Eager execution basics

Run in Google (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/er
Colab

This is an introductory tutorial for using TensorFlow. It will cover:

- Importing required packages
- Creating and using Tensors
- Using GPU acceleration
- Datasets

Import TensorFlow

To get started, import the **`tensorflow`** module and enable eager execution. Eager execution enables a more interactive frontend to TensorFlow, the details of which we will discuss much later.

```
import tensorflow as tf

tf.enable_eager_execution()
```

Tensors

A Tensor is a multi-dimensional array. Similar to NumPy **`ndarray`** objects, **`Tensor`** objects have a data type and a shape. Additionally, Tensors can reside in accelerator (like GPU) memory. 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 Tensors. These operations automatically convert native Python types. For example:

```
print(tf.add(1, 2))
print(tf.add([1, 2], [3, 4]))
print(tf.square(5))
print(tf.reduce_sum([1, 2, 3]))
print(tf.encode_base64("hello world"))

# 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(b'aGVsbG8gd29ybGQ', shape=(), dtype=string)
tf.Tensor(13, shape=(), dtype=int32)
```

Each Tensor has a shape and a datatype

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

```
(1, 2)
<dtype: 'int32'>
```

The most obvious differences between NumPy arrays and TensorFlow Tensors are:

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

2. Tensors are immutable.

NumPy Compatibility

Conversion between TensorFlow Tensors and NumPy ndarrays is quite simple as:

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

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

```
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 acceleration

Many TensorFlow operations can be accelerated by using the GPU for computation. Without any annotations, TensorFlow automatically decides whether to use the GPU or CPU for an operation (and copies 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:

```
x = tf.random_uniform([3, 3])

print("Is there a GPU available: "),
print(tf.test.is_gpu_available())

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

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

Device Names

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.

Explicit Device Placement

The term "placement" in TensorFlow refers to how individual operations are assigned (placed on) a device for execution. As mentioned above, 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`** (https://www.tensorflow.org/api_docs/python/tf/device) context manager. For example:

```
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.test.is_gpu_available():
  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: 75.15ms
```

Datasets

This section demonstrates the use of the (https://www.tensorflow.org/guide/datasets) **`tf.data.Dataset`** (https://www.tensorflow.org/api_docs/python/tf/data/Dataset) API to build pipelines to feed data to your model. It covers:

- Creating a `Dataset`.
- Iteration over a `Dataset` with eager execution enabled.

We recommend using the `Datasets` API for building performant, complex input pipelines from simple, re-usable pieces that will feed your model's training or evaluation loops.

If you're familiar with TensorFlow graphs, the API for constructing the `Dataset` object remains exactly the same when eager execution is enabled, but the process of iterating over elements of the dataset is slightly simpler. You can use Python iteration over the **`tf.data.Dataset`** (https://www.tensorflow.org/api_docs/python/tf/data/Dataset) object and do not need to explicitly create an **`tf.data.Iterator`** (https://www.tensorflow.org/api_docs/python/tf/data/Iterator) object. As a result, the discussion on iterators in the TensorFlow Guide (https://www.tensorflow.org/guide/datasets) is not relevant when eager execution is enabled.

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 Guide (https://www.tensorflow.org/guide/datasets#reading_input_data) for more information.

```
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)
```

Apply transformations

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), **`shuffle`** (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shuffle) etc. to apply transformations to the records of the dataset. See the API documentation for (https://www.tensorflow.org/api_docs/python/tf/data/Dataset)**`tf.data.Dataset`** (https://www.tensorflow.org/api_docs/python/tf/data/Dataset) for details.

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

ds_file = ds_file.batch(2)
```

Iterate

When eager execution is enabled `Dataset` objects support iteration. If you're familiar with the use of `Datasets` in TensorFlow graphs, note that there is no need for calls to `Dataset.make_one_shot_iterator()` or `get_next()` calls.

```
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:
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python
Instructions for updating:
Colocations handled automatically by placer.
tf.Tensor([1 9], shape=(2,), dtype=int32)
tf.Tensor([16  4], shape=(2,), dtype=int32)
tf.Tensor([25 36], shape=(2,), dtype=int32)

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

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the Apache 2.0 License (https://www.apache.org/licenses/LICENSE-2.0). For details, see the Google Developers Site Policies (https://developers.google.com/site-policies). Java is a registered trademark of Oracle and/or its affiliates.