Tensorflow框架实现中的“三”种图

张宸 · 4 个月前

图（graph）是 tensorflow 用于表达计算任务的一个核心概念。从前端（python）描述神经网络的结构，到后端在多机和分布式系统上部署，到底层 Device（CPU、GPU、TPU）上运行，都是基于图来完成。然而我在实际使用过程中遇到了三对API，

1. tf.train.Saver()/saver.restore()
2. export_meta_graph/Import_meta_graph
3. tf.train.write_graph()/tf.Import_graph_def()

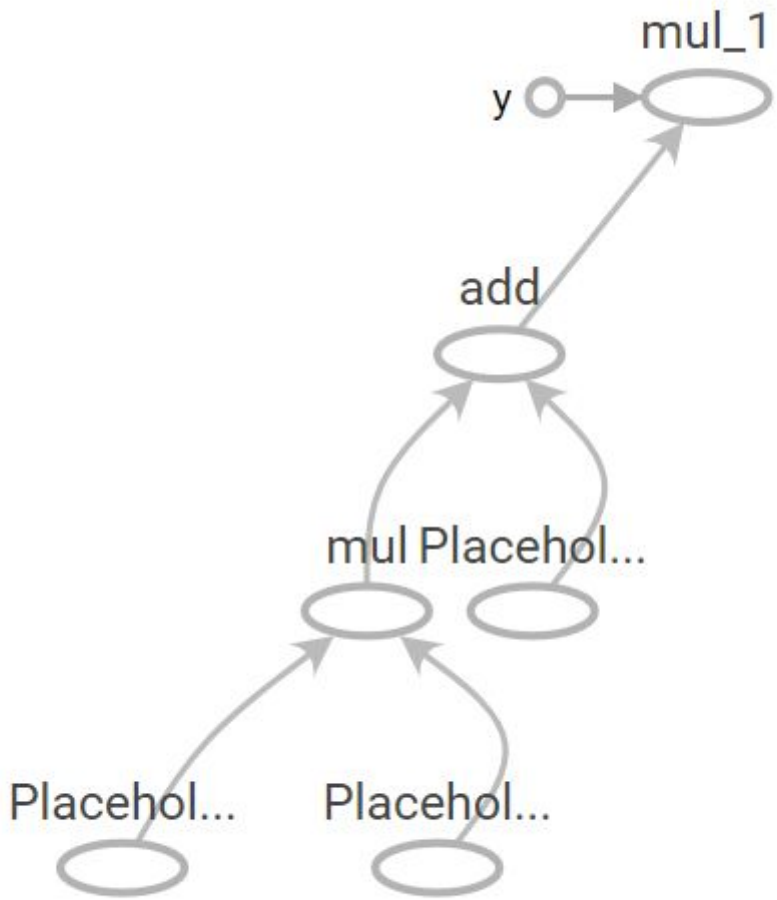
他们都是用于对图的保存和恢复。**同一个计算框架，为什么需要三对不同的API呢？他们保存/恢复的图在使用时又有什么区别呢？**初学的时候，常常闹不清楚他们的区别，以至常常写出了

Graph

首先介绍一下关于 Tensorflow 中 Graph 和它的序列化表示 Graph_def。在Tensorflow的官方文档中，Graph 被定义为“一些 Operation 和 Tensor 的集合”。例如我们表达如下的一个计算的python代码，

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
c = tf.placeholder(tf.float32)
d = a*b+c
e = d*2
```

就会生成相应的一张图，在Tensorboard中看到的图大概如下这样。其中每一个圆圈表示一个Operation（输入处为Placeholder），椭圆到椭圆的边为Tensor，箭头的指向表示了这张图Operation 输入输出 Tensor 的传递关系。



这张图所表达的数据流 与 python 代码中所表达的计算是对应的关系（为了称呼方便，我们下面将这张由Python表达式所描述的数据流动关系叫做 Python Graph）。然而在真实的Tensorflow 运行中，Python 构建的“图”并不是启动一个Session之后始终不变的东西。因为

上进行高性能/能效的计算。单纯使用 Python 肯定是无法有效完成的。实际上，Tensorflow而是首先将 python 代码所描绘的图转换（即“序列化”）成 Protocol Buffer，再通过 C/C++/CUDA 运行 Protocol Buffer 所定义的图。（Protocol Buffer的介绍可以参考这篇文章学习：ibm.com/developerworks/... ）

GraphDef

从 python Graph中序列化出来的图就叫做 GraphDef（这是一种不严格的说法，先这样进行理解）。而 GraphDef 又是由许多叫做 NodeDef 的 Protocol Buffer 组成。在概念上 NodeDef 与（Python Graph 中的）Operation 相对应。如下就是 GraphDef 的 ProtoBuf，由许多node组成的图表示。这是与上文 Python 图对应的 GraphDef：

