樂不思蜀
我留，我走，我是一个停顿~

TensorFlow Graph Transform Tool(GTT)

2017-05-14 18:41:20 | 分类： TensorFlow | 标签： tensorflow deeplearning

1、引言

当你完成训练模型，要在生产中部署的时候，我们更希望通过修改优化这个模型，使其在最终的环境里可以运行的更好。例如当你在一个移动端进行部署时，你可能想通过量化权重来压缩文件大小，或者优化掉批处理规则或其它训练时才有的特性。GTT框架提供了一系列的工具去修改或优化计算图，而且它是一个很容易让你按照自己的思路进行修改的框架。

本文分为三个部分：第一部分给出几个如何去执行普通任务的教程；第二部分是一份参考，它几乎涵盖所有转换模型和应用在这些转换模型上的选项；第三部分是如何运用GTT来自定义自己的图转换。

2、Using the Graph Transform Tool

GTT是设计用在训练模型上的，这些模型一般被保存成二进制的protobuf格式的图定义文件。这是一种低层次定义的Tensorflow计算图，它包含了节点清单和它们之间的输入输出连接。如果你用Python API训练模型，这个图定义文件经常会被保存在和你的checkpoints相同的目录下，并且通常以'.pb'为文件后缀。

如果你想处理训练参数的值，例如量化权重，那么你需要运行tensorflow/python/tools/freeze_graph.py 脚本把checkpoint值转换成图文件中的嵌入常量。

你可以调用GTT如下列方式：

```
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
--in_graph=tensorflow_inception_graph.pb \
--out_graph=optimized_inception_graph.pb \
--inputs='Mul:0' \
--outputs='softmax:0' \
--transforms='
strip_unused_nodes(type=float, shape="1,299,299,3")
remove_nodes(op=Identity, op=CheckNumerics)
fold_old_batch_norms'
```

上述参数指定了输入图文件、转换后的输出图文件、输入和输出层，以及什么操作去转换图。转换操作是以列表的形式给出，并且每种操作本身也有参数。为了产生输出，这些转换操作定义了被应用修改的pipeline。有时候有些转换操作要先于其它的操作，而操作列表允许你指定这些转换操作的顺序。注意remove_nodes(op=Identity, op=CheckNumerics)这条优化指令和控制流操作(such as tf.cond, tf.map_fn, and tf.while一样会打破模型。

3、Inspecting Graphs

GTT工具支持的很多转换都需要知道模型的输出和输出层，解决这个最好的方式是模型的训练过程，从训练过程中可以知道对于一个分类器而言，输入就是从训练集接收数据的节点，输出就是那些预测节点。如果你不确定输入和输出节点，summarize_graph工具可以检查模型并提供很可能是输入输出节点的猜测，以及一些其它对调试有用的信息。下面是一个如何把summarize_graph工具用在Inception V3 graph模型上的例子：

```
bazel build tensorflow/tools/graph_transforms:summarize_graph
bazel-bin/tensorflow/tools/graph_transforms/summarize_graph
--in_graph=tensorflow_inception_graph.pb
```

4、Common Use Cases

这部分包含了一些被最频繁使用的转换pipelines的使用指南，目的在于帮助想快速完成这些转换任务的用户。很多转换pipelines例子使用的是the Inception V3 model。

Optimizing for Deployment(优化部署)

如果我们完成了模型训练，并想把它部署在一个服务器或移动设备上，那么我们希望可以尽可能快速地加载运行模型，并且尽可能少的减少不太重要的依赖库。下面这个例子就是移除了那些所有在inference阶段不会被调用的节点，通过在单节点中常量化来缩减表达式，并且在batch normalization过程中通过为卷积操作提前乘以权重的方式操作来优化掉一些乘法操作。举例如下：

```
bazel build tensorflow/tools/graph_transforms:transform_graph
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
--in_graph=tensorflow_inception_graph.pb \
--out_graph=optimized_inception_graph.pb \
--inputs='Mul' \
--outputs='softmax' \
--transforms='
 strip_unused_nodes(type=float, shape="1,299,299,3")
 remove_nodes(op=Identity, op=CheckNumerics)
 fold_constants(ignore_errors=true)
```

```
 fold_old_batch_norms'
```

batch norm folding操作一般是包含两次的，因为在TensorFlow中有两种不同的batch normalization。旧版本是用单一的BatchNormWithGlobalNormalization op实现的，但是在最近新的方法中是被舍弃了，然后使用了一些单独的ops去实现同样的运算。因为有这两种版本的转换，所以它们都会被GTT识别和优化。

