labs_DL0120EN_Eager_Execution

October 25, 2020

1 Eager Execution in TensorFlow 2.X

Estaimted time needed: **15** minutes

1.1 Objectives

After completing this lab you will be able to:

- Understand the impact of eager execution and the need to enable it

In this notebook we will overview Eager Execution in TensorFlow 2.x

Table of Contents

Instructions

Eager Execution

Tensorflow Operations Without Eager Execution Mode

Tensorflow Operations With Eager Execution Mode

Dynamic Control Flow

Thank You

2 Instructions

2.1 Installing TensorFlow

We begin by installing TensorFlow version 2.2.0 and its required prerequistes.

[ ]:
```
!pip install grpcio==1.24.3
!pip install tensorflow==2.2.0
```

Notice: This notebook has been created with TensorFlow version 2.2, and might not work with other versions. Therefore we check:

[1]:
```
import tensorflow as tf
if not tf.__version__ == '2.2.0':
    print(tf.__version__)
    raise ValueError('please upgrade to TensorFlow 2.2.0, or restart your␣
 ↪Kernel (Kernel->Restart & Clear Output)')
```

3 Eager Execution

TensorFlow's **eager execution** is an imperative programming environment that evaluates operations immediately, without building graphs, operations return concrete values instead of constructing a computational graph to run later. This makes it easy to get started with TensorFlow and debug models.

With **TensorFlow 2.x**, **Eager Execution is enabled by default**. This allows TensorFlow code to be executed and evaluated line by line. Before version 2.x was released, every graph had to be run wihthin a TensorFlow **session**. This only allowed for the entire graph to be run all at once. This made it hard to debug the computation graph.

Eager execution is a flexible machine learning platform for research and experimentation, providing:

- **An intuitive interface**-Structure your code naturally and use Python data structures. Quickly iterate on small models and small data.
- **Easier debugging**- Execute operations directly to inspect code line by line and test changes. Use standard Python debugging tools for immediate error reporting.
- **Natural control flow**—Use Python control flow instead of graph control flow, simplifying the specification of dynamic models.

As I mentioned above, in **Tensorflow 2.x**, eager execution is enabled by default. To verify that please run the below code.

[2]:
```
tf.executing_eagerly()
```

[2]:
```
True
```

Now you can run TensorFlow operations and the results will return immediately:

But first let me show you how things get done without the eager execution in tensorflow.

4 Tensorflow Operations Without Eager Execution Mode

So, there is a **disable_eager_execution()** function in TensorFlow 2.x. You can call the function like this:

[3]:
```
from tensorflow.python.framework.ops import disable_eager_execution
disable_eager_execution()
```

Note: This function can only be called at the beginning before any Graphs, Ops, or Tensors have been created. Now, verify that the eager execution is disabled or not by running the below code.

[4]:
```
tf.executing_eagerly()
```

[4]:
```
False
```

As you can see **False** in the output that means it is disabled now.

Just execute the next cell. You will notice that we've created an object **a** of type **tensorflow.python.framework.ops.Tensor**

[8]:
```
import numpy as np
a = tf.constant(np.array([1., 2., 3.]))
type(a)
```

[8]:
```
tensorflow.python.framework.ops.Tensor
```

Let's create another Tensor **b** and apply the dot product between them. This gives us **c**

[11]:
```
b = tf.constant(np.array([4.,5.,6.]))
c = tf.tensordot(a, b, 1)
type(c)
```

[11]:
```
tensorflow.python.framework.ops.Tensor
```

[12]:
```
print(c)
```

```
Tensor("Tensordot:0", shape=(), dtype=float64)
```

Note that **c** is a **tensorflow.python.framework.ops.Tensor** as well. So any node of the execution graph resembles a Tensor type. **But so far not a single computation happened**. You need to execute the graph. You can pass any graph or subgraph to the TensorFlow runtime for execution. Each TensorFlow graph runs within a TensorFlow Session, therefore we need to create it first:

Note: Session can be accessed via **tf.compat.v1.Session()** in Tensorflow 2.x.

[13]:
```
session = tf.compat.v1.Session()
output = session.run(c)
session.close()
print(output)
```

```
32.0
```

Now you see the correct result of 32. But the problem is that debugging is pretty hard if you can only run complete graphs.

So let's actually re-enable the **eager execution**.

4.1 Tensorflow Operations With Eager Execution Mode

4.1.1 IMPORTANT! => Please don't forget restart the kernel by clicking on "Kernel"->"Restart" so that the changes take effect.

Enable or Disable Eager execution has to happen on program startup. This is the reason we have to restart the kernel.

Import the required libraries again.

[1]:
```
import numpy as np
import tensorflow as tf
```

Run the below command to re-enable the eager execution.

[3]:
```
from tensorflow.python.framework.ops import enable_eager_execution
enable_eager_execution()
```

Now you can run TensorFlow operations and the results will return immediately:

[4]:
```
x = [[4]]
m = tf.matmul(x, x)
print("Result, {}".format(m))
```

```
Result, [[16]]
```

Enabling eager execution changes how TensorFlow operations behave—now they immediately evaluate and return their values to Python.

Since there isn't a computational graph to build and run later in a session, it's easy to inspect results using print() or a debugger.

[7]:
```
a = tf.constant(np.array([1., 2., 3.]))
type(a)
```

[7]:
```
tensorflow.python.framework.ops.EagerTensor
```

So the very same code created a different type of object. So now **a** is of type **tensorflow.python.framework.ops.EagerTensor**. This is great, because without changing code we obtain a tensor object which allows us to have a look inside, without executing a graph in a session:

[8]:
```
print(a.numpy())
```

```
[1. 2. 3.]
```

Isn't this amazing? So from now on we can threat Tensors like ordinary python objects, work with them as usual, insert debug statements at any point or even use a debugger. So let's continue this example:

[9]:
```
b = tf.constant(np.array([4.,5.,6.]))
c = tf.tensordot(a, b, 1)
type(c)
```

[9]:
```
tensorflow.python.framework.ops.EagerTensor
```

Again, **c** is an **tensorflow.python.framework.ops.EagerTensor** object which can be directly read:

[10]:
```
print(c.numpy())
```

```
32.0
```

Without creating a session or a graph we obtained the result of the defined computation.

5 Dynamic Control Flow

A major benefit of eager execution is that all the functionality of the host language is available while your model is executing. So, for example, it is easy to write fizzbuzz:

[17]:
```
def fizzbuzz(max_num):
  max_num = tf.convert_to_tensor(max_num)
  for num in range(1, max_num.numpy()+1):
    num = tf.constant(num)
    if int(num % 3) == 0 and int(num % 5) == 0:
      print('FizzBuzz')
    elif int(num % 3) == 0:
      print('Fizz')
    elif int(num % 5) == 0:
      print('Buzz')
    else:
      print(num.numpy())
```

[18]:
```
fizzbuzz(15)
```

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

Thank you for completing this notebook

5.1 Author

Shubham Yadav

5.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-04	1.0	Lavanya	Added lab to demonstrate Tensorflow eager execution

##