機器學習於材料資訊的應用
Machine Learning on Material Informatics

陳南佑(NAN-YOW CHEN)

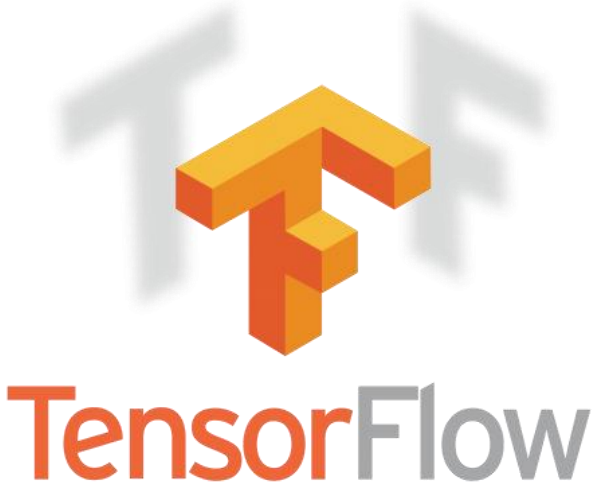
nanyow@narlabs.org.tw

楊安正(AN-CHENG YANG)

acyang@narlabs.org.tw

Framework For Machine Learning

提供
Machine Learning演算法
資料處理相關函式
特徵工程相關函式



TensorFlow 是一個廣為流行的機器學習軟體庫(框架)，由 Google 開源，透過Computational Graph，來進行數值演算。

由 Facebook 開源，建立在 Torch 之上的深度學習框架，GPUs First，Python First ，底層由LuaJIT和C/CUDA實作。

學習深度學習框架

- 學生和研究者：
  - 數學概念和運算的程式化表達。(回想一下自己刻linear regression和呼叫scikit-learn的差異)
  - 「符號計算」的能力，特別是針對偏微分運算。
- 開發者和工程師：
  - 在你的產品中加入一些與人工智慧相關的功能
  - 將已有的深度學習模型部署到各種場景中
  - 如何匯出訓練好的模型？
  - 如何在本機使用已有的預訓練模型？
  - 如何在伺服器、可攜式裝置、嵌入式設備甚至網頁上高效運行模型？

TensorFlow的特點

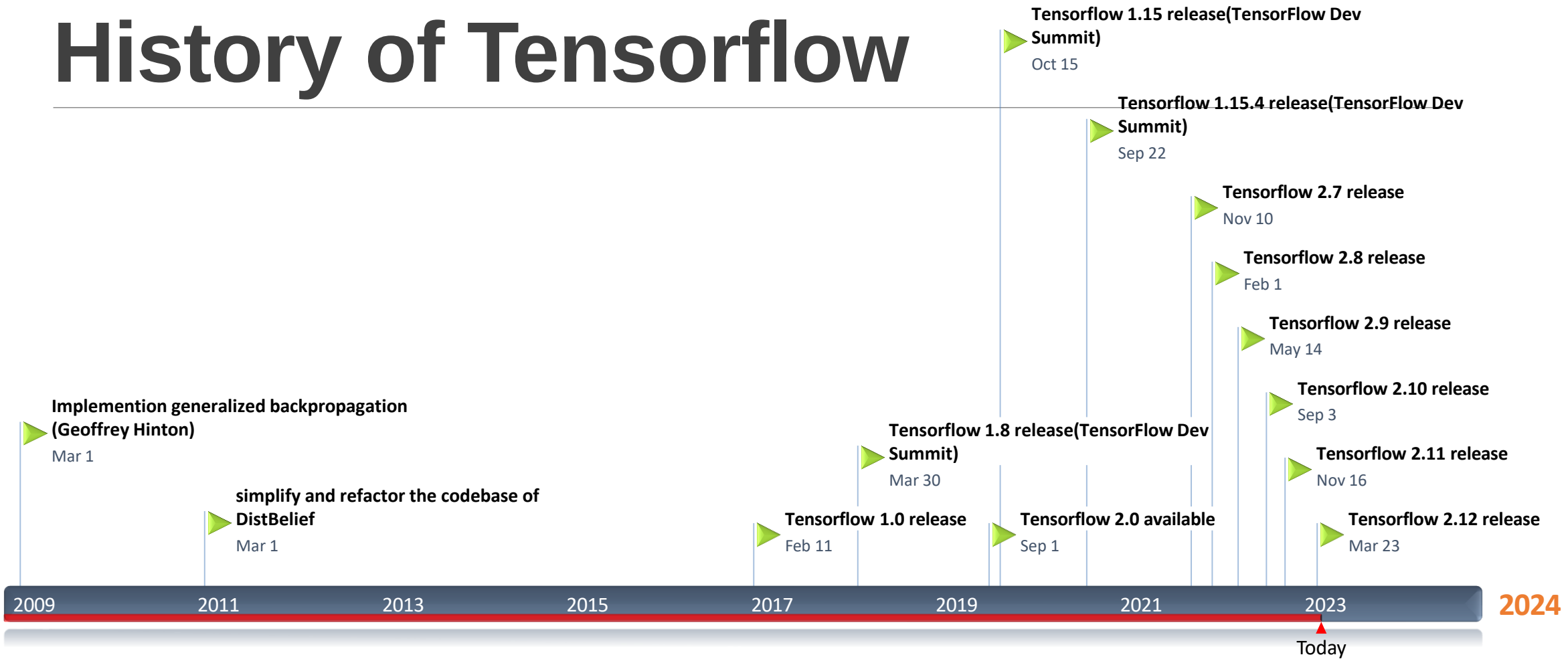
- 訓練流程
  - 資料的處理 ：使用 tf.data 和 TFRecord 可以高效的建構預處理資料集與訓練資料集。同時可以使用 TensorFlow Datasets 快速載入常用的公開資料集。
  - 模型的建立與測試 ：使用即時執行模式和著名的神經網路高層 API 框架 Keras，結合可視化工具 TensorBoard，簡易、快速地建立和測試模型。也可以通過 TensorFlow Hub 方便的載入已有的成熟模型。
  - 模型的訓練 ：支援在 CPU、GPU、TPU 上訓練模型，支援單機和多台電腦平行訓練模型，充分利用大量資料和計算資源進行高效訓練。
  - 模型的匯出 ：將模型打包匯出為統一的 SavedModel 格式，方便遷移和部署。(proto-buff format or graph define)

