定制基礎：張量和操作 (Customization basics: tensors and operations)

這是TensorFlow入門教程，顯示如何：

- 導入所需的包
- 創建和使用張量
- 使用GPU加速
- 演示tf.data.Dataset

This is an introductory TensorFlow tutorial that shows how to:

- Import the required package
- Create and use tensors
- Use GPU acceleration
- Demonstrate `tf.data.Dataset`

Import TensorFlow

首先，導入 `tensorflow` 模塊。 從TensorFlow 2開始，默認情況下將打開急切執行功能。 這樣可以使TensorFlow更具交互性的前端，我們將在以後詳細討論其細節。

To get started, import the `tensorflow` module. As of TensorFlow 2, eager execution is turned on by default. This enables a more interactive frontend to TensorFlow, the details of which we will discuss much later.

In [8]:

```
import tensorflow as tf
```

Tensors

張量是一個多維數組。 與NumPy的ndarray對象相似，tf.Tensor對象具有數據類型和形狀。 另外， `tf.Tensor` 可以駐留在加速器內存中（例如GPU）。 TensorFlow提供了豐富的操作庫（tf.add (https://www.tensorflow.org/api_docs/python/tf/add)，tf.matmul (https://www.tensorflow.org/api_docs/python/tf/matmul)、tf.linalg.inv (https://www.tensorflow.org/api_docs/python/tf/linalg/inv) 等，它們會消耗並產生 `tf.Tensor` 。 這些操作會自動轉換本機Python類型，例如：

A Tensor is a multi-dimensional array. Similar to NumPy `ndarray` objects, `tf.Tensor` objects have a data type and a shape. Additionally, `tf.Tensor` s can reside in accelerator memory (like a GPU). TensorFlow offers a rich library of operations (tf.add (https://www.tensorflow.org/api_docs/python/tf/add), tf.matmul (https://www.tensorflow.org/api_docs/python/tf/matmul), tf.linalg.inv (https://www.tensorflow.org/api_docs/python/tf/linalg/inv) etc.) that consume and produce `tf.Tensor` s. These operations automatically convert native Python types, for example:

In [10]:

```
print(tf.add(1, 2))
print(tf.add([1, 2], [3, 4]))
print(tf.square(5))
print(tf.reduce_sum([1, 2, 3]))

# Operator overloading is also supported
print(tf.square(2) + tf.square(3))
```

```
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor([4 6], shape=(2,), dtype=int32)
tf.Tensor(25, shape=(), dtype=int32)
tf.Tensor(6, shape=(), dtype=int32)
tf.Tensor(13, shape=(), dtype=int32)
```

每個 `tf.Tensor` 都有一個形狀和一個數據類型：

Each `tf.Tensor` has a shape and a datatype:

In [11]:

```
x = tf.matmul([[1]], [[2, 3]])
print(x)
print(x.shape)
print(x.dtype)
```

```
tf.Tensor([[2 3]], shape=(1, 2), dtype=int32)
(1, 2)
<dtype: 'int32'>
```

NumPy數組和tf.Tensor之間最明顯的區別是：

1. 張量可以由加速器內存（如GPU，TPU）支持。
2. 張量是不變的。

The most obvious differences between NumPy arrays and `tf.Tensor` s are:

1. Tensors can be backed by accelerator memory (like GPU, TPU).
2. Tensors are immutable.

NumPy兼容性(NumPy Compatibility)

在TensorFlow `tf.Tensor` s和NumPy `ndarray` 之間進行轉換很容易：

- TensorFlow操作自動將NumPy ndarray轉換為Tensors。
- NumPy操作自動將張量轉換為NumPy ndarray。

使用其 `.numpy()` 方法將張量顯式轉換為NumPy ndarray。 這些轉換通常很便宜，因為如果可能的話，數組和 `tf.Tensor` 共享底層的內存表示。 但是，共享底層表示並不總是可能的，因為 `tf.Tensor` 可能託管在GPU內存中，而NumPy數組始終由主機內存支持，並且轉換涉及從GPU到主機內存的複制。

Converting between a TensorFlow `tf.Tensor` s and a NumPy `ndarray` is easy:

- TensorFlow operations automatically convert NumPy ndarrays to Tensors.
- NumPy operations automatically convert Tensors to NumPy ndarrays.

Tensors are explicitly converted to NumPy ndarrays using their `.numpy()` method. These conversions are typically cheap since the array and `tf.Tensor` share the underlying memory representation, if possible. However, sharing the underlying representation isn't always possible since the `tf.Tensor` may be hosted in GPU memory while NumPy arrays are always backed by host memory, and the conversion involves a copy from GPU to host memory.

In [12]:

```
import numpy as np

ndarray = np.ones([3, 3])

print("TensorFlow operations convert numpy arrays to Tensors automatically")
tensor = tf.multiply(ndarray, 42)
print(tensor)


print("And NumPy operations convert Tensors to numpy arrays automatically")
print(np.add(tensor, 1))

print("The .numpy() method explicitly converts a Tensor to a numpy array")
print(tensor.numpy())
```

```
TensorFlow operations convert numpy arrays to Tensors automatically
tf.Tensor(
[[42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]], shape=(3, 3), dtype=float64)
And NumPy operations convert Tensors to numpy arrays automatically
[[43. 43. 43.]
 [43. 43. 43.]
 [43. 43. 43.]]
The .numpy() method explicitly converts a Tensor to a numpy array
[[42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]]
```

GPU 加速 (acceleration)

使用GPU進行計算可加速許多TensorFlow操作。 沒有任何註釋，TensorFlow會自動決定是使用GPU還是CPU進行操作-如有必要，在CPU和GPU內存之間複製張量。 由操作產生的張量通常由執行操作的設備的內存支持，例如：

