LAB4-MLP

國立雲林科技大學
夏世昌 特聘教授 / 電子工程系
王斯弘 助理教授 / 前瞻學位學士學程

2021, Fall Semester

Import library

```
import tensorflow as tf
tf.__version__
```

```
'2.0.0'
```

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
```

dataset

from sklearn.model_selection import train_test_split

train_test_split()是sklearn.model_selection中的分離器函式，用於將陣列或矩陣劃分為訓練集和測試集

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

函式

sklearn.model_selection中train_test_split()

函式樣式為： X_train, X_test, y_train, y_test = train_test_split(train_data, train_target, test_size, random_state，shuffle)

dataset

```
iris = datasets.load_iris()
X = iris.data
Y = iris.target
```

資料處理

```
dataset = []
for index, x in enumerate(X):
    x = [x[0],x[1], x[2],x[3]]#取X[0]和X[1]和X[2]和X[3]的資料
    dataset.append((x))
dataset = np.array(dataset)
X_train,X_test,y_train,y_test= train_test_split(dataset,Y,test_size=0.3,random_state=0)
```

資料處理

```
y_train = tf.keras.utils.to_categorical(y_train,3)
y_test = tf.keras.utils.to_categorical(y_test,3)
```

函式

tf.keras.utils.to_categorical()

Converts a class vector (integers) to binary class matrix.

https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical

定義Model架構

```
def mlp(x):
    x = tf.keras.layers.Dense(8)(x)
    x = tf.keras.layers.Dropout(.5)(x)
    x = tf.keras.layers.Dense(8)(x)
    x = tf.keras.layers.Dropout(.5)(x)
    x = tf.keras.layers.Dense(3,activation='softmax')(x)
    return x
```

函式

tf.keras.layers.Dense

Just your regular densely-connected NN layer.

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

定義Model架構

```
X_input = tf.keras.Input(shape=4)
output = mlp(X_input)
model = tf.keras.Model(X_input,output)
```

函式

tf.keras.Input

Input() is used to instantiate a Keras tensor.

https://www.tensorflow.org/api_docs/python/tf/keras/Input

函式

tf.keras.Model

Model groups layers into an object with training and inference features.

https://www.tensorflow.org/api_docs/python/tf/keras/Model

可視化Model架構

```
print(model.summary())
```

使用keras構建深度學習模型時會通過model.summary()輸出模型各層的參數狀況

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 4)]               0
_________________________________________________________________
dense (Dense)                (None, 8)                 40
_________________________________________________________________
dropout (Dropout)            (None, 8)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 72
_________________________________________________________________
dropout_1 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 27
=================================================================
Total params: 139
Trainable params: 139
Non-trainable params: 0
_________________________________________________________________
None
```

LOSS與優化器

```
adam = tf.keras.optimizers.Adam(0.01)
model.compile(optimizer=adam,loss='categorical_crossentropy',metrics=['accuracy'])
```

可以選擇不同優化器!!!

https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/keras/optimizers?hl=zh-tw

LOSS與優化器

Classes

`class Adadelta` : Optimizer that implements the Adadelta algorithm.

`class Adagrad` : Optimizer that implements the Adagrad algorithm.

`class Adam` : Optimizer that implements the Adam algorithm.

`class Adamax` : Optimizer that implements the Adamax algorithm.

`class Ftrl` : Optimizer that implements the FTRL algorithm.

`class Nadam` : Optimizer that implements the NAdam algorithm.

`class Optimizer` : Updated base class for optimizers.

`class RMSprop` : Optimizer that implements the RMSprop algorithm.

`class SGD` : Stochastic gradient descent and momentum optimizer.