TensorFlow

- 部署流程
  - 伺服器部署 ：使用 TensorFlow Serving 在伺服器上為訓練完成的模型提供高性能、且可平行運算的高流量 API。(Google Computing Platform)
  - 可攜式裝置和嵌入式設備部署 ：使用 TensorFlow Lite 將模型轉換為體積小、高效率的輕量化版本，並在可攜式裝置、嵌入式端等功耗和計算能力受限的設備上運行，支援使用 GPU 代理進行硬體加速，還可以配合 Edge TPU 等外接硬體加速運算。
  - 網頁端部署 ：使用 TensorFlow.js，在網頁端等支援 JavaScript 運行的環境上也可以運行模型，支援使用 WebGL 進行硬體加速。

Google 向 FDA 提交審批 Fitbit 的簡化版的心率不正偵測技術
https://www.nature.com/articles/s41598-019-49092-2

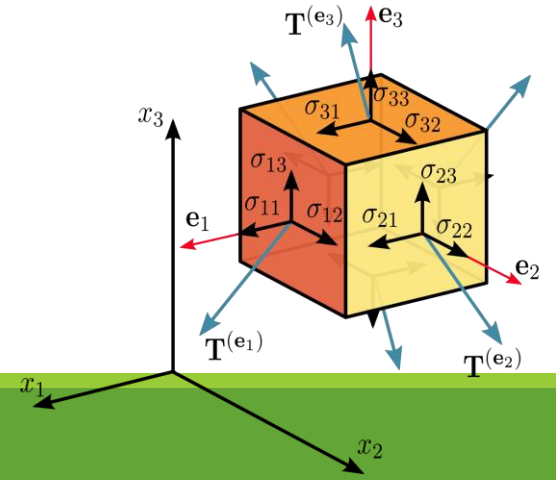
History of Tensorflow



What is tensor?

- 純量Scalar，只有大小、沒有方向、可用實數表示的一個量。(通常是為了和向量、張量作區別)
- 向量Vector，帶有方向與大小的量。例如卡式坐標系下，我們使用$\hat{\imath}, \hat{\jmath}, \hat{k}$ 來描述方向，重力加速度可以表示為(0, 0, -9.8)。
- 張量Tensor，要表達的物理量比較複雜，沒辦法單純靠大小和方向就可以完整描述，例如應力，單位面積的受力，需要一個面的法向量和力量的方向
- 即使增加n維向量的維度，只是改變參考系的單位向量數量(基底)，描述的性質還是只有大小與方向這兩種屬性)

- $\boldsymbol{T^{(n)} = n \cdot \sigma}$
- $\boldsymbol{\sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}}$



What is tensor?

- 2D張量圖例
- **能帶結構資料集(2023-ML@MGI-Week07-imp/data.csv)**，其中包含7種特徵，依序是’Pretty Formula', 'Density', 'Energy', 'Energy_per_Atom', 'Volume‘, 'Formation_Energy_per_Atom', 'Band Gap'，該資料一共採集了194種化合物。

