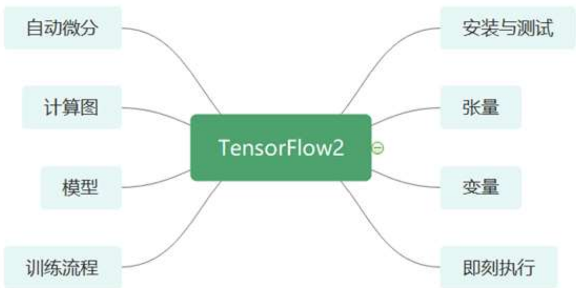
TensorFlow2

计算图

Graph

导学

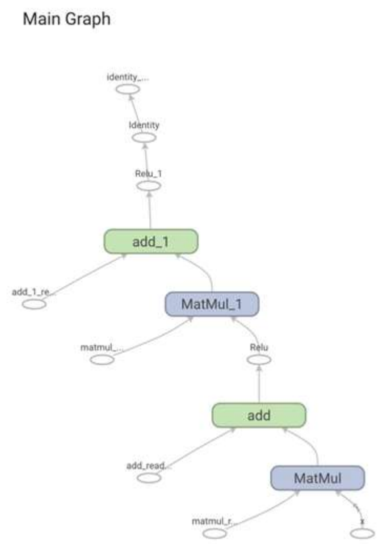


Graph简介

TensorFlow2

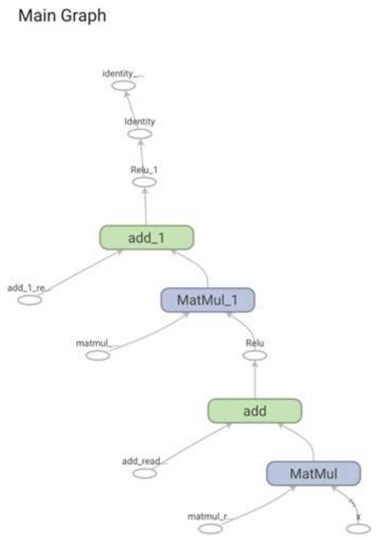
Graph（图）简介

- Graph（图）是包含一系列tensorflow操作（tf.Operation对象）的数据结构，这些操作代表计算单位。
- Graph包含（tf.Tensor对象）Tensor（张量），代表操作之间流动的数据单位
- 它们是在tf.Graph上下文中定义的。



Graph（计算图）简介

- 计算图的优势
  - 计算图拥有很大的灵活性
  - 计算图执行效率高
  - 计算图容易优化



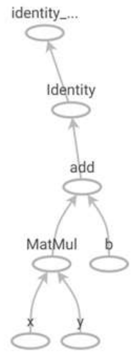
Graph的建立

TensorFlow2

Graph（图）的建立

- 利用tf.function建立图并进行追踪
- tf.function功能化的函数是Python可调用函数，其功能与Python等效函数相同

```python
# Define a Python function
def function_to_get_faster(x, y, b):
  x = tf.matmul(x, y)
  x = x + b
  return x

# Create a `Function` object that contains a graph
a_function_that_uses_a_graph = tf.function(function_to_get_faster)

# Make some tensors
x1 = tf.constant([[1.0, 2.0]])
y1 = tf.constant([[2.0], [3.0]])
b1 = tf.constant(4.0)

# It just works!
a_function_that_uses_a_graph(x1, y1, b1).numpy()
```



Graph（图）的建立

- tf.function 可以递归地跟踪它调用的任何Python函数

```python
def inner_function(x, y, b):
  x = tf.matmul(x, y)
  x = x + b
  return x

# Use the decorator
@tf.function
def outer_function(x):
  y = tf.constant([[2.0], [3.0]])
  b = tf.constant(4.0)

  return inner_function(x, y, b)

# Note that the callable will create a graph that
# includes inner_function() as well as outer_function()
outer_function(tf.constant([[1.0, 2.0]])).numpy()
```