優化器

tf.keras.optimizers.Adam()

Adam optimization is a stochastic gradient descent method that is based on adaptive
tf.keras.optimizers.Adam
estimation of first-order and second-order moments.

callback

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.8,patience=50, min_lr=0.00001)
```

callback

tf.keras.callbacks.ReduceLROnPlateau()

Reduce learning rate when a metric has stopped improving.

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau

訓練

```
history=model.fit(x=X_train,y=y_train,epochs=1000,validation_data=(X_test,y_test),callbacks=[reduce_lr])
```

函式

history=model.fit()

fit函數返回一個history的對象，其history屬性記錄了損失函數和其他指標的數值隨epoch變化的情況，如果有驗證集的話，也包含了驗證集的這些指標變化情況

成果

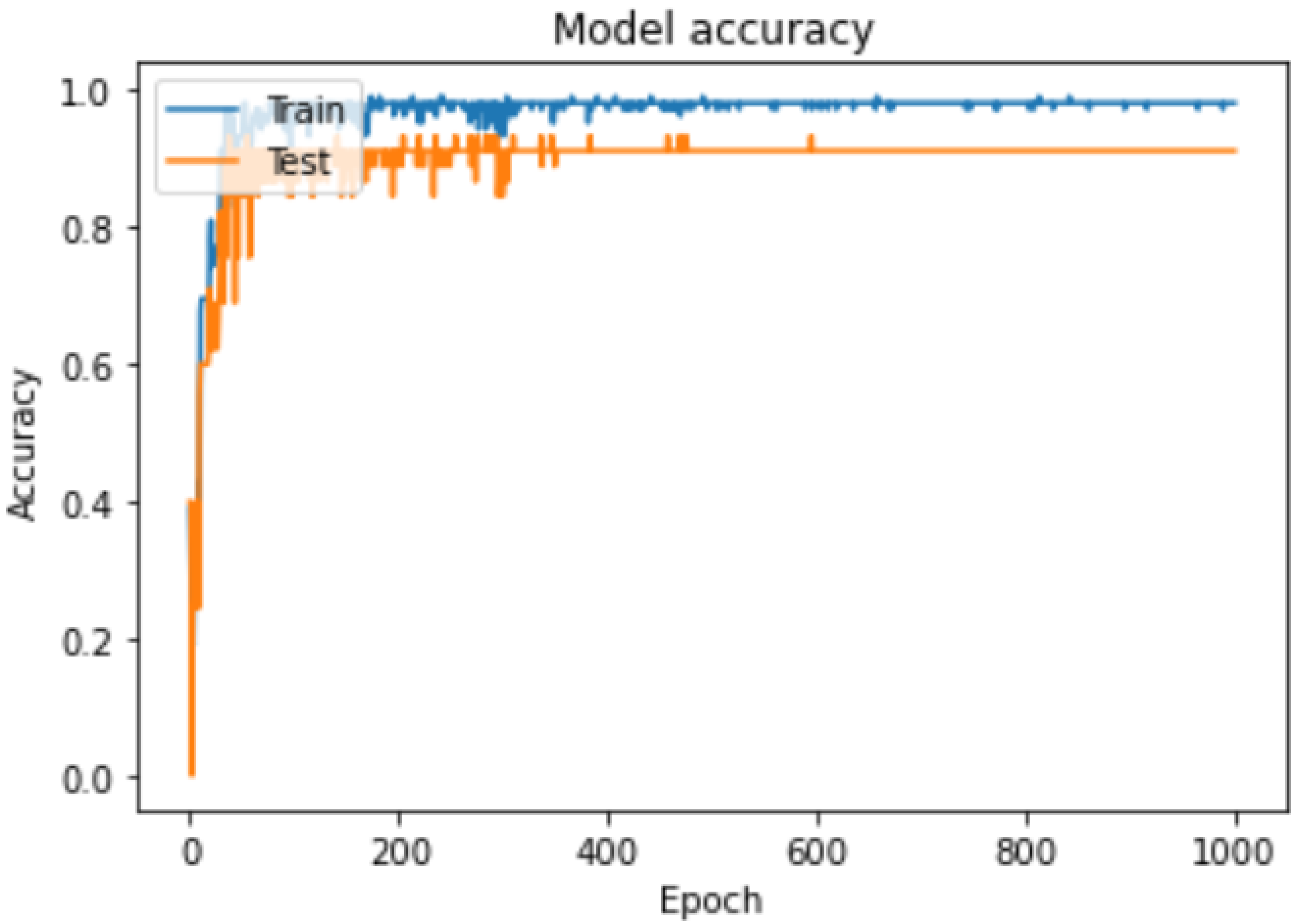
```
Epoch 985/1000
105/105 [==============================] - 0s 171us/sample - loss: 0.0454 - accuracy: 0.9810 - val_loss: 0.2414 - val_accuracy: 0.9111
Epoch 986/1000
105/105 [==============================] - 0s 178us/sample - loss: 0.0459 - accuracy: 0.9810 - val_loss: 0.2421 - val_accuracy: 0.9111
Epoch 987/1000
105/105 [==============================] - 0s 171us/sample - loss: 0.0453 - accuracy: 0.9810 - val_loss: 0.2427 - val_accuracy: 0.9111
Epoch 988/1000
105/105 [==============================] - 0s 180us/sample - loss: 0.0452 - accuracy: 0.9810 - val_loss: 0.2433 - val_accuracy: 0.9111
Epoch 989/1000
105/105 [==============================] - 0s 171us/sample - loss: 0.0457 - accuracy: 0.9714 - val_loss: 0.2447 - val_accuracy: 0.9111
Epoch 990/1000
105/105 [==============================] - 0s 180us/sample - loss: 0.0455 - accuracy: 0.9810 - val_loss: 0.2445 - val_accuracy: 0.9111