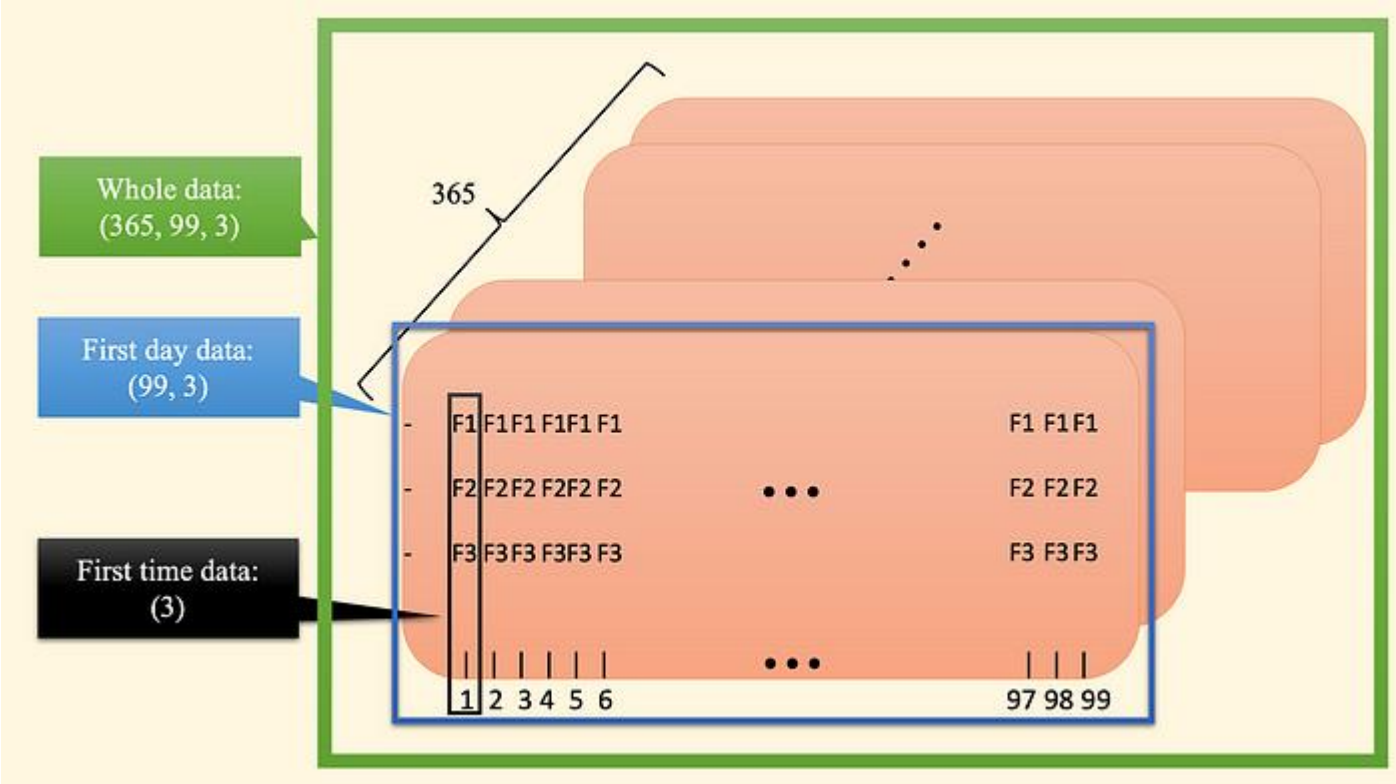
	Pretty Formula	Density	Energy	Energy_per_Atom	Volume	Formation_Energy_per_Atom	Band Gap
0	B13C2	2.438648	-106.358032	-7.090535	112.056191	-0.072416	0.0089
1	SiB3	2.449974	-204.992906	-6.406028	328.145035	-0.040804	1.4083
2	SnB6	4.110240	-42.852113	-6.121730	74.164800	0.175492	0.7527
3	B6Pb	6.003820	-42.968600	-6.138371	75.248129	0.114862	0.9209
4	BN	3.302143	-68.876677	-8.609585	49.920018	-1.294085	4.7885

```
In [7]: #dimension of the dataset
        print(np.shape(dataset))
        print(dataset.shape)

        (194, 7)
        (194, 7)
```

What is tensor?