```
node {
  name: "Placeholder"     # 注释：这是一个叫做 "Placeholder" 的node
  op: "Placeholder"
  attr {
    key: "dtype"
    value {
      type: DT_FLOAT
    }
  }
  attr {
    key: "shape"
    value {
```

```
      }
    }
  }
}
node {
  name: "Placeholder_1"     # 注释：这是一个叫做 "Placeholder_1" 的node
  op: "Placeholder"
  attr {
    key: "dtype"
    value {
      type: DT_FLOAT
    }
  }
  attr {
    key: "shape"
    value {
      shape {
        unknown_rank: true
      }
    }
  }
}
node {
  name: "mul"                 # 注释：一个 Mul（乘法）操作
  op: "Mul"
  input: "Placeholder"        # 使用上面的node（即Placeholder和Placeholder_1）
  input: "Placeholder_1"      # 作为这个Node的输入
  attr {
    key: "T"
```

```
    }
  }
}
```

以上三个 NodeDef 定义了两个Placeholder和一个Multiply。Placeholder 通过 attr（attribute的缩写）来定义数据类型和 Tensor 的形状。Multiply通过 input 属性定义了两个placeholder作为其输入。无论是 Placeholder 还是 Multiply 都没有关于输出（output）的信息。其实 Tensorflow 中都是通过 Input 来定义 Node 之间的连接信息。

那么既然 tf.Operation 的序列化 ProtoBuf 是 NodeDef，**那么 tf.Variable 呢？在这个 GraphDef 中只有网络的连接信息，却没有任何 Variables呀？没错，Graphdef 中不保存任何 Variable 的信息，所以如果我们从 graph_def 来构建图并恢复训练的话，是不能成功的。**比如以下代码，

```
with tf.Graph().as_default() as graph:
  tf.import_graph_def("graph_def_path")
  saver= tf.train.Saver()
  with tf.Session() as sess:
    tf.trainable_variables()
```

其中 tf.trainable_variables() 只会返回一个空的list。Tf.train.Saver() 也会报告 no variables to save。

然而，在实际线上 inference 中，通常就是使用 GraphDef。**然而，GraphDef中连Variable都没有，怎么存储weight呢？原来GraphDef 虽然不能保存 Variable，但可以保存 Constant 。**通过 tf.constant 将 weight 直接存储在 NodeDef 里，tensorflow 1.3.0 版本也提供了一套叫做

tensorflow.org/extend/t...

tensorflow.org/mobile/p...

tf.train.write_graph()/tf.Import_graph_def() 就是用来进行 GraphDef 读写的API。那么，我们怎么才能从序列化的图中，得到 Variables呢？这就要学习下一个重要概念，MetaGraph。

MetaGraph

Meta graph 的官方解释是：一个Meta Graph 由一个计算图和其相关的元数据构成。其包含了用于继续训练，实施评估和（在已训练好的的图上）做前向推断的信息。（A MetaGraph consists of both a computational graph and its associated metadata. A MetaGraph contains the information required to continue training, perform evaluation, or run inference on a previously trained graph. From <tensorflow.org/versions...> ）

这一段看的云里雾里，不过这篇文章（tensorflow.org/versions... ）进一步解释说，Meta Graph在具体实现上就是一个MetaGraphDef（同样是由 Protocol Buffer来定义的）。其包含了四种主要的信息，根据Tensorflow官网，这四种 Protobuf 分别是

1. MetaInfoDef，存一些元信息（比如版本和其他用户信息）
2. GraphDef， MetaGraph的核心内容之一，我们刚刚介绍过
3. SaverDef，图的Saver信息（比如最多同时保存的check-point数量，需保存的Tensor名字等，但并不保存Tensor中的实际内容）
4. CollectionDef 任何需要特殊注意的 Python 对象，需要特殊的标注以方便

在以上四种 ProtoBuf 里面，1 和 3 都比较容易理解，2 刚刚总结过。这里特别要讲一下 Collection（CollectionDef是对应的ProtoBuf）。

Tensorflow中并没有一个官方的定义说 collection 是什么。简单的理解，它就是为了方别用户对图中的操作和变量进行管理，而创建的一个概念。它可以说是一种“集合”，通过一个 key（string类型）来对一组 Python 对象进行命名的集合。这个key既可以是tensorflow在内部定义的一些key，也可以是用户自己定义的名字（string）。

Tensorflow 内部定义了许多标准 Key，全部定义在了 tf.GraphKeys 这个类中。其中有一些常用的，tf.GraphKeys.TRAINABLE_VARIABLES, tf.GraphKeys.GLOBAL_VARIABLES 等等。tf.trainable_variables() 与 tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES) 是等价的；tf.global_variables() 与 tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES) 是等价的。

对于用户定义的 key，我们举一个例子。例如：