Fixing Missing Kernel Errors on Mobile

移动版本的TensorFlow专注于推理预测，因此默认所支持操作(定义在tensorflow/core/kernels/BUILD:android_extended_ops for Bazel和tensorflow/contrib/makefile/tf_op_files.txt for make builds 两个地方)的清单不包含很多训练相关的操作。这样在加载图的时候，会导致No OpKernel was registered to support Op 的错误，即使这个op没有被执行。

如果你遇到了这样的错误，并且这个op确实是想要在移动端运行的，那么你需要局部的修改工程构建文件，使其包含right .cc 文件。很多情况下，这样的op只是训练过程中的残余物，你可以通过使用strip_unused_nodes 命令选项，并指定预测用途的输入输出来删掉这些没必要的节点。举例如下：

```
bazel build tensorflow/tools/graph_transforms:transform_graph
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
--in_graph=tensorflow_inception_graph.pb \
--out_graph=optimized_inception_graph.pb \
--inputs='Mul' \
--outputs='softmax' \
--transforms='
 strip_unused_nodes(type=float, shape="1,299,299,3")
 fold_constants(ignore_errors=true)
 fold_batch_norms
 fold_old_batch_norms'
```

Shrinking File Size

如果你希望部署自己的模型到移动端作为app的一部分，那么保证下载文件尽可能的小时很重要的TensorFlow模型来说，造成文件大小的最主要的原因是传递到卷积层和全连接层的权重，因此任何可以方式都是有用的。幸运的是很多神经网络的抗干扰能力强，这就可以通过降低这些权重的精度去存储它经网络的精度损失不大。

无论是IOS还是Android系统，它们的app包下载之前都是经过压缩的，因此减少用户所需带宽最简单的方式就是提供更容易压缩的原始数据。默认情况下，训练权重存储成浮点值，即使数字之间微小的差别也会导致非常不同的位模式，因此这些值不能很好的被压缩。如果近似这些权重，使附近的其他权重也存储成相同的近似值，结果bit流就会有许多重复的值，这样压缩会 更高效。可以通过运行round_weights 转换方式来做到这点：

```
bazel build tensorflow/tools/graph_transforms:transform_graph
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
--in_graph=tensorflow_inception_graph.pb \
--out_graph=optimized_inception_graph.pb \
--inputs='Mul' \
--outputs='softmax' \
--transforms='
 strip_unused_nodes(type=float, shape="1,299,299,3")
```

```
 fold_batch_norms
 fold_old_batch_norms
 round_weights(num_steps=256)'
```

运行后你会看到输出文件optimized_inception_graph.pb 和输入文件一样大小，但是如果你执行zip命令去压缩它，你会发现至少比压缩原输入文件小了70%的大小。最棒的是这个转换操作不改变任何一点的图结构，因此转换后该图的运行和转换之前有着一模一样的运算、延迟和内存占用。你可以调整num_steps参数来控制每个权重缓冲近似到多少个值，参数越小，以准确率为代价的压缩率越高。

进一步的，我们可以直接地将权重存储成8-bit的值，下面是示例代码：

```
bazel build tensorflow/tools/graph_transforms:transform_graph
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
--in_graph=tensorflow_inception_graph.pb \
--out_graph=optimized_inception_graph.pb \
--inputs='Mul' \
--outputs='softmax' \
--transforms='
 strip_unused_nodes(type=float, shape="1,299,299,3")
 fold_constants(ignore_errors=true)
 fold_batch_norms
 fold_old_batch_norms
 quantize_weights'
```

