ 然後下面就是各層的資訊，總參數量、訓練參數量、為訓練參數量

For CNN:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 128)       1280
_________________________________________________________________
batch_normalization_5 (Batch (None, 26, 26, 128)       104
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 128)       147584
_________________________________________________________________
batch_normalization_6 (Batch (None, 24, 24, 128)       96
_________________________________________________________________
dropout_3 (Dropout)          (None, 24, 24, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 256)       295168
_________________________________________________________________
batch_normalization_7 (Batch (None, 22, 22, 256)       88
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 20, 20, 256)       590080
_________________________________________________________________
batch_normalization_8 (Batch (None, 20, 20, 256)       80
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 256)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 10, 10, 256)       0

_________________________________________________________________
flatten_1 (Flatten)          (None, 25600)             0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               6553856
_________________________________________________________________
batch_normalization_9 (Batch (None, 256)               1024
_________________________________________________________________
dense_7 (Dense)              (None, 256)               65792
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                2570
=================================================================
Total params: 7,658,746
Trainable params: 7,657,538
Non-trainable params: 1,208
```

➔ 參數意義同 NN 所提，可以注意因為是第二個用 Sequential 創建的 Model，因此名字也有變化了

(e)

```
def show_train_history(train_history, train, validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train History')
    plt.ylabel('train')
    plt.xlabel('Epoch')
    plt.legend(['train', 'validation'])
    plt.show()
```

➔ 使用此函數還繪製圖片

For NN:

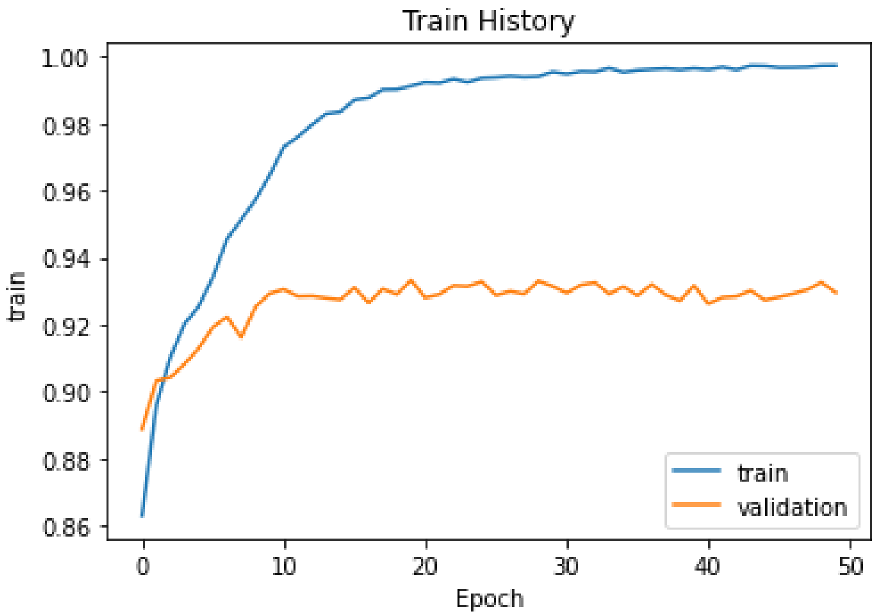
```
show_train_history(train_history, 'accuracy','val_accuracy')
```

➔ 可以看到隨著 Epoch 上升，training 的準確度也越來越高，但 validation 卻不久就開始浮動了。

- 這其實跟我之前提到的現象一樣，Validation 不用等到 50 個 Epoch，就已經訓練得差不多了，開始浮動了

For CNN:

```
show_train_history(train_history_cnn, 'accuracy','val_accuracy')
```

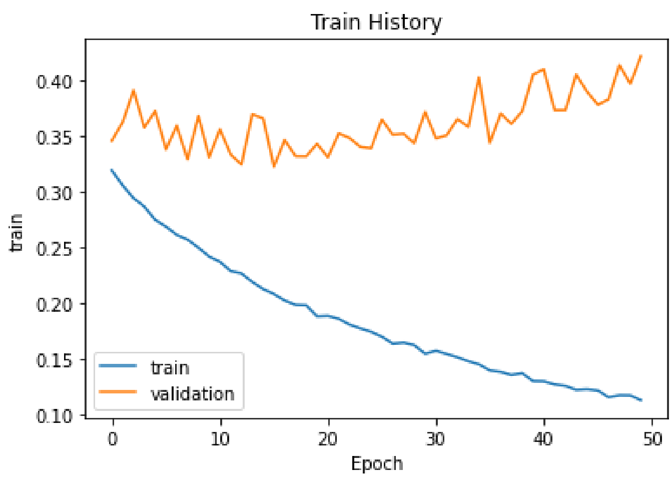


➔ 在 training 部分比使用 NN 訓練得更為完整了，Validation 部分準確度也更高了，但 Validation 部分還是很快就開始浮動了

(f)

For NN:

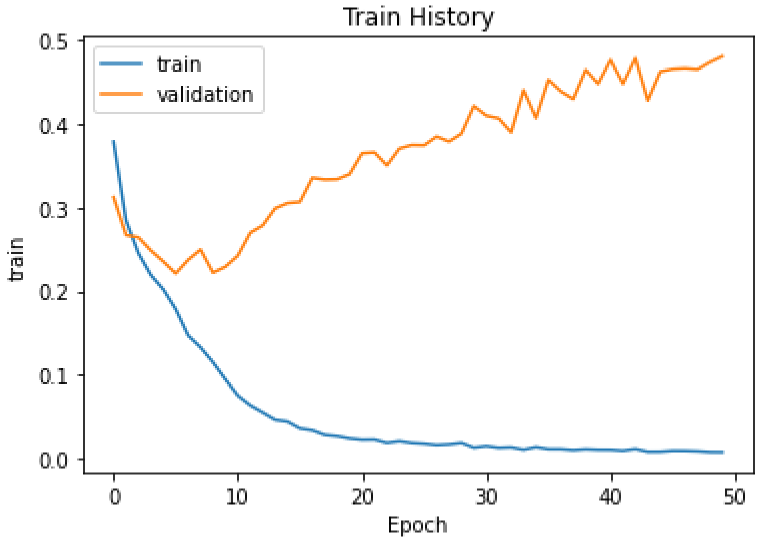
```
show_train_history(train_history, 'loss','val_loss')
```



➔ 如圖所示 Loss 會隨著訓練而下降，跟 Accuracy 相反

For CNN:

```
show_train_history(train_history_cnn, 'loss','val_loss')
```



(g)

For NN:

```
import pandas as pd
prediction = model.predict_classes(x_test)

pd.crosstab(y_test, prediction, rownames=['label'], colnames=['predict'])
```

predict	0	1	2	3	4	5	6	7	8	9
label										
0	817	1	11	17	0	1	148	0	5	0
1	7	968	0	15	2	0	7	0	1	0
2	17	0	796	12	78	0	95	0	2	0
3	30	6	6	887	19	0	48	0	4	0
4	6	2	75	47	773	0	91	0	6	0
5	0	0	0	0	0	963	0	19	3	15
6	121	1	59	20	44	0	743	0	12	0
7	0	0	0	0	0	4	0	954	0	42
8	4	1	0	3	3	1	6	3	979	0
9	1	0	0	0	0	4	0	35	0	960

➔ 混淆矩陣如上圖所示，大多在斜直線上，代表預測準確度高

For CNN:

```
import pandas as pd
prediction = model_CNN.predict_classes(x_test_cnn)

pd.crosstab(y_test, prediction, rownames=['label'], colnames=['predict'])
```

predict label	0	1	2	3	4	5	6	7	8	9
0	886	1	16	9	2	1	81	0	4	0
1	1	987	0	7	2	0	1	0	2	0
2	17	2	905	4	32	0	40	0	0	0
3	13	1	10	911	16	0	48	0	1	0
4	0	0	33	18	877	0	71	0	1	0
5	0	0	0	0	0	989	0	8	0	3
6	99	2	29	16	36	0	808	0	10	0
7	0	0	0	0	0	4	0	973	0	23
8	3	1	1	5	0	2	2	1	985	0
9	0	0	1	0	0	5	0	20	0	974

➔ 更多數字集中在斜直線上了，預測更準確