```
pred = model_network(X)
loss=tf.reduce_mean(…, pred, …)
train_op=tf.train.AdamOptimizer(lr).minimize(loss)
```

这样一段 Tensorflow程序，用户希望特别关注 pred, loss train_op 这几个操作，那么就可以使用如下代码，将这几个变量加入到 collection 中去。(假设我们将其命名为 “training_collection”)

```
tf.add_to_collection("training_collection", pred)
tf.add_to_collection("training_collection", loss)
```

并且可以通过 Train_collect = tf.get_collection(“training_collection”) 得到一个python list，其中的内容就是 pred, loss, train_op的 Tensor。这通常是为了在一个新的 session 中打开这张图时，方便我们获取想要的操作。比如我们可以直接工通过get_collection() 得到 train_op，然后通过 sess.run(train_op)来开启一段训练，而无需重新构建 loss 和optimizer。

通过export_meta_graph保存图，并且通过 add_to_collection 将 train_op 加入到 collection 中：

```
with tf.Session() as sess:
  pred = model_network(X)
  loss=tf.reduce_mean(…,pred, …)
  train_op=tf.train.AdamOptimizer(lr).minimize(loss)
  tf.add_to_collection("training_collection", train_op)
  Meta_graph_def =
      tf.train.export_meta_graph(tf.get_default_graph(), 'my_graph.meta')
```

通过 import_meta_graph将图恢复（同时初始化为本 Session的 default 图），并且通过 get_collection 重新获得 train_op，以及通过 train_op 来开始一段训练（ sess.run() ）。

