# bert-frontend study

**23-1**

修改docker file，用opu-compiler的代替，原先的报错，不是很好用。

[x]目标1：代码运行

Q1 代码不能zip本地，只能通过git ssh的方式，否则第三方模组关联将丢失。

ope-compiler的docker和vit-bert的docker都没问题，均需要换源。

运行和编译成功：vit branch doc docker一处pip版本报错需要在sh文件中修改。

执行 python3 driver.py报错，原因为：huggingFace的load\_datasets无法连接HTTP。本地方案未解决，玄学解决方案：关闭宿主机的V2ray代理，wsl内即可连接。

**配置调试环境**

调试环境：WSL2+vscode+Docker，Python+C++联合调试

改动的地方：

1. ubuntu1804\_install\_python.sh，添加

|  |
| --- |
| apt-get update  apt-get install -y python3 python3.7-dev  rm -rf /usr/bin/python3  ln -s /usr/bin/python3.7 /usr/bin/python3 |

并重新build docker环境，将python版本更新至3.7以上（联合调试的前提）。

（可选）DockerFile中对apt-get和pip进行换源：

|  |
| --- |
| # Using douban pipy mirror  RUN pip3 install -i https://pypi.douban.com/simple/ -U pip  RUN pip3 config set global.index-url https://pypi.douban.com/simple/ |

1. vscode中安装Python C++ Debugger插件。
2. 添加调试配置，内容如下：

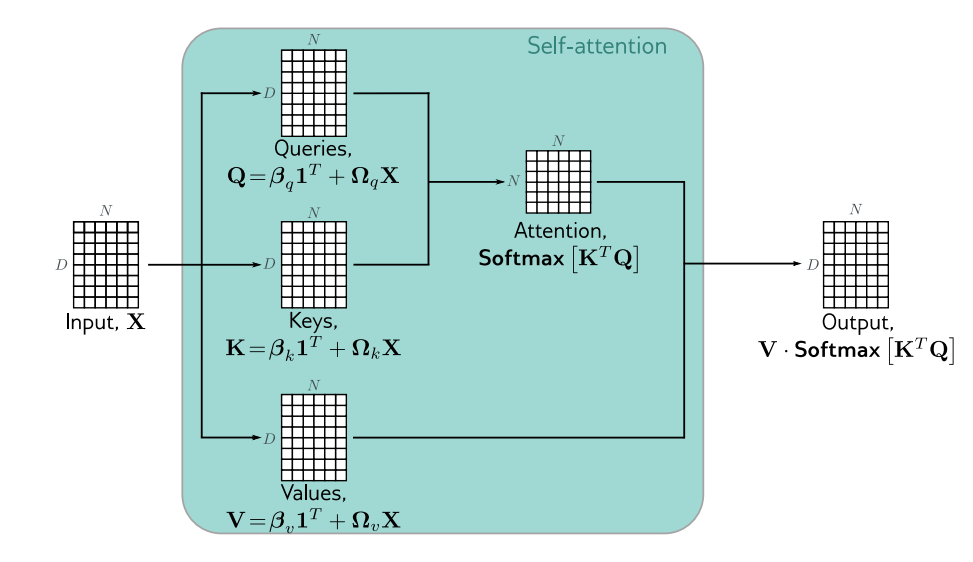
|  |
| --- |
| {  "version": "0.2.0",  "configurations": [  {  "name": "(gdb) Attach",  "type": "cppdbg",  "request": "attach",  "program": "/usr/bin/python3",  "MIMode": "gdb",  "miDebuggerPath": "/usr/bin/gdb",  "processId":"",  "setupCommands": [  {  "description": "为 gdb 启用整齐打印",  "text": "-enable-pretty-printing",  "ignoreFailures": true  }  ]  },  {  "name": "Python C++ Debugger",  "type": "pythoncpp",  "request": "launch",  "pythonLaunchName": "Python: Current File",  "cppAttachName": "(gdb) Attach"  },  {  "name": "Python: Current File",  "type": "python",  "request": "launch",  "program": "${file}",  "console": "integratedTerminal",  "justMyCode": true,  "cwd": "${fileDirname}",  }  ]  } |