- 3D張量圖例

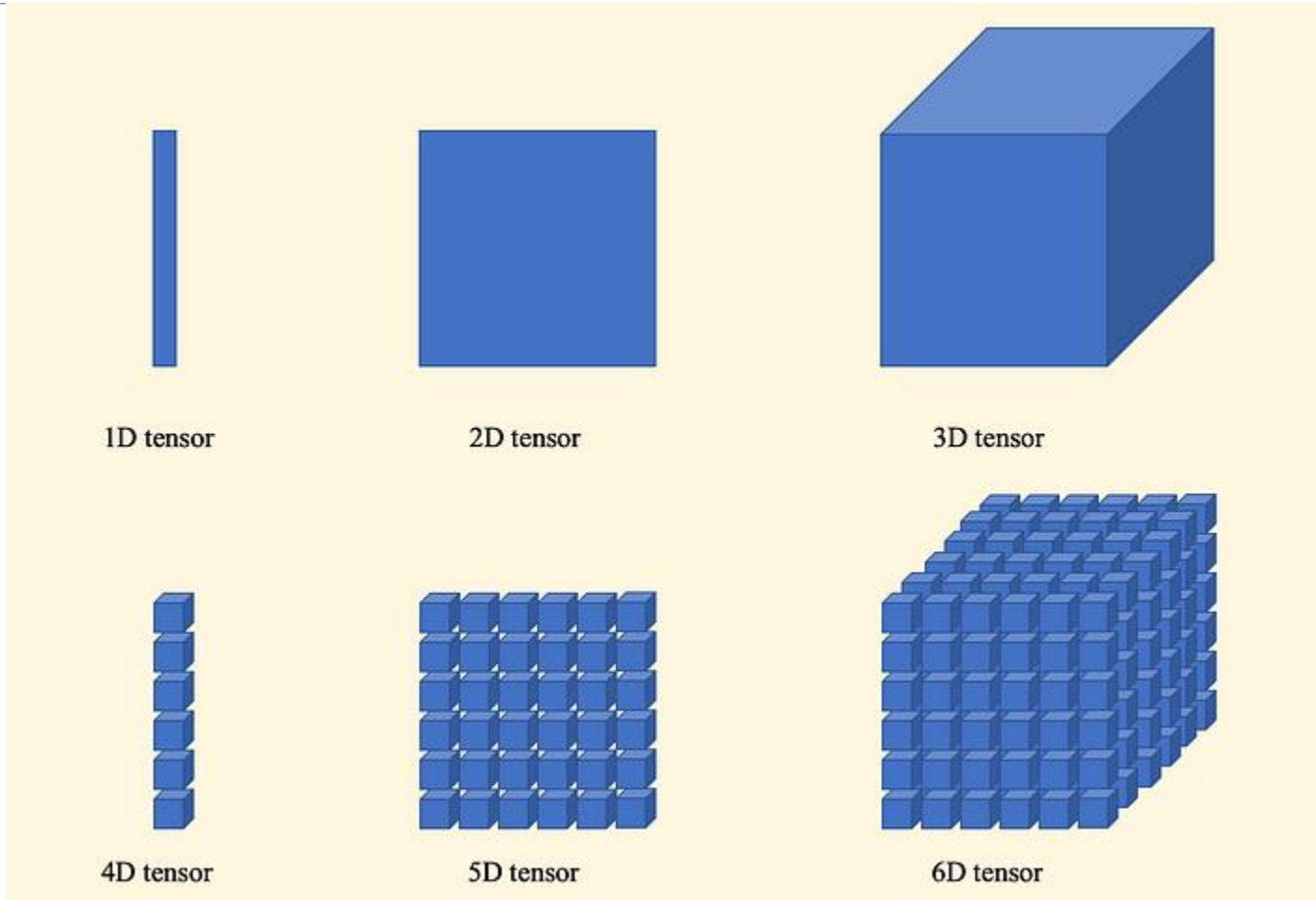
  記錄日常溫度變化的資料，每次測量記錄三項資料，分別為最高溫、最低溫、濕度，一天記錄99次，持續紀錄一年。



https://medium.com/hunter-cheng/機何為張量-tensor-三分鐘圖解類神經網路基本資料結構-ab0ccd115aff

What is tensor?

- 高維張量圖例
- Keras 之父 François Chollet大神提到區分“**向量**”和“**張量**”的重點：**維度(dimensionality)**是指特定軸上的元素數量或張量中的軸數量，所以容易造成混淆。
- 一個軸組成的向量，軸上有3個元素，**稱為3維向量**。
- 一個張量由3個軸組成，稱為**3維張量**。
- 若是怕搞混，可以將後者說成**3階張量**因為張量的階就是軸的數量。



https://medium.com/hunter-cheng/機何為張量-tensor-三分鐘圖解類神經網路基本資料結構-ab0ccd115aff

What is tensor?

- 彩色圖片就是很典型的例子:一個圖片由像素矩陣所構成，而每一個像素又需要靠RGB色彩空間的三個基本單位來描述一個色彩向量。舉例3*3的彩色圖片，包含水平位置，垂直位置，以及色彩向量(R,G,B)。總共三個屬性，我們可以定義其階數為3。
- 純量表示：[1]
- 向量表示：[1,2,3]
- 階數為2的張量表示：

[[1,2],

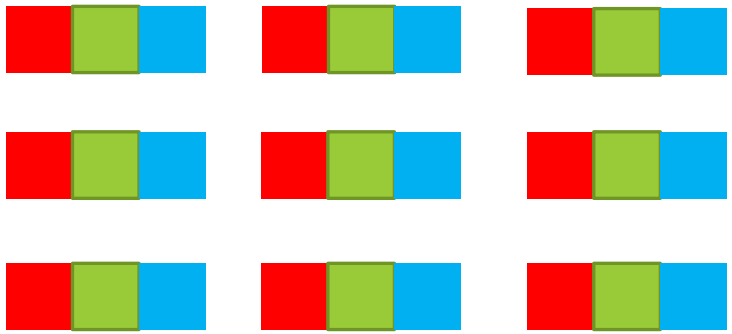
[3,4]]

- 階數為3的張量表示：

[[[R,G,B], [R,G,B], [R,G,B]],

[[R,G,B], [R,G,B], [R,G,B]] ,

[[R,G,B], [R,G,B], [R,G,B]]]



What is tensor?