运行后会发现输出图文件大约是输入文件的1/4大小，与round_weights 方式相比，这种方式的缺点就是需要插入一个把8-bit值转换成浮点数的额外解压缩操作，但是在TensorFlow运行时的优化会确保这些结果被缓存起来，因此你不会看到图的运行有任何变慢。

到目前为止我们都是专注于训练权重，因为这些权重占据了大部分的空间。如果有一张具有许多小节点的图，那这些节点的名字也会占据不容忽视的空间。为了压缩这些空间，我们可以调用obfuscate_names 转换方式，这种方式会以短小、含义模糊却独一无二的ID替换掉这些节点名字(除了输入和输出之外)：

```
bazel build tensorflow/tools/graph_transforms:transform_graph
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
--in_graph=tensorflow_inception_graph.pb \
--out_graph=optimized_inception_graph.pb \
--inputs='Mul:0' \
--outputs='softmax:0' \
--transforms='
 obfuscate_names'
```

8-bit Calculations

对许多平台来说，能够做尽可能多的8-bit数运算而不是浮点数运算，是很有好处的。TensorFlow中对这种方式运算的支持还处于试验和改进阶段，不过你可以用GTT工具把模型转换成可量化的，代码如下：

```
bazel build tensorflow/tools/graph_transforms:transform_graph
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
```

```
--out_graph=optimized_inception_graph.pb \
--inputs='Mul' \
--outputs='softmax' \
--transforms='
 add_default_attributes
 strip_unused_nodes(type=float, shape="1,299,299,3")
 remove_nodes(op=Identity, op=CheckNumerics)
 fold_constants(ignore_errors=true)
 fold_batch_norms
 fold_old_batch_norms
 quantize_weights
 quantize_nodes
 strip_unused_nodes
 sort_by_execution_order'
```

这个过程把图中所有的运算操作转换成了具有8-bit量化的等同变量，剩余的以浮点数存在。只有一部分的运算操作支持这种转换，并且在很多平台上，这种量化方式代码运行速度甚至比浮点型运算还慢，不过当所有环境配置正确的时候，这是一种从本质上提升性能的方式。

对于量化方式的优化的完整操作指南不在本文的范畴内，但是在训练过程中类似Conv2D的这种操作之后调用FakeQuantWithMinMaxVars op运算是非常有帮助的。这个运算训练用来控制量化范围的min/max变量，因此在训练或预测过程中不得不用RequantizationRange 来动态计算量化范围。

5、Transform Reference(GTT参数介绍)

--transforms：一些列的转换方式名字，每个方式又可以自己的多个参数（用括号括住的）。参数之间用逗号隔开，如果参数本身含有逗号，那该参数要用双引号括住。

--inputs and outputs：被整个转换过程共享，在调用GTT之前要确保这两个参数设置正确，如果怀疑参数的正确性，可以调用summarize_graph工具来检查输入和输出。

ignore_errors标识：所有的转换都可以传递这个参数，可被设置成true或者false，默认情况下，任何一个错误都会中断整个转换过程，如果设置成true，这些错误只会被记录，转换过程也会跳过去继续。当是一些版本错误或者不重要的问题引发的错误时，这个参数特别有用。

add_default_attributes

参数：无

如果想添加一些属性到新版本的TensorFlow，一般情况下这些属性是默认向后兼容的。这些默认的属性在加载图的时候会被添加进去，当你加载模型是在TensorFlow框架外的时候，这个时候这个更新添加的过程就会起作用了。这个过程会加载所有当前版本TensorFlow含有，而你保存的模型里没有的op属性，并且设置为这些属性为默认值。

backport_concatv2

参数：无

如果有一张被较新版本的TensorFlow框架生成的图文件，该图文件包含ConcatV2运算，而你又想在只支持Concat运算旧版本TensorFlow上运行它，那么这个参数会处理这种等价的转换。