Epoch 991/1000
105/105 [==============================] - 0s 171us/sample - loss: 0.0455 - accuracy: 0.9810 - val_loss: 0.2442 - val_accuracy: 0.9111
Epoch 992/1000
105/105 [==============================] - 0s 171us/sample - loss: 0.0454 - accuracy: 0.9810 - val_loss: 0.2439 - val_accuracy: 0.9111
Epoch 993/1000
105/105 [==============================] - 0s 190us/sample - loss: 0.0454 - accuracy: 0.9810 - val_loss: 0.2436 - val_accuracy: 0.9111
Epoch 994/1000
105/105 [==============================] - 0s 180us/sample - loss: 0.0453 - accuracy: 0.9810 - val_loss: 0.2433 - val_accuracy: 0.9111
Epoch 995/1000
105/105 [==============================] - 0s 180us/sample - loss: 0.0452 - accuracy: 0.9810 - val_loss: 0.2429 - val_accuracy: 0.9111
Epoch 996/1000
105/105 [==============================] - 0s 176us/sample - loss: 0.0458 - accuracy: 0.9810 - val_loss: 0.2417 - val_accuracy: 0.9111
Epoch 997/1000
105/105 [==============================] - 0s 180us/sample - loss: 0.0452 - accuracy: 0.9810 - val_loss: 0.2415 - val_accuracy: 0.9111
Epoch 998/1000
105/105 [==============================] - 0s 171us/sample - loss: 0.0455 - accuracy: 0.9810 - val_loss: 0.2414 - val_accuracy: 0.9111
Epoch 999/1000
105/105 [==============================] - 0s 156us/sample - loss: 0.0453 - accuracy: 0.9810 - val_loss: 0.2410 - val_accuracy: 0.9111
Epoch 1000/1000
105/105 [==============================] - 0s 180us/sample - loss: 0.0463 - accuracy: 0.9810 - val_loss: 0.2408 - val_accuracy: 0.9111
```

成果圖