调试时选择Python C++ Debugger，在调试时调用堆栈显示Python: Current File和(gdb)Attach即成功。

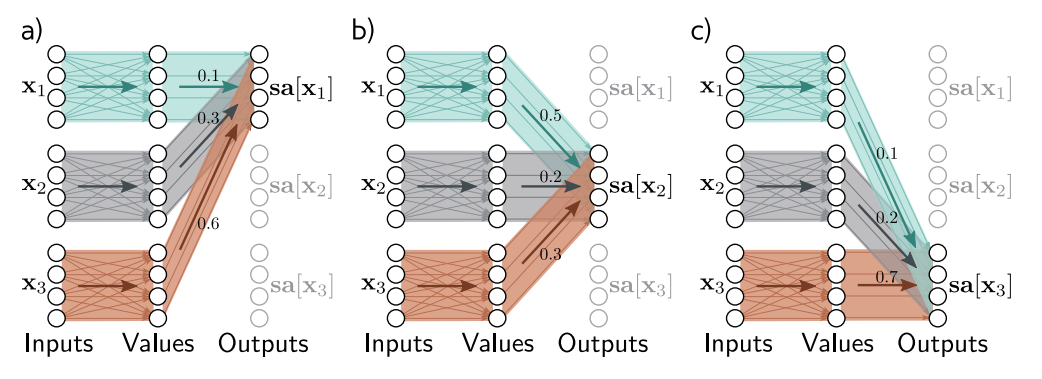
[ ]目标2：学习Bert，掌握BertModel的训练流程，最终目标：读懂进入tvm的BertModel是什么，以及读懂driver.py。

[x]目标3：学习transformers算法

transformers中的self-attention块如下图所示：

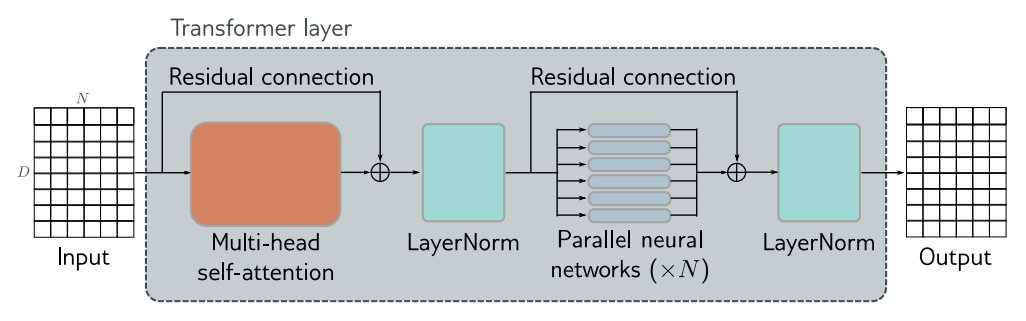


这张图很好的反映了transformers主要的核心计算思想。其中XQKV都已是矩阵，所有的计算均被扩展成矩阵运算；Q，K分别为query和key矩阵，作用是通过计算两者的关联度来作为attention使用，self即体现在QK矩阵均是关于input X自身的线性组合。剔除attention内部展示后的计算过程如下：

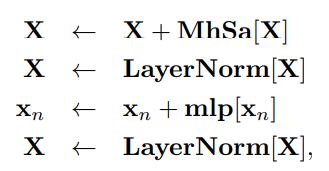


这是每一个具体结点的计算过程，input X经过线性组合扩展成V后通过注意力机制得到output sa。

transformers结构还有其他部分和扩展，例如position encoding，Multi-head机制，attention标准化等，最终形成的transformers layer如下：



其代码表示为：



其中mlp即多层感知机，即图示的Parallel neural networks，全连接网络。

后续要补充的有Tokenization和embeddings，tokenization的byte pair encoding Algorithm：

（1）统计给定文本的单字符出现次数；

（2）迭代处理次数最高的两个单字符，将其合并成新字符，并将其个数从单字符中剔除；

（3）循环过程直到处理完所有字符或字符个数达到预先设定的个数。

而embedding过程：one-hot vector相当于从词汇表中将给定文本中的这些词提取出来组成新的inputX的过程。

[ ]目标4：理解下述fuse Pass

Custom-Pass as below:

|  |  |
| --- | --- |
| **Pass\_name** | **Content** |
| fuse\_op.h |  |
| fuse\_embedding.cc |  |
| fuse\_gelu.cc |  |
| fuse\_ops\_custom.cc |  |
| fuse\_attention\_mask.cc |  |
| gen\_ir.cc |  |
| gen\_ir.h |  |
| hw\_info.h |  |
| quantize\_util.h |  |
| peephole.cc |  |
| replace\_attention\_mask.cc |  |
| relay/backend/build\_module.cc | [+] FuseEmbedding()(relay\_module);  [+] FuseGelu()(relay\_module);  [+] ReplaceAttentionMask()(relay\_module);  [+] FuseOpsCustom()(relay\_module);  [+] GenIR()(relay\_module); |