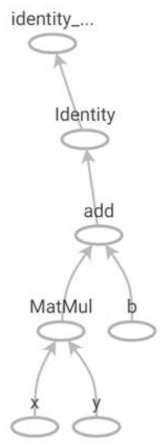
Graph（图）流控制

- tf.autograph默认情况下，流控制和循环会转换为TensorFlow

```
def my_function(x):
  if tf.reduce_sum(x) <= 1:
    return x * x
  else:
    return x-1

a_function = tf.function(my_function)

print("First branch, with graph:", a_function(tf.constant(1.0)).numpy())
print("Second branch, with graph:", a_function(tf.constant([5.0, 5.0])).numpy())
```

利用Tensorboard显示图

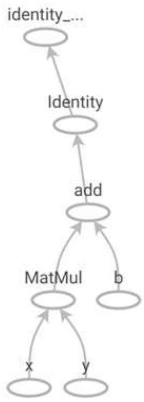
```
%load_ext tensorboard
def inner_function(x, y, b):
  x = tf.matmul(x, y)
  x = x + b
  return x

# Use the decorator
@tf.function
def outer_function(x):
  y = tf.constant([[2.0], [3.0]])
  b = tf.constant(4.0)

  return inner_function(x, y, b)
```



利用Tensorboard显示图

```python
# Set up logging.
stamp = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = 'logs/func/%s' % stamp
writer = tf.summary.create_file_writer(logdir)
# Bracket the function call with
# tf.summary.trace_on() and tf.summary.trace_export().
tf.summary.trace_on(graph=True, profiler=True)
# Call only one tf.function when tracing.
outer_function(tf.constant([[1.0,2.0]]))
with writer.as_default():
  tf.summary.trace_export(
      name="my_func_trace",
      step=0,
      profiler_outdir=logdir)
```

```
%tensorboard --logdir logs/func
```



Graph的特点

TensorFlow2

Graph（图）的加速

- 对于复杂的计算，图形可以显着提高速度。这是因为图形减少了Python与设备之间的通信并加快了速度。
- 仅包装使用张量的函数tf.function并不会自动加快代码速度。对于在单个计算机上多次调用的小函数，调用图形或图形片段的开销可能会占主导地位。同样，如果大多数计算已经在加速器上进行，例如大量GPU的卷积，则图形加速不会很大。

Graph（图）的加速

```
# Create an oveerride model to classify pictures
class SequentialModel(tf.keras.Model):
  def __init__(self, **kwargs):
    super(SequentialModel, self).__init__(**kwargs)
    self.flatten = tf.keras.layers.Flatten(input_shape=(28, 28))
    self.dense_1 = tf.keras.layers.Dense(128, activation="relu")
    self.dropout = tf.keras.layers.Dropout(0.2)
    self.dense_2 = tf.keras.layers.Dense(10)

  def call(self, x):
    x = self.flatten(x)
    x = self.dense_1(x)
    x = self.dropout(x)
    x = self.dense_2(x)
    return x

input_data = tf.random.uniform([60, 28, 28])

eager_model = SequentialModel()
graph_model = tf.function(eager_model)

print("Eager time:", timeit.timeit(lambda: eager_model(input_data), number=10000))
print("Graph time:", timeit.timeit(lambda: graph_model(input_data), number=10000))
```

多态函数

- 跟踪函数时，将创建一个多态的Function对象。
- tf.function上可以使用不同类型和形状的数据

```
print(a_function)

print("Calling a `Function`:")
print("Int:", a_function(tf.constant(2)))
print("Float:", a_function(tf.constant(2.0)))
print("Rank-1 tensor of floats", a_function(tf.constant([2.0, 2.0, 2.0])))
```

Graph（图）恢复eager模式

- 在模型调试阶段，Eager模式更易于debug
- 恢复eager模式方法：
  - 直接调用模型和图层
  - 使用Keras编译/拟合时，在编译时使用 model.compile(run_eagerly=True)
  - 通过设置全局执行模式 tf.config.run_functions_eagerly(True)

谢谢指正！