類神經網路(一)

Speaker: Malo

起手式

- 定義訓練資料
- 定義神經網絡模型
- 訓練模型
- 測試模型

Demo1

- MNIST資料集
- 算是類神經網路的hello world
- 親手建立模組，使用colab實作

Demo1

先引入需要的module

```
from  tensorflow.keras  import  models
from  tensorflow.keras  import  layers
from  tensorflow.keras.datasets  import  mnist
from  tensorflow.keras.utils  import  to_categorical
```

Demo1

載入資料

```
#- 備資料
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

#- 整理資料
train_images = train_images.reshape((60000, 28 * 28))    #reshape 是 NumPy 陣列的 method
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Demo1

建立模型

```
#-  建立layer,  model
network  =  models.Sequential()
network.add(layers.Dense(512,  activation='relu',  input_shape=(28  *  28,)))
network.add(layers.Dense(10,  activation='softmax'))

network.compile(optimizer='rmsprop',
                                loss='categorical_crossentropy',
                                metrics=['accuracy'])
```

Demo1

訓練模型

```
#-  準備標籤
train_labels  =  to_categorical(train_labels)
test_labels  =  to_categorical(test_labels)

#-  training
network.fit(train_images,  train_labels,  epochs=5,  batch_size=128)
```

Demo1

測試模型(使用模型預測)

```
#-  testing
test_loss,  test_acc  =  network.evaluate(test_images,  test_labels)
print('test_acc:',  test_acc)
```

```
Epoch 1/5
469/469 [==============================] - 5s 10ms/step - loss: 0.2582 - accuracy: 0.9256
Epoch 2/5
469/469 [==============================] - 5s 10ms/step - loss: 0.1033 - accuracy: 0.9692
Epoch 3/5
469/469 [==============================] - 5s 10ms/step - loss: 0.0692 - accuracy: 0.9796
Epoch 4/5
469/469 [==============================] - 5s 10ms/step - loss: 0.0494 - accuracy: 0.9849
Epoch 5/5
469/469 [==============================] - 5s 10ms/step - loss: 0.0367 - accuracy: 0.9886
313/313 [==============================] - 1s 3ms/step - loss: 0.0742 - accuracy: 0.9801
test_acc: 0.9800999760627747
```

回頭再把code看細一點

- Load_data()就已經分好train, test資料了

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

- 之前有說過，輸入model的基本上都是一維的資料型態
- 這邊是為了觀察一筆資料是28x28的型態，並不符合需求
- 另外，每一個值是0~255的灰階數值

```
print(train_images.ndim)#ndim 為 3, 有 3 個軸
```

```
3
```

```
print(train_images.shape)#shape 為 60000x28x28 維的 3D 張量 (有 3 個元素)
```

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

```
print(train_images.dtype)#元素的資料型別為 0~255 的整數
```

```
uint8
```

- 這邊算是對資料的前處理
- Reshape用來把28x28的2, 3維轉換為1維的資料，也就是一個vector的形式
- /255的原因是要讓資料正規化為0~1的數值

```
#-  整理資料
train_images  =  train_images.reshape((60000,  28  *  28))    #reshape  是  NumPy  陣列的  method
train_images  =  train_images.astype('float32')  /  255

test_images  =  test_images.reshape((10000,  28  *  28))
test_images  =  test_images.astype('float32')  /  255
```

- 轉換前

```
print(train_images[0].shape)
```

```
(28, 28)
```

```
print(train_images[0][10])
```

```
[  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

- 轉換後

```
print(train_images[0].shape)
```

```
(784,)
```

```
print(train_images[0])
```

```
 0.74509805 0.99215686 0.27450982 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.13725491 0.94509804
 0.88235295 0.627451   0.42352942 0.00392157 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.21764707 0.9411765  0.99215686
```

- 指定我們要建立的model長什麼樣子
- Sequential()指定要以線性堆疊建立模型，是最常見的架構
- Layers.Dense的參數1指定輸出的vector是512的元素
- 參數2是指定計算的方式為relu，前面我們有介紹過多種計算方式
- 參數3是指定輸入的vector的維度

```
#-  建立layer,  model
network  =  models.Sequential()
network.add(layers.Dense(512,  activation='relu',  input_shape=(28  *  28,)))
network.add(layers.Dense(10,  activation='softmax'))
```

- 第二個layers.Dense的第一個參數指定輸出為10，對應我們要分類的項次
- 第二個參數則是指定以softmax，會讓輸出的值都介於(0, 1)

```
#-  建立layer,  model
network  =  models.Sequential()
network.add(layers.Dense(512,  activation='relu',  input_shape=(28  *  28,)))
network.add(layers.Dense(10,  activation='softmax'))
```

- 編譯我們的Model，用以指定訓練model期間的準則參數
- Optimizer指定我們使用 root sqr的方式評估
- Loss 指定損失函式的種類
- Metrics指定accuracy為評量準則

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

- 轉換為我們分類用的標簽

```
#-  準備標籤
train_labels  =  to_categorical(train_labels)
test_labels  =  to_categorical(test_labels)
```

- 開始訓練模型

```
#-  training
network.fit(train_images,  train_labels,  epochs=5,  batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 5s 10ms/step - loss: 0.2595 - accuracy: 0.9254
Epoch 2/5
469/469 [==============================] - 5s 10ms/step - loss: 0.1057 - accuracy: 0.9684
Epoch 3/5
469/469 [==============================] - 5s 10ms/step - loss: 0.0687 - accuracy: 0.9795
Epoch 4/5
469/469 [==============================] - 5s 10ms/step - loss: 0.0496 - accuracy: 0.9849
Epoch 5/5
469/469 [==============================] - 5s 10ms/step - loss: 0.0375 - accuracy: 0.9886
<keras.callbacks.History at 0x7f9e03fcde90>
```

- 測試模型
- 我們可以看到accuracy有較小，這是可能有點overfiting

```
#-  testing
test_loss,  test_acc  =  network.evaluate(test_images,  test_labels)
print('test_acc:',  test_acc)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0686 - accuracy: 0.9805
test_acc: 0.9804999828338623
```