fold_batch_norms

参数：无

先决条件：fold_constants

在训练过程中，当使用了batch normalization后，这个转换是用来优化掉Co

Mul运算相乘，这样在模型预测时就可以省略掉这个Mul遥算。需要确保先运行 fold_constants方式，因为只有当为乘法输入训练产生的复杂的表达式转换成简单的常量时，这种模式才会被GTT发现。

fold_constants

参数：无

先决条件：无

该转换在模型中寻找任何一个子图并评估成常量表达式，然后用常量代替这些子图。当图被加载进来时，这个优化操作就会被执行，因此离线运行并不会减少延迟，但可以简化图让接下来的处理过程更容易。经常结合fold_constants(ignore_errors=true)一起使用来跳过暂态错误，所以这种方式只是一种优化操作。

fold_old_batch_norms

参数：无

先决条件：无

这是老版本的TensorFlow实现batch normalization 所用的一种操作的方式。在新版本中，从Python添加batch normalization 会提供一些列更小的数学操作来替代，在不添加特别代码的情况下达到相同的效果。

未完待续。。。

6、Writing Your Own Transforms

GTT的初衷就是尽可能的让创建自己的图优化、编辑和预处理转换操作变得简单。在其内部，所有定义的图文件中，经过一些修改优化输出一个新的图文件。每一个图定义文件其实是一个包含图中所有列表。你可以在this guide to TensorFlow model files寻找更多的图格式信息，

在tensorflow/tools/graph_transforms/rename_op.cc有个实现 rename_op的小例子，代码如下：

```
Status RenameOp(const GraphDef& input_graph_def,
                const TransformFuncContext& context,
                GraphDef* output_graph_def) {
 if (!context.params.count("old_op_name") ||
     (context.params.at("old_op_name").size() != 1) ||
     !context.params.count("new_op_name") ||
     (context.params.at("new_op_name").size() != 1)) {
   return errors::InvalidArgument(
       "remove_nodes expects exactly one 'old_op_name' and 'new_op_name' "
       "argument, e.g. rename_op(old_op_name=Mul, new_op_name=Multiply)");
 }

 const string old_op_name = context.params.at("old_op_name")[0];
 const string new_op_name = context.params.at("new_op_name")[0];
 output_graph_def->Clear();
 for (const NodeDef& node : input_graph_def.node()) {
   NodeDef* new_node = output_graph_def->mutable_node()->Add();
```

```
  if (node.op() == old_op_name) {
    new_node->set_op(new_op_name);
  }
 }

 return Status::OK();
}
REGISTER_GRAPH_TRANSFORM("rename_op", RenameOp);
```

这个转换的核心在于对输入图文件节点的循环遍历，我们遍历每一个运算操作，向输出添加一个新的op，拷贝原op的内容，然后如果它与参数匹配就更改新op。对于每个转换都有标准的参数集，因此它们都发生在图定义文件和上下文中，并且写进一个新的图文件。代码底部的宏告诉GTT当在转换列表中发现rename_op字符串时调用了哪个函数。

Transform Functions

所有转换函数拥有的标准签名都被定义为TransformFunc，它输入一个图文件，TransformFuncContext包含了环境信息，往输出图文件写，并且返回一个转换是否成功的状态信息。

TransformFuncContext有一个包含图的输入输出和用户进行转换所要传参数变量的列表。

如果你写了一个函数满足了签名并且注册了它，GTT将会负责调用它。

Pattern Syntax(模式语法)

rename_op的例子只需要一次关注一个节点，但大多数情况下需要修改模型中的子图。GTT提供了OpTypePattern语法让这种修改变得简单。这是一种简洁又快速的方法去寻找特定模式的节点。格式如下：

```
OP_TYPE_PATTERN ::= "{" OP "," INPUTS "}"
INPUTS ::= OP_TYPE_PATTERN
```

其中OP支持“*”通配符来匹配任意op类型，op类型直间用“|”符号隔开(例如"Conv2D|MatMul|BiasAdd")。普通的正则表达式模式不支持的，只支持这几个特殊的符号。