- 從TensorFlow 的基本運算單位是張量（Tensor），零維張量等於是純量(Scalar)，一維張量是向量(Vector)，二維張量是矩陣(Matrix)等等。
- 和Numpy相對比，ndarray（n 維陣列）觀念與 Tensor（n 維張量）觀念類似以外，TensorFlow 函數的命名、參數與設計概念都很接近 NumPy。

What is tensor?

NUMPY

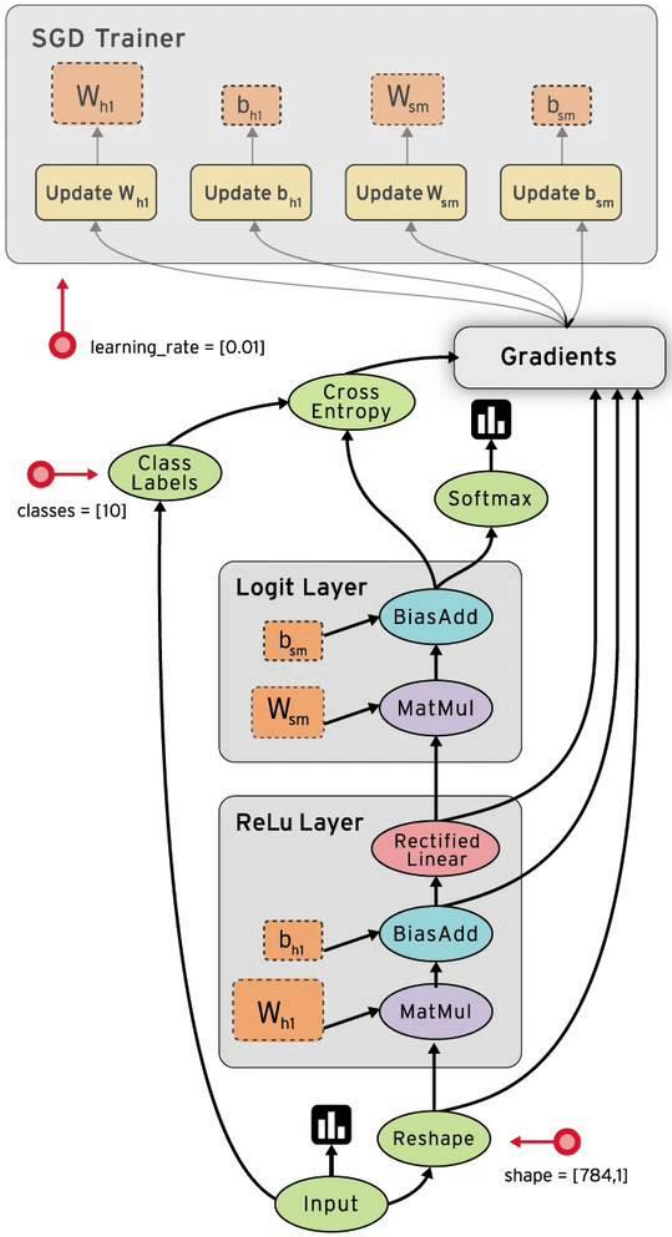
```
one_d_arr = np.arange(24)

two_d_arr = np.arange(24).reshape(6, 4)

three_d_arr = np.arange(24).reshape(2, 3, 4)
```

TENSORFLOW

```
one_d_tensor = tf.constant(np.arange(24))

two_d_tensor = tf.constant(np.arange(24), shape=(6, 4))

three_d_tensor = tf.constant(np.arange(24), shape=(2, 3, 4))
```

What is Flow?

- Tensorflow1.x 主要的運作流程分為以下兩個部分
  - 建立模型(Build Model)
  - 執行運算(Run)
- Tensorflow1.x設計的核心就是Tensor的流動，建立Graph的過程其實只是定義好Tensor如何流動並運算的過程，但真正的資料其實並沒有被運算，真正的計算需要用session來執行。
- 靜態的資料結構設計成Tensor張量，Flow表達運算的動態路徑流，TensorFlow利用Graph來表示Tensor的運算流程。



diff Tensorflow1 Tensorflow2

- 1. API clean up:
  - TensorFlow 2 整理了之前為人詬病的 API，以前同一種功能、API可能有很多種 call 法，這次 TensorFlow 將 API 整頓了一番，並刪除了很多不必要的 APIs。
- 2. Eager Execution(動態圖模式、即時執行模式)
  - 以前在 TensorFlow1 需要將 tensor 餵給 session.run() 來 compile graph，在 TensorFlow 2 可以動態建立 graph，” executes eagerly”：在執行時可以即時把值 return 回來不再需要 session.run()，這樣更容易 debug、寫起來更直覺也更像 Python。
- 3. SavedModel
  - SavedModel 包含了整個 TensorFlow Model，完全獨立於程式碼。在 TensorFlow 2，model 的傳送、分享、deployment (TFLite, TensorFlow.js …)變得更直覺 。例如NVIDIA Triton (TensorRT)。

