实操教程｜Pytorch转ONNX详解

CV开发者都爱看的 极市平台 2023-02-26 22:00:04 发表于广东 手机阅读

作者｜立交桥跳水冠军@知乎

来源｜https://zhuanlan.zhihu.com/p/272767300

编辑｜极市平台

极市导读

本文作者总结了自己参与Pytorch到ONNX的模型转换转换工作中的经验，主要介绍了该转换工作的意义，模型部署的路径以及Pytorch本身的局限。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

之前几个月参与了OpenMMlab的模型转ONNX的工作（github account: drcut），主要目标是支持OpenMMLab的一些模型从Pytorch到ONNX的转换。这几个月虽然没做出什么成果，但是踩了很多坑，在这里记录下来，希望可以帮助其他人。

这篇是第一部分，理论篇，主要介绍了和代码无关的一些宏观问题。再接下来我会专门写一篇实战篇，针对OpenMMlab中一些具体代码做分析，说明Pytorch转化ONNX过程中的一些代码上的技巧和注意事项。

（1）Pytorch转ONNX的意义

一般来说转ONNX只是一个手段，在之后得到ONNX模型后还需要再将它做转换，比如转换到TensorRT上完成部署，或者有的人多加一步，从ONNX先转换到caffe，再从caffe到tensorRT。原因是Caffe对tensorRT更为***友好***，这里关于***友好***的定义后面会谈。

因此在转ONNX工作开展之前，**首先必须明确目标后端**。ONNX只是一个格式，就和json一样。只要你满足一定的规则，都算是合法的，因此单纯从Pytorch转成一个ONNX文件很简单。但是不同后端设备接受的onnx是不一样的，因此这才是坑的来源。

Pytorch自带的torch.onnx.export转换得到的ONNX，ONNXRuntime需要的ONNX，TensorRT需要的ONNX都是不同的。

这里面举一个最简单的Maxpool的例：

Maxunpool可以被看作Maxpool的逆运算，咱们先来看一个Maxpool的例子，假设有如下一个C*H*W的tensor（shape[2, 3, 3]），其中每个channel的二维矩阵都是一样的，如下所示

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

在这种情况下，如果我们在Pytorch对它调用MaxPool（kernel_size=2, stride=1,pad=0）

那么会得到两个输出，第一个输出是Maxpool之后的值：

$$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

另一个是Maxpool的Idx，即每个输出对应原来的哪个输入，这样做反向传播的时候就可以直接把输出的梯度传给对应的输入：

$$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$$

细心的同学会发现其实Maxpool的Idx还可以有另一种写法：

$$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 13 & 14 \\ 16 & 16 \end{bmatrix},$$

即每个channel的idx放到一起，并不是每个channel单独从0开始。这两种写法都没什么问题，毕竟只要反向传播的时候一致就可以。

但是当我在支持OpenMMEditing的时候，会涉及到Maxunpool，即Maxpool的逆运算：输入MaxpoolId和Maxpool的输出，得到Maxpool的输入。

Pytorch的MaxUnpool实现是接收每个channel都从0开始的Idx格式，而Onnxruntime则相反。因此如果你希望用Onnxruntime跑一样的结果，那么必须对输入的Idx（即和Pytorch一样的输入）做额外的处理才可以。换言之，Pytorch转出来的神经网络图和ONNXRuntime需要的神经网络图是不一样的。

（2）ONNX与Caffe

主流的模型部署有两种路径，以TensorRT为例，一种是Pytorch->ONNX->TensorRT，另一种是Pytorch->Caffe->TensorRT。个人认为目前后者更为成熟，这主要是ONNX，Caffe和TensorRT的性质共同决定的

	ONNX	Caffe
灵活性	高	低
op粒度	细粒度	粗粒度
条件分支	不支持	支持
动态shape	支持	不支持

上面的表列了ONNX和Caffe的几点区别，其中最重要的区别就是op的粒度。举个例子，如果对Bert的Attention层做转换，ONNX会把它变成MatMul，Scale，SoftMax的组合，而Caffe可能会直接生成一个叫做Multi-Head Attention的层，同时告诉CUDA工程师：“你去给我写一个大kernel“（很怀疑发展到最后会不会把ResNet50都变成一个层。。。）

