TRT Project: Vision Transformer(ViT)

一、背景介绍

深蓝学院《CUDA入门与深度神经网络加速》 课程，TensorRT 部分的作业。

内容是使用TensorRT Python API 手动搭建图像识别Vision Transformer(ViT)模型。学员不需要从零开始搭建，这里已经搭好了一个模型结构框架，进行相应的填充即可。

ViT 模型知识给大家推荐一些博客文章，请大家自行学习：

Vision Transformer 超详细解读系列文章 https://zhuanlan.zhihu.com/p/340149804

详解 Vision Transformer (ViT) https://blog.csdn.net/qq_39478403/article/details/118704747

注意： 整个作业只支持batch=1

二、模型数据介绍和下载

2.1 模型下载

github上有多个ViT项目，这里选择了一个最适合学习的项目（已经下载了，见压缩包**ViT-pytorch-main.zip**）。

```
https://github.com/jeonsworld/ViT-pytorch
model_type: ViT-B_16
dataset: cifar10-100_500
input: [batch, 3, 224, 224]
output: [batch, 10]
```

但这个项目有个比较大的缺点，只提供了pre-train和 fine-tuned 的npy格式的模型，识别率为0，无法直接使用。有两个改进方案：

方案1，按照该项目readme中介绍的使用方法，训练出来一个模型，并转成onnx模型。

方案2，课程提供了一个只训练了100 step的模型，一个checkpoint模型和一个转好的onnx模型。文件下载地址：https://pan.baidu.com/s/1StW6Z4yy8uTklR52X9K8Cg?pwd=m3w7，提取码：m3w7。

2.2 数据下载

使用 CIFAR10-100_500 数据集。

1. github上的项目，训练时为了提高读取文件的效率，采用的是将多张图片合并成一个文件的形式。https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
2. 咱们作业使用的更直观的 jpg 格式的数据，共十类，每类1000张图片。https://github.com/YoongiKim/CIFAR-10-images

2.3 验证base准确性

1. 使用 onnx 模型和 jpg 格式的数据 验证模型准确性。
2. 方案2提供的模型，识别准确率为89.72% （很低，因为只训练了100 step）
3. 验证命令

```
python valid.py -x ViT-B_16.onnx -d CIFAR-10-images-master/test/
```

2.4 测速

1. 测试 onnx cpu 速度 （选做）
2. 测试 onnx gpu 速度 （选做）
3. 测试 使用trt onnxparser 直接转出来的trt模型速度
4. 测试 手动转换出来的 不同精度 trt 模型的速度

trt 模型测速方式，建议使用trtexec，选择 Host Latency or GPU Compute 的 mean 时间即可：

```
./trtexec --loadEngine=model.plan --shapes=input:1x1x224x224 --
plugins=LayerNorm.so
```

三、目录文件介绍

```
ViT-pytorch # https://github.com/jeonsworld/ViT-pytorch
model2onnx.py # 将 npy 模型 转成 onnx 模型的代码，注意重要的参数
do_constant_folding=False

# layernorm  相关
LayerNormPlugin
test_nn_layer_norm.py
test_nn_layer_norm.sh

# 验证onnx和trt模型准确率的脚本
valid.sh
valid.py

# onnx 模型转成 trt 格式的相关
# builder.py 中的结构和 ViT-pytorch/models/modeling.py 差不多。
builder.sh
builder.py
trt_helper.py
calibrator.py
ViT-B_16.onnx

# 这个脚本很重要，支持load checkpoint 格式模型，并跑一张图片
# 当转出来的 trt 模型结果对不上时，可以使用这个脚本进行调试。
# 使用方法自己琢磨吧。
test.py
```

四、作业内容

0. 使用trt onnx-parser 将 onnx 模型转成 trt 格式，并测速。

```
# 使用onnx-parser将onnx模型转成trt plan 格式
#  验证
python valid.py -p model.plan -d CIFAR-10-images-master/test/
```

1. 学习 使用trt python api 搭建网络

填充trt_helper.py 中的空白函数，包括Linear，LayerNorm，addSoftmax等。学习使用 api 搭建网络的过程。

2. 编写layernorm plugin

trt不支持layer_norm算子，编写layer_norm plugin，并将算子添加到网络中，进行验证。

1. 及格：将 “基础款LayerNormPlugin.zip”中实现的基础版 layer_norm算子 插入到 trt_helper.py addLayerNorm函数中。
2. 优秀：将整个layer_norm算子实现到一个kernel中，并插入到 trt_helper.py addLayerNorm函数中。可以使用testLayerNormPlugin.py对合并后的plugin进行单元测试验证。
3. 进阶：在2的基础上进一步优化，线索见 https://www.bilibili.com/video/BV1i3411G7vN?spm_id_from=333.999.0.0

3. 观察GELU算子的优化过程

GELU算子使用一堆基础算子堆叠实现的（详细见trt_helper.py addGELU函数），直观上感觉很分散，计算量比较大。
但在实际build过程中，这些算子会被合并成一个算子。build 过程中需要设置log为 trt.Logger.VERBOSE，观察build过程。

4. 学习 builder.py build_embeddings_layer函数逻辑，体会 pytorch api 和 trt api 的差异、

4. 进行 fp16 加速，观察模型大小，准确率和速度

需要注意plugin 是否支持 fp16？是否设置了fp16?

1. 及格：设置build_config，对模型进行fp16优化。
2. 优秀：编写fp16 版本的layer_norm算子，使模型最后运行fp16版本的layer_norm算子。

5. 进行 int8 加速，观察模型大小，准确率和速度

完善calibrator.py内的todo函数

1. int8 出来的模型，准确率可能会掉很多，为什么
2. int8 和 fp16 可以都 enable，观察 模型大小，准确率和速度

(选做) 6. 支持batch_size > 1，可以按照以下提示进行修改

```
1. 修改 builder.py 中的 profile 和valid.py
2. builder.py build_embeddings_layer 函数中，cls_token 的batch 维度只有1
3. builder.py build_vision_transformer_layer函数中，Slice batch维度
4. calibrator.py 中，需要进行 batch 拼接。
```