C

TensorFlow 2.x 版與Keras

- C-1 安裝 TensorFlow 2.x 版
- C-2 使用客製化 WinPython 套件
- C-3 使用 TensorFlow 2.x 版的 tf.keras
- C-4 在 Colaboratory 使用 TensorFlow 1.x 版

C-1 安裝 TensorFlow 2.x 版

TensorFlow 2.0 版已經在 2019 年 9 月 30 日正式釋出，因為 Keras 在 2.x 版已經成為 TensorFlow 預設的高階函式庫，即 TensorFlow 子模組 tf.keras，所以不再需要獨立安裝 Keras。

雖然 2019 年 9 月 17 日 Keras 已經釋出最新 2.3.0 版，不過，這是最後一個支援多後台（Backend）的版本，而且 Keras 套件與 tf.keras 模組是同步釋出，Keras 開發者 Francois Chollet 建議在未來開發深度學習專案時，使用 TensorFlow 2.0 和 tf.keras。

安裝 TensorFlow 2.x

在 Anaconda 安裝 TensorFlow 目前預設安裝的就是 2.x 版（因為已經有 tf.keras，不再需要安裝 Keras 套件）。請執行「開始/Anaconda3 (64-bits)/Anaconda Prompt」命令開啟「Anaconda Prompt」命令提示字元視窗後，使用 pip install 指令安裝 TensorFlow 套件，如下所示：

```
(base) C:\Users\JOE>pip install tensorflow Enter
```

安裝 TensorFlow GPU 2.x

TensorFlow 2.1 之後版本已經整合 CPU 和 GPU，並不需要額外安裝 GPU 版的 TensorFlow，其硬體需求如下所示：

- CUDA® Compute Capability 3.5 以上版本 NVIDIA® GPU 顯示卡。

在使用 TensorFlow GPU 前，我們需要安裝 NVIDIA 軟體套件（請注意！如果沒有 cudart64_101.dll 檔案，TensorFlow GPU 並不會載入），如下所示：

- NVIDIA® GPU 驅動程式：CUDA 10.1 需要 418.x 以上版本。

- CUDA® Toolkit：TensorFlow 支援 CUDA 10.1 (TensorFlow 2.1.0 以上版本)。
- CUDA Toolkit 隨附 CUPTI。
- cuDNN SDK (7.6 以上版本)。

Anaconda 建立 tf2_gpu 虛擬環境後，在「Anaconda Prompt」命令提示字元視窗輸入下列指令安裝 NVIDIA 軟體套件（在安裝前請先更新顯示卡的驅動程式），如下所示：

```
(tf2_gpu) C:\Users\JOE>conda install cudatoolkit=10.1 Enter
(tf2_gpu) C:\Users\JOE> conda install cudnn=7.6 Enter
```

如果是自行安裝 NVIDIA 軟體套件，需要將 CUDA、CUPTI 和 cuDNN 安裝路徑新增至 %PATH% 環境變數。例如：CUDA Toolkit 安裝至「 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1」路徑，cuDNN 是安裝至「C:\tools\cuda」，其路徑如下所示：

```
SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\bin;%PATH%
SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\extras\CUPTI\libx64;%PATH%
SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\include;%PATH%
SET PATH=C:\tools\cuda\bin;%PATH%
```

如果 TensorFlow 使用 2.0.2 之前的版本，Anaconda 建立 tf2_gpu 虛擬環境後，請在「Anaconda Prompt」命令提示字元視窗輸入下列指令安裝 TensorFlow GPU，如下所示：

```
(tf2_gpu) C:\Users\JOE>conda install tensorflow-gpu Enter
```

上述指令的提示符號字串前是(tf2_gpu)，表示是安裝在 tf2_gpu 虛擬環境，因為需下載安裝 CUDA 和 cuDNN，需花些時間，請稍等一下，等到安裝完成，可以在最後看到 done 訊息文字。

C-2 使用客製化 WinPython 套件

WinPython 是支援 Windows 作業系統的一套免費且開放原始碼的科學和教育用途可攜式版本的 Python 整合散發套件，其官方網址如下所示：

- http://winpython.github.io/

上述網頁提供多種不同版本和預設安裝套件的下載網址，點選即可下載指定的套件檔案。

下載與安裝 fChart 客製化 WinPython 套件

在本書提供整合 fChart 教學工具和客製化 WinPython 套件的 Python 開發環境 – AI_tf 版（已經安裝 TensorFlow 2.3.0 版，不含 NVIDIA 軟體套件），並且提供工作列主選單來快速啟動相關工具，這是 7-Zip 格式的自解壓縮檔：fChartPython6_3.76AI_tf.exe。

https://drive.google.com/file/d/1IoI1wjCBSv2l7f3CqRG5dO1c5ROR4oPj/view?usp=sharing

