AI学习笔记--Tensorflow--基础数据模型

本章主要内容包含了：

- 需要导入的库
- 创建和使用Tensor
- 使用 GPU
- 数据操作

首先，我们导入一个 TensorFlow 模块并且开启执行。编写代码如下：

```
from __future__ import absolute_import, division, print_function

import tensorflow as tf

tf.enable_eager_execution()
```

Tensor

Tensor 是一个多维数组，与 NumPy 的 ndarray 对象类似，Tensor 内部包含了一些基本的数据类型和相对应的数据，并且，在实际运算过程中，这部分空间常驻在内存中以便以运算加速。Tensorflow 提供了师傅强大的函数库以便于操作和创建一个 Tensor 对象，这类操作可以自由的转换成为 Python 对象，例如代码如下：

```
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))
```

输出的结果:

```


tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor([4 6], shape=(2,), dtype=int32)
tf.Tensor(25, shape=(), dtype=int32)
W0906 16:37:25.644229 140734986499520 deprecation_wrapper.py:119] From
/Users/genesis/Workplace/Python_PyCharm_Workplace/Tensorflow/HelloTensorflow.
py:37: The name tf.encode_base64 is deprecated. Please use
tf.io.encode_base64 instead.

tf.Tensor(6, shape=(), dtype=int32)
tf.Tensor(b'aGVsbG8gd29ybGQ', shape=(), dtype=string)
tf.Tensor(13, shape=(), dtype=int32)
```

每个 Tensor 都有一个 shape 对象和一个 datatype，编辑代码如下：

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

```
print(x.dtype)
```

执行后的输出：

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

NumPy ndarrays 和 Tensorflow 的 Tensor 差异在于：

- Tensor可应用于加速计算（如 GPU 加速、TPU加速等）
- Tensor 是不可以修改的

TensorFlow 的 Tensor 和 NumPy ndarrays 转换也十分方便。

- TensorFlow 的操作函数可以自动将 NumPy ndarrays 转换成为 Tensor。
- NumPy ndarrays 也可以自动转换成为 Tensor

Tensor 可以用.numpy（）方法转换成为 ndarrays，这些转换对于性能损耗也十分低，但是由于 Tensor 的内存可能寄宿在 GPU 等物理空间内，当调用转换方法时，可能会涉及到 GPU 显存到内存再到转换的操作过程。

下例是 NumPy 和 Tensor 进行转换的例子。

```
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())
```

实际执行后输出如下：

```
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

许多 TensorFlow 的操作函数都可以通过 GPU 来进行算法加速，不需要任何注释性代码，Tensorflow 自己会决定使用 CPU 还是 GPU 作为算法运算的处理器。在一些场景下，他也可以在 CPU 和 GPU 之间进行内存的转换 copy，这类操作往往需要系统的硬件支持。

```
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'))
```

以 Mac平台为例，输出的结果如下：

```


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

我们可以实际检验一下在 CPU 和 GPU 之间的运算时间损耗，如下例：

```
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():
    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)
```

在某台带 GPU 的机器上执行的结果：

```
On CPU:
10 loops: 108.19ms
On GPU:
10 loops: 278.58ms
```