```
with tf.Session() as new_sess:
  tf.train.import_meta_graph('my_graph.meta')
  train_op = tf.get_collection("training_collection")[0]
  new_sess.run(train_op)
```

更多的代码例子可以在这篇文档（tensorflow.org/api_guid... ）中的 Import a MetaGraph 章节中看到。

那么，从 Meta Graph 中恢复构建的图可以被训练吗？是可以的。Tensorflow的官方文档 tensorflow.org/api_guid... 说明了使用方法。这里要特殊的说明一下，**Meta Graph中虽然包含Variable的信息，却没有 Variable 的实际值。所以从Meta Graph中恢复的图，其训练是从随机初始化的值开始的。训练中Variable的实际值都保存在check-point中，如果要从之前训练的状态继续恢复训练，就要从check-point中restore。**进一步读一下Export Meta Graph的代码，可以看到，事实上variables并没有被export到meta_graph 中

github.com/tensorflow/t...（1872行）

github.com/tensorflow/t... (829，845行)

export_meta_graph/Import_meta_graph 就是用来进行 Meta Graph 读写的API。**tf.train.saver.save() 在保存check-point的同时也会保存Meta Graph。但是在恢复图时，tf.train.saver.restore() 只恢复 Variable，如果要从MetaGraph恢复图，需要使用import_meta_graph。**这是其实为了方便用户，有时我们不需要从MetaGraph恢复的图，而是需要在 python 中构建神经网络图，并恢复对应的 Variable。

Check-point

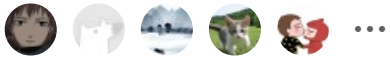
Check-point 里全面保存了训练某时间截面的信息，包括参数，超参数，梯度等等。tf.train.Saver()/saver.restore() 则能够完完整整保存和恢复神经网络的训练。Check-point分为

总结

Tensorflow 三种 API 所保存和恢复的图是不一样的。这三种图是从Tensorflow框架设计的角度出发而定义的。但是从用户的角度来看，TF文档的写作难免有些云里雾里，弄不清他们的区别。需要读一读Tensorflow的代码，做一些实验来对他们进行辨析。

简而言之，Tensorflow 在前端 Python 中构建图，并且通过将该图序列化到 ProtoBuf GraphDef，以方便在后端运行。在这个过程中，图的保存、恢复和运行都通过 ProtoBuf 来实现。GraphDef，MetaGraph，以及Variable，Collection 和 Saver 等都有对应的 ProtoBuf 定义。ProtoBuf 的定义也决定了用户能对图进行的操作。例如用户只能找到Node的前一个Node，却无法得知自己的输出会由哪个Node接收。

深度学习（Deep Learning）



9 条评论

立冬

当他无法实现动态图的时候就不适合做机器学习的人用了吧，实现一个复杂loss基本等于要命。。也许是我矩阵运算没学到家也说不定

4 个月前

沿途的笔记

暑假折腾了好几天，作者总结的真好

孙小晗

好棒！膜拜

4 个月前

陈朝才 回复 **立冬**

查看对话

动态和静态没啥本质区别啊

4 个月前

肖扬

作者山西人？

4 个月前

李轩辕

我觉得tf官文挺好的啊，大而全😂

4 个月前

小白将

题主总结的不错，其实graph本质上统一的。pb文件，其实就是graph_def，但是指的一般是做了constant化，这样可以直接加载做inference，安卓部署用。另外一个是check_point文件，其实包含三个主要文件，meta, index, data，meta主要有各种def，一个很重要的就是graph_def，而data保存真正的weight。还有一个是tf serving里面的saved model，感兴趣可以了解。其实三者都可以相互转化，本质上都是图定义加数据

宋伟

写得很赞！有个问题想请教下，文中提到“从Meta Graph中恢复的图，其训练是从随机初始化的值开始的。训练中Variable的实际值都保存在check-point中，如果要从之前训练的状态继续恢复训练，就要从check-point中restore" ， 以神经网络为例，我理解如果要重新训练的话，神经网络中的所有参数（各种embedding、weight、bias）全都要从check-point中restore，那每个变量都要restore的话不是很麻烦吗，不是很确定restore 每个变量的方法。。。是先add_to_collection然后get_collection 就可以获得之前训练得到的参数值吗？多谢！

14 天前

张宸（作者） 回复 **宋伟** 查看对话

谢谢！我们在tensorflow中不需要逐个添加variable到collection中，tensorflow自带了一些collection，比如 tf.trainable_variables() 返回的就是所有 trainable 的 variable, tf.all_variables() 返回的就是所有的variable。我们在声明 save 对象的时候，tensorflow 默认是save all variable的。

13 天前

推荐阅读

TensorFlow中的一些细节

首发于
Tensorflow自习室

ant是… 查看全文 >

黄璞 · 6 个月前

TensorFlow初步（1）

大家好我是zyy，本人是机器学习和深度学习的初学爱好者，想跟大家一起分享我的学习经验，大家一起交流。我写的东西不一定全对，但肯定是我一步一步走出来的坑，嚼烂了的经验，可以供大家直… 查看全文 >

Mr.张 · 1 年前 · 发表于 zyy的机器学习与深度学习之路

TensorFlow初步(5)

大家好我是zyy，本人是机器学习和深度学习的初学爱好者，想跟大家一起分享我的学习经验，大… 查看全文 >

Mr.张 · 1 年前 · 发表于 zyy的机器学习与深度学习之路

深度学习框架TensorFlow学习笔记（1）

本文为学习TensorFlow时的一些笔记和注意事项。 1.TensorFlow的基本使用 使用图来表示计算… 查看全文 >

马天猫Masn · 9 个月前