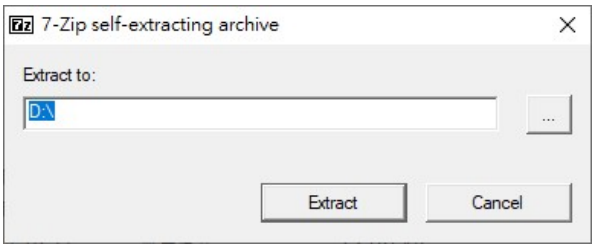
TensorFlow 2.1 版之後版本已經整合 CPU 和 GPU，如果 Windows 電腦的 CUDA 和 cuDNN 沒有安裝正確版本，TensorFlow 會產生錯誤，如下所示：

- ImportError: DLL load failed: 找不到指定的模組。
- ImportError: No module named '_pywrap_tensorflow_internal'

如果 TensorFlow 不使用 GPU，只使用 CPU，TensorFlow 需安裝 2.0.2 版，7-Zip 格式的自解壓縮檔：fChartPython6_3.76AI_tf_cpu.exe。

https://drive.google.com/file/d/1nHsAhy3Mi1U_ZxSfKHX-iXUXosJcE3mW/view?usp=sharing

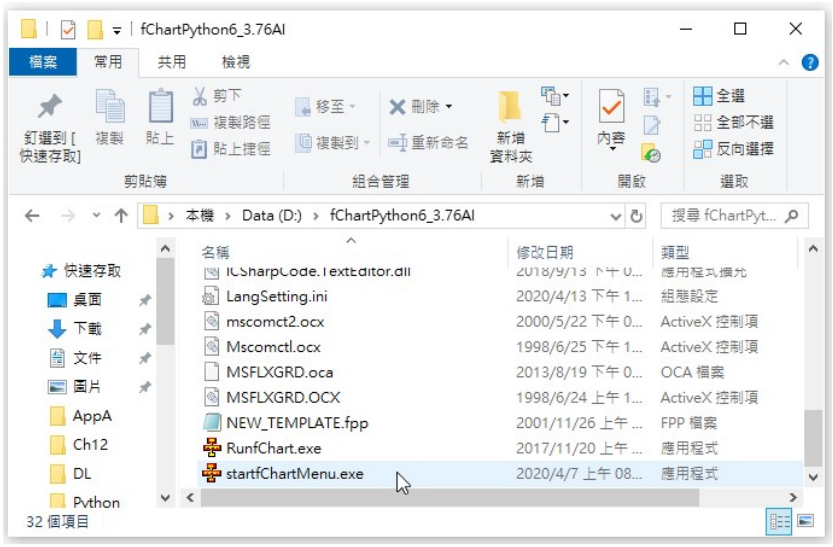
在成功下載 fChart 客製化 WinPython 套件後，請執行 7-Zip 自解壓縮檔，在欄位輸入解壓縮的硬碟，例如：「C:\」或「D:\」等，按【Extract】鈕，即可解壓縮安裝客製化 WinPython，如下圖所示：



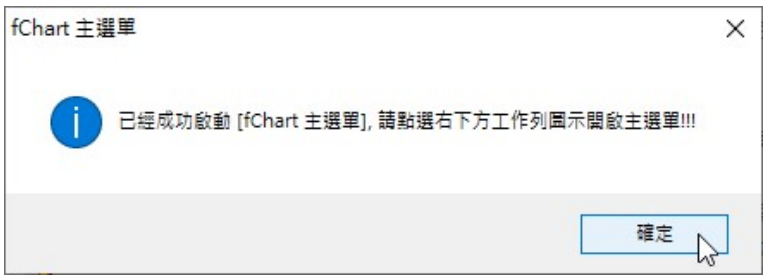
在成功解壓縮後，預設建立名為「\fChartPython6_3.76AI_tf」目錄。

使用 fChart 客製化 WinPython 套件

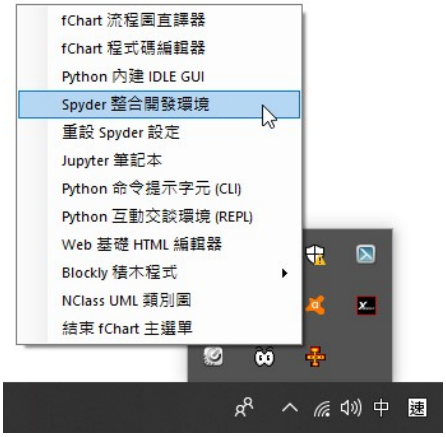
請開啟解壓縮的目錄捲動至最後，按二下【startfChartMenu.exe】執行 fChart 主選單。



可以看到訊息視窗顯示已經成功在工作列啟動主選單，請按【確定】鈕。



然後，在右下方工作列可以看到 fChart 圖示，點選圖示可以看到 fChart 主選單，請點選命令來啟動 fChart 和 Python 相關工具，如下圖所示：