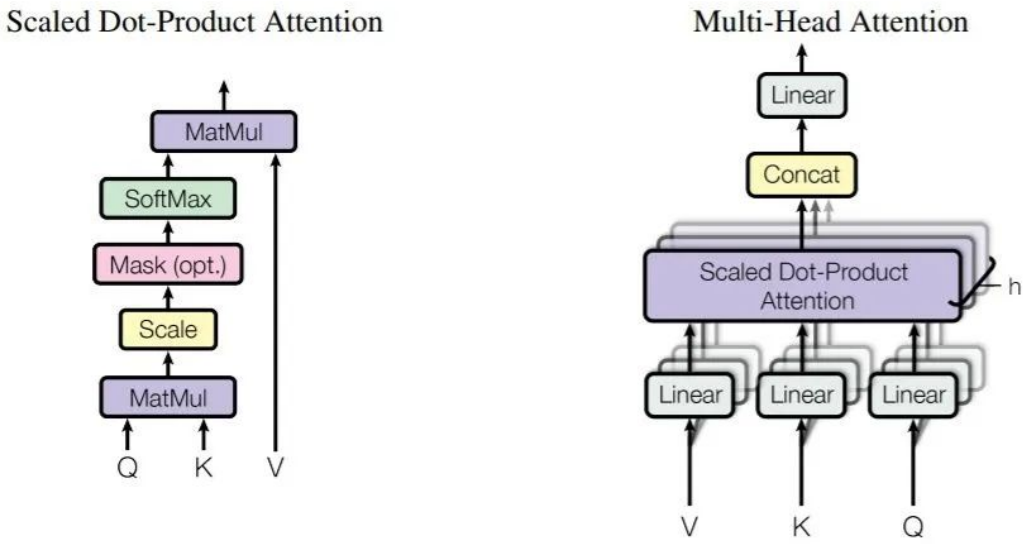


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

因此如果某天一个研究员提了一个新的State-of-the-art的op，很可能它直接就可以被转换成ONNX（如果这个op在Pytorch的实现全都是用Aten的库拼接的），但是对于Caffe的工程师，需要重新写一个kernel。

细粒度op的好处就是非常灵活，坏处就是速度会比较慢。这几年有很多工作都是在做op fushion（比如把卷积和它后面的relu合到一起算），XLA和TVM都有很多工作投入到了op fushion，也就是把小op拼成大op。

TensorRT是NVIDIA推出的部署框架，自然性能是首要考量的，因此他们的layer粒度都很粗。在这种情况下把Caffe转换过去有天然的优势。

除此之外粗粒度也可以解决分支的问题。TensorRT眼里的神经网络就是一个单纯的DAG：给定固定shape的输入，执行相同的运算，得到固定shape的输出。

**目前TensorRT的一个发展方向是支持dynamic shape，但是还很不成熟。

```
tensor i = funcA();
if(i==0)
  j = funcB(i);
else
  j = funcC(i);
funcD(j);
```

对于上面的网络，假设funcA,funcB,funcC和funcD都是onnx支持的细粒度算子，那么ONNX就会面临一个困难，它转换得到的DAG要么长这样：funcA->funcB->funcD，要么funcA->funcC->funcD。但是无论哪种肯定都是有问题的。

而Caffe可以用粗粒度绕开这个问题

```
tensor i = funcA();
coarse_func(tensor i) {
  if(i==0) return funcB(i);
  else return funcC(i);
```

```
}
funcD(coarse_func(i))
```

因此它得到的DAG是：funcA->coarse_func->funcD

当然，Caffe的代价就是苦逼的HPC工程师就要手写一个coarse_func kernel。。。（希望Deep Learning Compiler可以早日解放HPC工程师）

(3)Pytorch本身的局限

熟悉深度学习框架的同学都知道，Pytorch之所以可以在tensorflow已经占据主流的情况下横空出世，成功抢占半壁江山，主要的原因是它很灵活。举个不恰当的例子，tensorflow就像是C++，而Pytorch就是Python。

tensorflow会把整个神经网络在运行前做一次编译，生成一个DAG（有向无环图），然后再去跑这张图。Pytorch则相反，属于走一步看一步，直到运行到这个节点算出结果，才知道下一个节点该算啥。

ONNX其实就是把上层深度学习框架中的网络模型转换成一张图，因为tensorflow本身就有一张图，因此只需要直接把这张图拿到手，修修补补就可以。

但是对于Pytorch，没有任何图的概念，因此如果想完成Pytorch到ONNX的转换，就需要让ONNX再旁边拿个小本子，然后跑一遍Pytorch，跑到什么就把什么记下来，把记录的结果抽象成一张图。因此Pytorch转ONNX有两个天然的局限。

1. 转换的结果只对特定的输入。如果换一个输入导致网络结构发生了变化，ONNX是无法察觉的（最常见的情况是如果网络中有if语句，这次的输入走了if的话，ONNX就只会生成if对应的图，把else里面全部的信息都丢掉）。

2. 需要比较多的计算量，因为需要真刀真枪的跑一遍神经网络。

PS：针对于以上的两个局限，我的本科毕设论文提出了一种解决方案，就是通过编译器里面的词法分析，语法分析直接扫描Pytorch或者tensorflow的源代码得到图结构，这样可以轻量级的完成模型到ONNX的转换，同时也可以得到分支判断等信息，这里放一个github链接(https://github.com/drcut/NN_transform)，希望大家多多支持

*目前Pytorch官方希望通过用TorchScript的方式解决分支语句的问题，但据我所知还不是很成熟。

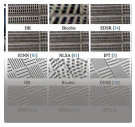
收获更多技术干货

阅读原文

喜欢此内容的人还喜欢

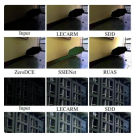
ICCV 2023｜南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）

极市平台



ICCV23｜将隐式神经表征用于低光增强，北大张健团队提出NeRCo

极市平台



YOLOv5帮助母猪产仔？南京农业大学研发母猪产仔检测模型并部署到Jetson Nano开发板

极市平台