Hello World in Tensorflow1.x

```
import tensorflow as tf

hw = tf.constant("Hello World")
with tf.Session() as sess:
    print(sess.run(hw))
```

sess.run是Tensorflow1.x靜態圖最大的特徵

Tensorflow2.x Eager mode，讓你用起來更直覺

```
import tensorflow as tf

hw = tf.constant("Hello World")
print(hw)
```

matrix multiplication in Tensorflow1.x

```
import tensorflow as tf

# Build a dataflow graph.
c = tf.constant([[1.0, 2.0], [3.0, 4.0]])
d = tf.constant([[1.0, 1.0], [0.0, 1.0]])
e = tf.matmul(c, d)

# Construct a `Session` to execute the graph.
with tf.Session() as sess:
  # Execute the graph and store the value that `e` represents in `result`.
  result = sess.run(e)

print(result)
```

matrix multiplication in Tensorflow2.x

```
import tensorflow as tf


# Build a dataflow graph.
c = tf.constant([[1.0, 2.0], [3.0, 4.0]])
d = tf.constant([[1.0, 1.0], [0.0, 1.0]])
e = tf.matmul(c, d)


print(e)
```

Eager Execution的好處

- 立即返回數值，方便除錯。
- 無需 Session.run() 就可以把它們的值返回到 Python。
- 幾乎所有 TensorFlow 運算都適用。

How to avoid using tensorflow1.x

- 在學習資料看到這些函式呼叫或關鍵字代表這還是TensorFlow 1x，視情況取用。
  - tf.enable_eager_execution()：tensorflow 1.8後加入支援，但是需要用戶自己呼叫tf.enable_eager_execution()函式切換。
  - session.run
  - tf.placeholder
  - feed_dict

Installation of tensorflow

- CPU
  - conda install tensorflow=1.15.0
  - conda install tensorflow (2.x)
- GPU
  - conda install tensorflow-gpu=1.15.0
  - conda install tensorflow-gpu (2.x)
- 有自信的可以
  - 自己管gpu driver, cuda, cudnn
  - pip+virtual env

在ipython環境內切換kernel ipykernel

- 檢視 Jupyter Notebook / Lab 的核心（Kernel）
  - 前提是要先安裝 ipykernel 套件
  - jupyter kernelspec list
- 把特定的虛擬環境註冊為ipython可用的核心，
  - python -m ipykernel install --user --name myenvname
- 在colab安裝完要記得restart kernel

```
(tf1) C:\Users\ac_ya>jupyter kernelspec list
Available kernels:
  python3    C:\Users\ac_ya\.conda\envs\tf1\share\jupyter\kernels\python3

(tf1) C:\Users\ac_ya>python -m ipykernel install --user --name tf1
Installed kernelspec tf1 in C:\Users\ac_ya\AppData\Roaming\jupyter\kernels\tf1

(tf1) C:\Users\ac_ya>jupyter kernelspec list
Available kernels:
  tf1        C:\Users\ac_ya\AppData\Roaming\jupyter\kernels\tf1
  python3    C:\Users\ac_ya\.conda\envs\tf1\share\jupyter\kernels\python3
```

```
(py39-tf2-gpu) C:\Users\ac_ya>python -m ipykernel install --user --name py39-tf2-gpu
Installed kernelspec py39-tf2-gpu in C:\Users\ac_ya\AppData\Roaming\jupyter\kernels\py39-tf2-gpu

(py39-tf2-gpu) C:\Users\ac_ya>jupyter kernelspec list
Available kernels:
  py39-tf2-gpu    C:\Users\ac_ya\AppData\Roaming\jupyter\kernels\py39-tf2-gpu
  tf1             C:\Users\ac_ya\AppData\Roaming\jupyter\kernels\tf1
  python3         C:\Users\ac_ya\.conda\envs\py39-tf2-gpu\share\jupyter\kernels\python3
```

Tensor的類型

- Tensor可以被宣告為
  - 常數（Constant）
  - 變數（Variable）
  - 佔位符(Placeholder)-非即時運算模式

常數（Constant）

PYTHON

```
print("Python:")
A = 19
B = 3
```

TENSORFLOW

```
print("TensorFlow:")


x = tf.constant(19)
y = tf.constant(3)
```

數值運算

- Tensor數值運算
  - 加 +：tf.add()
  - 減 -： tf.sub()
  - 乘 *： tf.multiply()
  - 除 /： tf.divide()
  - 次方 **： tf.pow()
  - ~~求餘數 %： tf.mod()~~
  - ~~求商數 //：tf.div()~~

在非即時運算模式下，這些函數都必須在 TensorFlow 的 Session 中執行才會有運算結果的輸出，否則只是顯示張量物件的資訊而已。

數值運算(非即時運算模式)

```
x = tf.constant(19)
y = tf.constant(3)

print("TensorFlow:")
print( tf.add(x, y) )
print( tf.subtract(x, y) )
print( tf.multiply(x, y) )
print( tf.divide(x, y) )
print( tf.pow(x, y) )
print( tf.mod(x, y) )
print( tf.div(x, y) )
```

```
TensorFlow:
Tensor("Add:0", shape=(), dtype=int32)
Tensor("Sub:0", shape=(), dtype=int32)
Tensor("Mul:0", shape=(), dtype=int32)
Tensor("truediv:0", shape=(), dtype=float64)
Tensor("Pow:0", shape=(), dtype=int32)
Tensor("FloorMod:0", shape=(), dtype=int32)
WARNING:tensorflow:From <ipython-input-5-47e0c147644b>:11: div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Tensor("div:0", shape=(), dtype=int32)
```

