BERT Inference Using TensorRT Python API

深蓝学院《CUDA入门与深度神经网络加速》 课程第6、7两章的作业（TensorRT plugin的用法，TensorRT 量化加速）。作业内容是使用TensorRT Python API 手动搭建 BERT 模型。同学们不需要从零开始搭建，课程组已经搭好了一个模型结构框架，同学们进行相应的填充即可。

模型架构框架：https://github.com/shenlan2017/TensorRT

一. 文件信息

1. model2onnx.py 使用pytorch 运行 bert 模型，生成demo 输入输出和onnx模型
2. onnx2trt.py 将onnx，使用onnx-parser转成trt模型，并infer
3. builder.py 输入onnx模型，并进行转换
4. trt_helper.py 对trt的api进行封装，方便调用
5. calibrator.py int8 calibrator 代码
6. 基础款LayerNormPlugin.zip 用于学习的layer_norm_plugin

二. 模型信息

2.1 介绍

1. 标准BERT 模型，12 层, hidden_size = 768
2. 不考虑tokensizer部分，输入是ids，输出是score
3. 为了更好的理解，降低作业难度，将mask逻辑去除，只支持batch=1 的输入

BERT模型可以实现多种NLP任务，作业选用了fill-mask任务的模型

```
输入：
The capital of France, [mask], contains the Eiffel Tower.

topk10输出：
The capital of France, paris, contains the Eiffel Tower.
The capital of France, lyon, contains the Eiffel Tower.
The capital of France,, contains the Eiffel Tower.
The capital of France, tolilleulouse, contains the Eiffel Tower.
The capital of France, marseille, contains the Eiffel Tower.
The capital of France, orleans, contains the Eiffel Tower.
The capital of France, strasbourg, contains the Eiffel Tower.
The capital of France, nice, contains the Eiffel Tower.
The capital of France, cannes, contains the Eiffel Tower.
The capital of France, versailles, contains the Eiffel Tower.
```

2.2 输入输出信息

输入

1. input_ids[1, -1]： int 类型，input ids，从BertTokenizer获得
2. token_type_ids[1, -1]：int 类型，全0
3. position_ids[1, -1]：int 类型，[0, 1, ..., len(input_ids) - 1]

输出

1. logit[1, -1, 768]

三. 作业内容

第6章节作业

3.1 学习使用 trt python api 搭建网络

填充trt_helper.py 中的空白函数（addLinear， addSoftmax等）。学习使用api 搭建网络的过程。

3.2 编写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

3.3 观察GELU算子的优化过程

1. 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。
2. 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。
3. 体会trt在转换过程中的加速优化操作。

第7章节作业

3.4 进行 fp16 加速并测试速度

及格标准：设置build_config，对模型进行fp16优化；
优秀标准：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。

3.5 进行 int8 加速并测试速度

1. 完善calibrator.py内的todo函数，使用calibrator_data.txt 校准集，对模型进行int8量化加速。

四. 深度思考

4.1 还有那些算子能合并?

1. emb_layernorm 模块，3个embedding和一个layer_norm，是否可以合并到一个kernel中?
2. self_attention_layer 中，softmax和scale操作，是否可以合并到一个kernel中?
3. self_attention_layer，要对qkv进行三次矩阵乘和3次转置。三个矩阵乘是否可以合并到一起，相应三个转置是否可以？如果合并了，那么后面的q*k*和*attn*v，该怎么计算?
4. self_output_layer中，add 和 layer_norm 层是否可以合并?

以上问题的答案，见 https://github.com/NVIDIA/TensorRT/tree/release/6.0/demo/BERT

4.2 除了上面那些，还能做那些优化？

1. 增加mask逻辑，多batch一起inference，padding的部分会冗余计算。比如一句话10个字，第二句100个字，那么padding后的大小是[2, 100]，会有90个字的冗余计算。这部分该怎么优化？除了模型内部优化，是否还可以在外部拼帧逻辑进行优化？
2. self_attention_layer层，合并到一起后的QkvToContext算子，是否可以支持fp16计算？int8呢？

以上问题答案，见 https://github.com/NVIDIA/TensorRT/tree/release/8.2/demo/BERT