你可以把这些模式当成受限制的正则表达式，被用来在图中寻找子树。它们被故意限制成我们所熟知的类型来确保能被尽量直接地创建和调试。

例如，如果想调用一个有一个常量作为第二输入的Conv2D节点，可以用C++初始化列表来设置一个如下的模式：

```
OpTypePattern conv_pattern({"Conv2D", {{"*"}, {"Const"}}});
```

可以很容易地可视化初始化器来展示树结构：

```
OpTypePattern conv_pattern({
 "Conv2D",
 {
  {"*"},
  {"Const"}
 }
});
```

通俗易懂的说，就是一个拥有两个输入的Conv2D操作，第一个输入时任意op类型

```
{"QuantizeV2",
 {
  {"Dequantize"},
  {"Min",
   {
    {"Reshape",
     {
      {"Dequantize"},
      {"Const"},
     }
    },
    {"Const"},
   }
  },
  {"Max",
   {
    {"Reshape",
     {
      {"Dequantize"},
      {"Const"},
     }
    },
    {"Const"},
   }
  },
 }
}
```

这是寻找QuantizeV2节点，它有三个输入，第一个是 Dequantize，第二个是最终从Dequantize拉取的Min输入，第三个输入Max和Min类似。假设我们知道Dequantize操作从8-bit缓冲取值，并且这个子图的最终结果是no-op的，因为刚把8-bit转换成float型，就会被立即转换回8-bit型，因此我们可以找到这种模式并且在不改变输出结果的情况下优化图并删除掉它。

ReplaceMatchingOpTypes

有一种常见的情况，我们希望找到模型中所有某种特性的子图，然后在不改变输入输出连接的情况下替换掉它们。以fuse_convolutions为例，我们需要找到所有的从BilinearResizes读取输入的Conv2D操作，然后用单独的FusedResizeAndPadConv2D操作来替换掉它们，并且不影响其它的操作节点。

为了让上述转换操作变得容易，GTT创建了ReplaceMatchingOpTypes，它输入一个图，用OpTypePattern定义的所要寻找的子图和一个寻找每个出现地方的回调函数。这个回调函数的功能就是寻找包含当前子图信息的匹配节点，返回一个在新的节点清单中的新子图，并用来替换老的子图。实际中如何使用这种转换，可以参考fuse_convolutions：

```
TF_RETURN_IF_ERROR(ReplaceMatchingOpTypes(
  input_graph_def, // clang-format off
  {"Conv2D",
```

```
        {"ResizeBilinear"},
        {"*"}
      }
    },  // clang-format on
    [](const NodeMatch& match, const std::set<string>& input_nodes,
       const std::set<string>& output_nodes,
       std::vector<NodeDef>* new_nodes) {
      // Find all the nodes we expect in the subgraph.
      const NodeDef& conv_node = match.node;
      const NodeDef& resize_node = match.inputs[0].node;
      const NodeDef& weights_node = match.inputs[1].node;

      // We'll be reusing the old weights.
      new_nodes->push_back(weights_node);

      // Create a 'no-op' mirror padding node that has no effect.
      NodeDef pad_dims_node;
      pad_dims_node.set_op("Const");
      pad_dims_node.set_name(conv_node.name() + "_dummy_paddings");
      SetNodeAttr("dtype", DT_INT32, &pad_dims_node);
      SetNodeTensorAttr<int32>("value", {4, 2}, {0, 0, 0, 0, 0, 0, 0, 0},
                               &pad_dims_node);
      new_nodes->push_back(pad_dims_node);

      // Set up the new fused version of the convolution op.
      NodeDef fused_conv;
      fused_conv.set_op("FusedResizeAndPadConv2D");
      fused_conv.set_name(match.node.name());
      AddNodeInput(resize_node.input(0), &fused_conv);
      AddNodeInput(resize_node.input(1), &fused_conv);
      AddNodeInput(pad_dims_node.name(), &fused_conv);
      AddNodeInput(conv_node.input(1), &fused_conv);
      CopyNodeAttr(resize_node, "align_corners", "resize_align_corners",
                   &fused_conv);
      SetNodeAttr("mode", "REFLECT", &fused_conv);
      CopyNodeAttr(conv_node, "T", "T", &fused_conv);
      CopyNodeAttr(conv_node, "padding", "padding", &fused_conv);
      CopyNodeAttr(conv_node, "strides", "strides", &fused_conv);
      new_nodes->push_back(fused_conv);

      return Status::OK();
    },
    {}, &replaced_graph_def));
```