數值運算(即時運算模式)

```
x = tf.constant(19)
y = tf.constant(3)

print("TensorFlow:")
print( tf.add(x, y) )
print( tf.subtract(x, y) )
print( tf.multiply(x, y) )
print( tf.divide(x, y) )
print( tf.pow(x, y) )
```

```
TensorFlow:
tf.Tensor(22, shape=(), dtype=int32)
tf.Tensor(16, shape=(), dtype=int32)
tf.Tensor(57, shape=(), dtype=int32)
tf.Tensor(6.333333333333333, shape=(),
dtype=float64)
tf.Tensor(6859, shape=(), dtype=int32)
```

常數張量建構函數

- tf.zeros() ：建構內容數值皆為 0 的常數張量
- tf.ones() ：建構內容數值皆為 1 的常數張量
- tf.fill() ：建構內容數值皆為特定值的常數張量
- tf.range() ：建構內容數值為 (start, limit, delta) 數列的常數張量
- tf.random.normal() ：建構內容數值為符合常態分佈數列的常數張量
- tf.random.uniform() ：建構內容數值為符合均勻分佈數列的常數張量

矩陣運算函數

- tf.reshape() ：調整矩陣外觀
- tf.eye() ：建構單位矩陣
- tf.linalg.diag() ：建構對角矩陣
- tf.linalg.matrix_transpose() ：轉置矩陣
- tf.matmul() ：矩陣相乘

變數（Variable）

- 程式設計中為了保持彈性，必須將值賦與給變數（Variables），讓使用者能夠動態地進行相同的計算來得到不同的結果，這在 TensorFlow 中是以 tf.Variable() 來完成。

重新賦值

- 初始化成功後的變數張量，可以透過 .assign() 方法重新賦予不同值。
- 重新賦值時必須要注意類型，賦予不同類型的值會得到 TypeError。
- 不僅是值的類型，外觀也必須跟當初所宣告的相同，賦予不同外觀的值會得到 ValueError。

重新賦值

```
var_tf = tf.Variable(47)
print("Before assign " ,var_tf)
var_tf.assign(24)
print("After assign " ,var_tf)
```

Python 的 with 使用方法

```
# 開啟檔案
f = open(filename)


# ...


# 關閉檔案
f.close()
```

使用檔案的過程中發生了一些例外狀況，造成程式提早跳開時，這個開啟的檔案就會沒有被關閉

```
# 開啟檔案
f = open(filename)


try:
    # ...

finally:
    # 關閉檔案
    f.close()
```

正規寫法雖然不會有問題，但是缺點就是必須手動加入關閉檔案的程式碼，不是很方便，也很容易忘記。

Python 的 with 使用方法

```
# 以 with 開檔並寫入檔案
with open('file.txt', 'w') as f:
  f.write('Hello, world!')
```

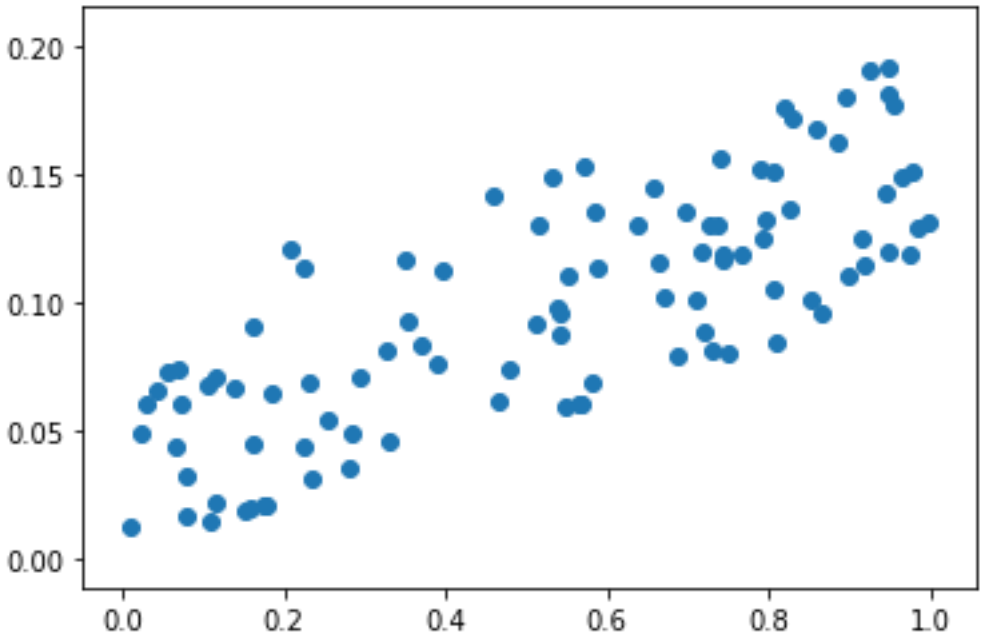
使用 with 的話，檔案使用完之後就會自動關閉

with 開啟檔案時，會將開啟的檔案一樣放在 f 變數中，但是這個 f 只有在這個 with 的範圍內可以使用，而離開這個範圍時 f 就會自動被關閉，回收相關的資源。