```
import matplotlib.pyplot as plt


plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

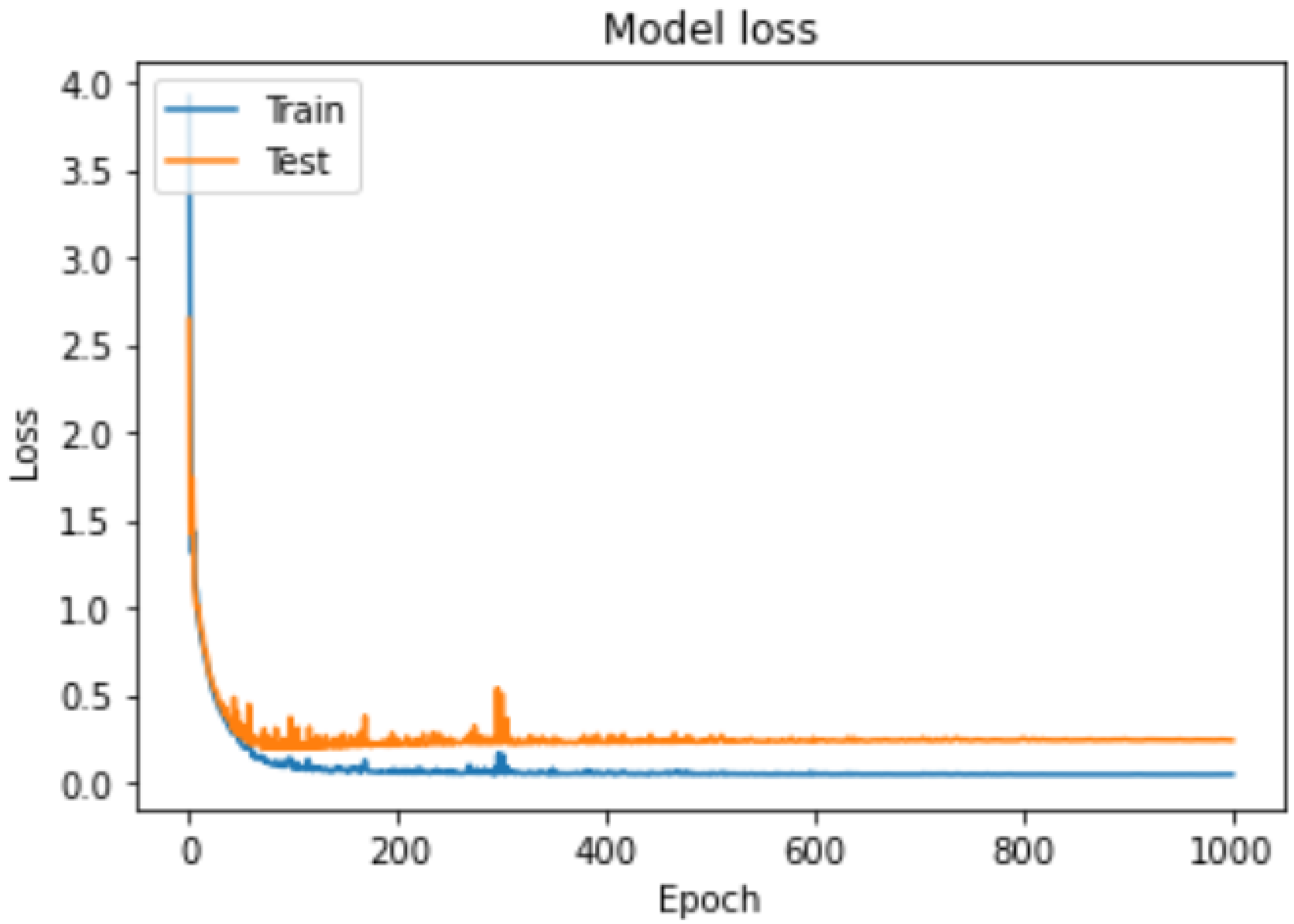
成果圖


Model accuracy
Train
Test
Accuracy
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000