Jupyter Notebook 筆記本儲存的根目錄是位在「\fChartPython6_3.76AI_tf\WinPython\notebooks」。

C-3 使用 TensorFlow 2.x 版的 tf.keras

TensorFlow 2.x 的 tf.keras 子模組基本上和 Keras 套件是相同的，我們可以輕鬆將舊版 Keras 套件的 Python 程式修改成 TensorFlow 2.x 的 tf.keras 版本。

C-3-1 將 Python 程式改成使用 tf.keras

將本書 Python 程式升級成為 TensorFlow 2.x 的 tf.keras 版本，其步驟如下所示：

步驟一：修改匯入模組的程式碼

將 Python 程式開頭原來匯入 Keras 模組的程式碼，如下所示：

```
from keras.??? Import ???
```

上述程式碼匯入的是舊版 Keras 套件的模組，我們只需加上 tensorflow 就可以改為匯入 tf.keras 的相同模組，如下所示：

```
from tensorflow.keras.??? Import ???
```

目前來說，大部分模組可以直接修改匯入的程式碼，有差異的模組，如下所示：

- Ch16_1a.ipynb 的 model_to_dot，原匯入的程式碼如下所示：

```
from keras.utils.vis_utils import model_to_dot
```

上述程式碼需改成下列的程式碼，如下所示：

```
from tensorflow.keras.utils import model_to_dot
```

- 第 16-4 節 Python 程式的 concatenate，原匯入的程式碼如下所示：

```
from keras.layers.merge import concatenate
```

上述程式碼需改成下列的程式碼，如下所示：

```
from tensorflow.keras.layers import concatenate
```

步驟二：在 model.evaluate()函數新增 verbose 參數

舊版 Keras 的 model.evaluate()函數，其 verbose 參數預設值是 1，如下所示：

```
loss, accuracy = model.evaluate(X, Y)
```

上述程式碼的其執行結果會顯示進度列符號「=」，如下所示：

```
768/768 [==============================] - 0s 341us/step
```

在 TensorFlow 2.x 的 tf.keras，其執行結果如下所示：

```
768/1
[==============================================================
==============================================================
....
==============================================================
==============================] - 0s 226us/sample - loss: 0.4928 -
accuracy: 0.7669
準確度 = 0.77
```

上述執行結果會有很長「=」符號的進度列（2.1 之後版本並不會顯示此進度列），為了讓 2.0.2 之前版本不顯示這些進度列符號，本書 Python 程式修改 model.evaluate()函數新增 verbose 參數值 0，如下所示：

```
loss, accuracy = model.evaluate(X, Y, verbose=0)
```

上述程式碼的執行結果不會顯示進度列的執行結果。

步驟三：將 history 屬性的 acc 改成 accuracy

在 Keras 的 history.history 屬性原來的準確度和驗證準確度是 acc 和 val_acc，tf.keras 已經改成 accuracy 和 val_accuracy，原來顯示訓練和驗證準確度圖表的程式碼，如下所示：

```
acc = history.history["acc"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_acc"]
plt.plot(epochs, acc, "b-", label="Training Acc")
plt.plot(epochs, val_acc, "r--", label="Validation Acc")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

我們需要將"acc"和"val_acc"改成"accuracy"和"val_accuracy"，如下所示：

```
acc = history.history["accuracy"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "b-", label="Training Acc")
plt.plot(epochs, val_acc, "r--", label="Validation Acc")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

同理，在 ModelCheckpoint 物件的 monitor 參數值也需改為 val_accuracy，如下所示：

```
mc = ModelCheckpoint("best_model.h5", monitor="val_accuracy",
                     mode="max", verbose=1,
                     save_best_only=True)
```

C-3-2 TensorFlow 2.x 的 Eager Execution 問題

TensorFlow 2.x 是使用 Eager Execution 來加速訓練，如果是使用 2.0.2 之前的版本，大部分舊版 Keras 都沒有問題，只有少數需要停用 Eager Execution 才能執行，如下所示：

- Ch13_2a.py：此 Python 程式需要停用 Eager Execution 才能正確執行，如下所示：

```
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
```

- Ch13_5_3.py：此 Python 程式執行十份緩慢，幾乎停在哪沒有執行，而 2.1 之後版本的執行結果並不正確，我們需要停用 Eager Execution 才能正確執行，如下所示：

```
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
```

上述程式碼停用 2.x 版的 Eager Execution，改用 1.x 版來執行，請注意！需重新啟動 Spyder 才能恢復 TensorFlow 2.x 版。

C-4 在 Colaboratory 使用 TensorFlow 1.x 版

目前 Colaboratory 雲端服務預設使用 TensorFlow 2.x 版，我們可以使用下列指令來指定使用 TensorFlow 1.x 版，如下所示：

```
%tensorflow_version 1.x
import tensorflow
print(tensorflow.__version__)
```

如果已經啟動 Runtime，需要重新啟動 Runtime。