with 這個獨特的語法，可以來幫助使用者管理使用的資源，像是開啟的檔案、網路 socket，包含tensorflow的 session。

LinearRegression

- 在這裡鍵入方程式。範例程式：LinearRegression_tf2.ipynb
- 目的：使用線性方程式來描述這些資料點，找出線性方程式的兩個參數斜率w_1、截距w_0。
- $y = w_1 x + w_0$



$$\widehat{w_0} = \bar{y} - \widehat{w_1}\bar{x}$$

$$\widehat{w_1} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Data

```
# Random 100 points by numpy
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.1*np.random.rand(100).astype(np.float32)
def f(x):
    y = x*0.1+0.1
    return y
# plot data
plt.scatter(x_data, y_data)
plt.plot(x_data, f(x_data),  label='Ground truth')
plt.show()
```

Model building (1)

```
# Try to find values for W and b that compute y_data = a * x_data + b
# (We know that a should be 0.1 and b 0.03, but TensorFlow will
# figure that out for us.)
# Use tensorflow to find weighting of fitting

X = tf.constant(x_data)
y = tf.constant(y_data)

a = tf.Variable(initial_value=0.)
b = tf.Variable(initial_value=0.)
variables = [a, b]
```

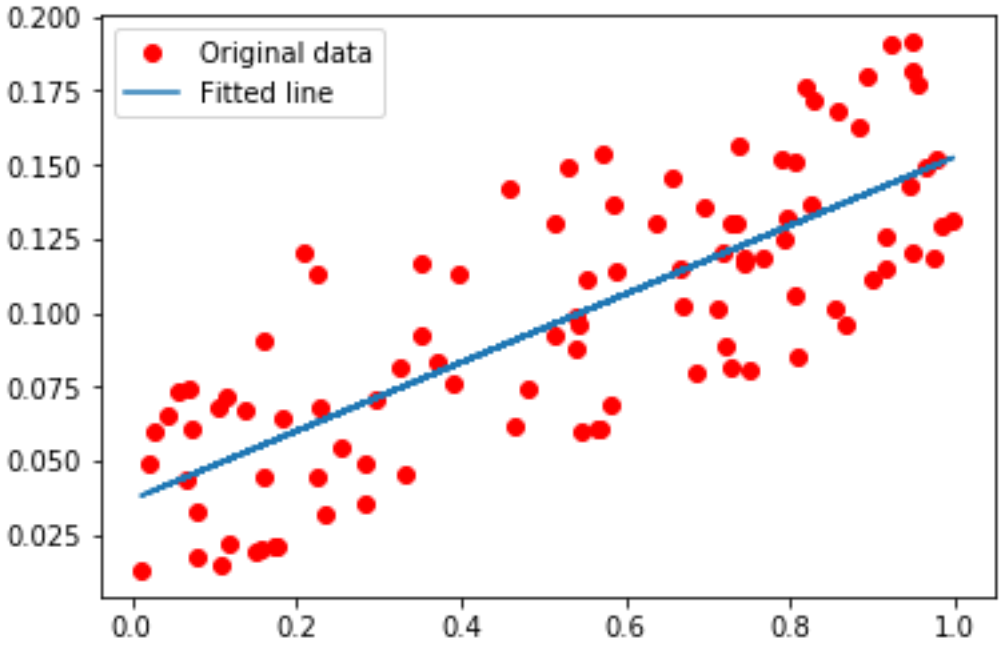
Model building (2)

```
# Define optimizer
optimizer = tf.optimizers.SGD(learning_rate=5e-4)

# 使用tf.GradientTape()記錄損失函數的梯度資訊
    with tf.GradientTape() as tape:
        y_pred = a * X + b
        loss = tf.reduce_sum(tf.square(y_pred - y))  #Minimize the mean squared errors.

    # TensorFlow自動計算損失函數關於自變數（模型參數）的梯度
    grads = tape.gradient(loss, variables)
    # TensorFlow自動根據梯度更新參數
    optimizer.apply_gradients(grads_and_vars=zip(grads, variables))
```

Training(2)

```
# Fit the line.


    if e % 50 == 0:
        print(e, a, b )
        plt.plot(x_data, y_data, 'ro', label='Original data')
        plt.plot(x_data, a * X + b, label='Fitted line')
        plt.legend()
        plt.show()
# Learns best fit is W: [0.1], b: [0.03]
```

Result



Iter:500 a:[0.10874703] b:[0.04185103]