成果圖

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

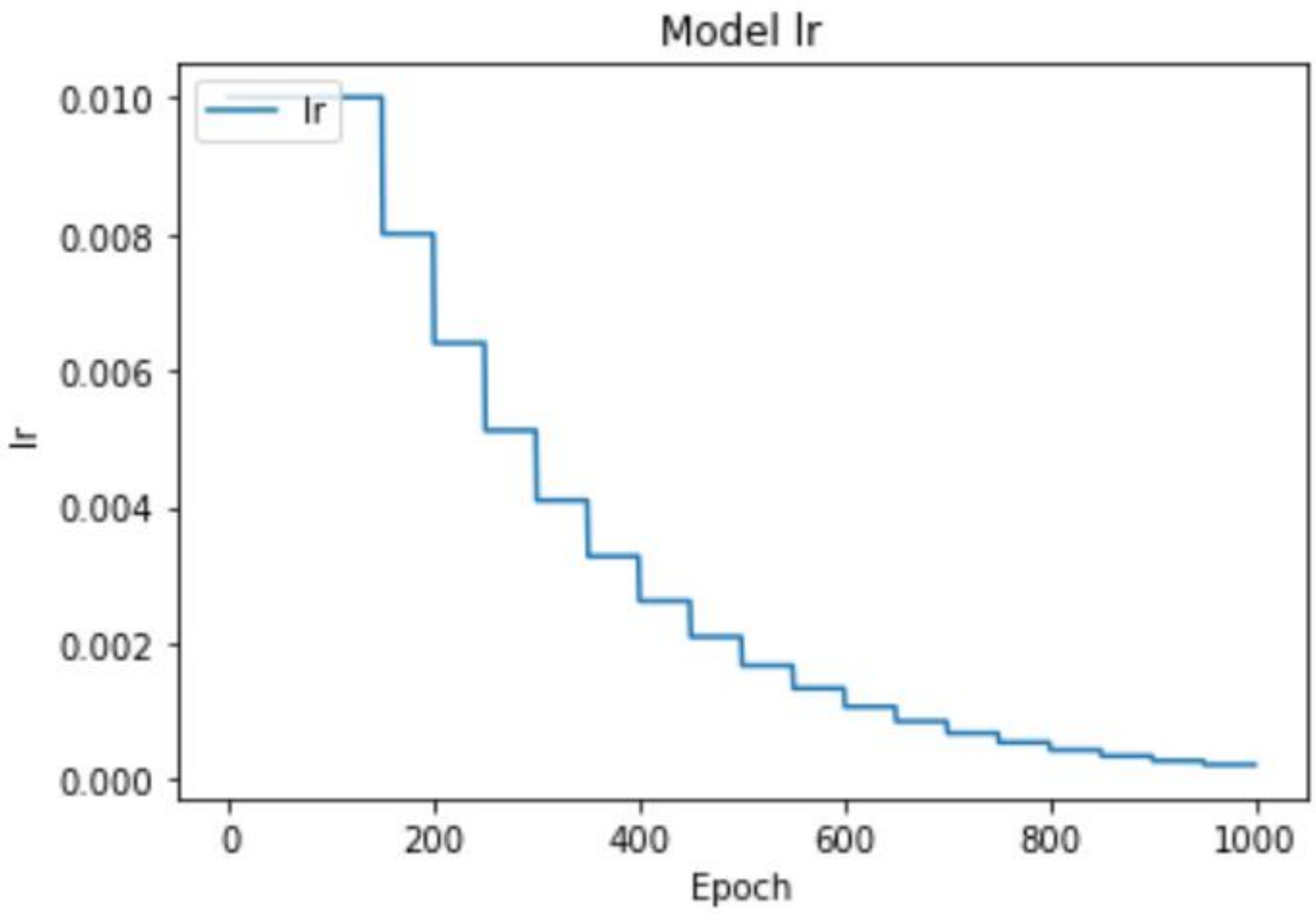
成果圖


Model loss
Train
Test
Loss
Epoch
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000

成果圖

```
plt.plot(history.history['lr'])
plt.title('Model lr')
plt.ylabel('lr')
plt.xlabel('Epoch')
plt.legend(['lr'], loc='upper left')
plt.show()
```

成果圖


Model lr
lr
0.010
0.008
0.006
0.004
0.002
0.000
0
200
400
600
800
1000
Epoch

預測

```
import numpy as np
XX=[[7,3.2,4.7,1.4]]
XX=np.array(XX)
print(XX)
```

```
[[7.   3.2 4.7 1.4]]
```

預測

```
import numpy as np
XX=[[5.4, 3.9, 1.7, 0.4]]
XX=np.array(XX)
print(XX)
print("--------------------------------")
print("XX資料預測為各類別機率如下:")
print(model.predict(XX))
print("--------------------------------")
print("XX資料預測類別如下:")
print(np.argmax(model.predict(XX)))
print("--------------------------------")
```

```
[[5.4 3.9 1.7 0.4]]
--------------------------------
XX資料預測為各類別機率如下:
[[1.0000000e+00 2.0533608e-09 0.0000000e+00]]
--------------------------------
XX資料預測類別如下:
0
--------------------------------
```

預測

```
pred = np.argmax(model.predict(XX))
#label對應
if pred==0:
    print("XX資料預測類別 : ","Iris Setosa")
elif pred==1:
    print("XX資料預測類別 : ","Iris Versicolour")
elif pred==2:
    print("XX資料預測類別 : ","Iris Virginica")
else:
    print("ERROR!!")
```

```
XX資料預測類別 :  Iris Setosa
```

函數

np.argmax()

在使用argmax()函數時，比如在深度學習裡面計算acc經常要用到這個參數，這個參數返回的是沿軸axis最大值的索引值

END