从上可以看到我们定义了要找的模型，在回调函数中用每个老的子图的信息创建新的fused node。当然也拷贝了那些老的权重输入以至于不会丢失。

（1）所有匹配子图中的节点将会在这个函数返回的新子图里被移除。如果有一些节点有用，那这个回调函数就负责把它们再加进来。如果你觉得没有必要把子图里的一些地方修改，可以方便的调用CopyOriginalMatch 来拷贝所有的原始节点。

（2）假设相同的节点在匹配的子图中出现的不会超过一次。这是为了确保子图只会被替换一次，这也就意味着如果有些子图与之前的匹配重叠了，那么它们可能不会被发现。

（3）这个调用框架是保留图的完整性的，它是通过寻找你返回的新节点并且确定在子图之外的输入和输出节点不会被移除。这些重要的节点保存在output_nodes参数内，通过每次替换函数调用传进去。你可以通过设置allow_inconsistencies为true来忽略选项检测，不然的话任何打破图的限制的替换将会被取消。如果你允许inconsistencies，GTT将会在返回最终结果前修复回来。像RenameNodeInputs函数可能会在批量修改节点名时非常有用。

Paramenters

GTT被调用之后所有的参数被解析并放在每个转换的TransformFuncContext参数成员中。对于每个命名的参数，都有一个字符串数组包含了按传递顺序存放的所有值。它们有点像命令行参数，GTT负责解析它们所需要的数据类型，并返回一个错的状态提示引起的错误。下面是一个假设的转换调用：

```
some_transform(foo=a, foo=b, bar=2, bob="1,2,3")
```

下面是string类型的std::map在参数成员里的形式：

```
{{"foo", {"a", "b"}}, {"bar", {"2"}}, {"bob", {"1,2,3"}}}
```

上述语句中的逗号分隔符旁边的双括号很重要，否则将会被认为是分开的变量，从而解析失败。下面是 round_weights如何读取它自己的num_steps参数的例子：

```
TF_RETURN_IF_ERROR(context.GetOneInt32Parameter("num_steps", 256, &num_steps));
```

如果转换失败或者变量出现了不止一次，帮助函数将通过返回状态发起一个有意义的错误提示。如果参数没有被指定将会使用默认值。

Function Libraries

TensorFlow的一个新特性是可以把函数作为图的一部分创建成动态库。有点像定义宏操作替代部分模块模板，封装成库后可以在图内部用不同的输入和输出连接来实例化，类似正规的op操作。但现在GTT只是在输入和输出图之间拷贝这些库，或许不久的将来GTT可以支持更加复杂的操作。

Registering

GTT是用de >REGISTER_GRAPH_TRANSFORM()de>宏来把转换操作名和实现代码联系在一起的。这需要一个字符串和一个函数，然后自动注册转换工具。然而你还是要从以下两点小心操作：

（1）因为每个文件中使用了全局C++对象，链接有时候会跳过REGISTER_GRAPH_TRANSFORM()宏而注册失败。Bazel编译时，需要确保以库的形式链接到了所有新的转换操作，并在cc_binary调用中使用alwayslink选项。

（2）应该通过链接在tensorflow/tools/graph_transforms/BUILD目录下的transform_graph_main_lib library的方式来创建自己的GTT拷贝，它包含了所有去解析命令行参数和调用转换操作的main()函数逻辑。

7、结束语

what a long guide! o(╯□╰)o

参考：https://github.com/tensorflow/tensorflow/blob/master/tensorflow/tools/graph_transforms/README.md