Many TensorFlow operations are accelerated using the GPU for computation. Without any annotations, TensorFlow automatically decides whether to use the GPU or CPU for an operation—copying the tensor between CPU and GPU memory, if necessary. Tensors produced by an operation are typically backed by the memory of the device on which the operation executed, for example:

In [13]:

```
x = tf.random.uniform([3, 3])

print("Is there a GPU available: "),
print(tf.config.experimental.list_physical_devices("GPU"))

print("Is the Tensor on GPU #0:  "),
print(x.device.endswith('GPU:0'))
```

```
Is there a GPU available:
[]
Is the Tensor on GPU #0:
False
```

設備名稱(Device Names)

`Tensor.device` 屬性提供託管張量內容的設備的標準字符串名稱。 此名稱編碼許多詳細信息，例如正在執行此程序的主機的網絡地址以及該主機中的設備的標識符。 這是分佈式執行TensorFlow程序所必需的。 如果將張量放置在主機上的第 `N` 個GPU上，則字符串以 `GPU:<N>` 結尾。

The `Tensor.device` property provides a fully qualified string name of the device hosting the contents of the tensor. This name encodes many details, such as an identifier of the network address of the host on which this program is executing and the device within that host. This is required for distributed execution of a TensorFlow program. The string ends with `GPU:<N>` if the tensor is placed on the `N` -th GPU on the host.

顯式設備放置(Explicit Device Placement)

在TensorFlow中，*placement*表示如何將單個操作分配（放置）在設備上以執行。 如前所述，當沒有提供明確的指導時，TensorFlow會自動決定執行哪個設備並在需要時將張量複製到該設備上。 但是，可以使用'tf.device'上下文管理器將TensorFlow操作顯式放置在特定設備上，例如：

In TensorFlow, *placement* refers to how individual operations are assigned (placed on) a device for execution. As mentioned, when there is no explicit guidance provided, TensorFlow automatically decides which device to execute an operation and copies tensors to that device, if needed. However, TensorFlow operations can be explicitly placed on specific devices using the `tf.device` context manager, for example:

In [14]:

```
import time

def time_matmul(x):
  start = time.time()
  for loop in range(10):
    tf.matmul(x, x)

  result = time.time()-start

  print("10 loops: {:0.2f}ms".format(1000*result))

# Force execution on CPU
print("On CPU:")
with tf.device("CPU:0"):
  x = tf.random.uniform([1000, 1000])
  assert x.device.endswith("CPU:0")
  time_matmul(x)

# Force execution on GPU #0 if available
if tf.config.experimental.list_physical_devices("GPU"):
  print("On GPU:")
  with tf.device("GPU:0"): # Or GPU:1 for the 2nd GPU, GPU:2 for the 3rd etc.
    x = tf.random.uniform([1000, 1000])
    assert x.device.endswith("GPU:0")
    time_matmul(x)
```

```
On CPU:
10 loops: 118.68ms
```

數據集 (Datasets)

本部分使用 `tf.data.Dataset` API (https://www.tensorflow.org/guide/datasets)構建用於將數據饋入模型的管道。 `tf.data.Dataset` API用於從簡單，可重複使用的部分構建高性能，複雜的輸入管道，這些輸入管道將為模型的訓練或評估循環提供支持。

This section uses the `tf.data.Dataset` API (https://www.tensorflow.org/guide/datasets) to build a pipeline for feeding data to your model. The `tf.data.Dataset` API is used to build performant, complex input pipelines from simple, re-usable pieces that will feed your model's training or evaluation loops.

創建一個源 `Dataset` (Create a source `Dataset`)

Create a *source* dataset using one of the factory functions like `Dataset.from_tensors` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_tensors), `Dataset.from_tensor_slices` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_tensor_slices), or using objects that read from files like `TextLineDataset` (https://www.tensorflow.org/api_docs/python/tf/data/TextLineDataset) or `TFRecordDataset` (https://www.tensorflow.org/api_docs/python/tf/data/TFRecordDataset). See the TensorFlow Dataset guide (https://www.tensorflow.org/guide/datasets#reading_input_data) for more information.

In [15]:

```
ds_tensors = tf.data.Dataset.from_tensor_slices([1, 2, 3, 4, 5, 6])

# Create a CSV file
import tempfile
_, filename = tempfile.mkstemp()

with open(filename, 'w') as f:
  f.write("""Line 1
Line 2
Line 3
  """)

ds_file = tf.data.TextLineDataset(filename)
```

應用轉換(Apply transformations)

使用類似的轉換功能 `map` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#map), `batch` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#batch), and `shuffle` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shuffle) 將轉換應用於數據集記錄.

Use the transformations functions like `map` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#map), `batch` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#batch), and `shuffle` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shuffle) to apply transformations to dataset records.

In [9]:

```
ds_tensors = ds_tensors.map(tf.square).shuffle(2).batch(2)

ds_file = ds_file.batch(2)
```

重複 (Iterate)

`tf.data.Dataset` 對象支持迭代以遍歷記錄:

`tf.data.Dataset` objects support iteration to loop over records:

In [16]:

```
print('Elements of ds_tensors:')
for x in ds_tensors:
  print(x)

print('\nElements in ds_file:')
for x in ds_file:
  print(x)
```

```
Elements of ds_tensors:
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(5, shape=(), dtype=int32)
tf.Tensor(6, shape=(), dtype=int32)

Elements in ds_file:
tf.Tensor(b'Line 1', shape=(), dtype=string)
tf.Tensor(b'Line 2', shape=(), dtype=string)
tf.Tensor(b'Line 3', shape=(), dtype=string)
tf.Tensor(b'  ', shape=(), dtype=string)
```

In [ ]:

```
```