reference:

主体结构连接：https://github.com/DA-southampton/Read_Bert_Code/tree/7e3c619aef9e462ec8865cde75c7a0ff2aefe60f

内容补充：Bert代码详解（一、二）

https://blog.csdn.net/cpluss/article/details/88418176，

https://blog.csdn.net/cpluss/article/details/88418353

代码采用：30分钟带你彻底掌握Bert源码(Pytorch)，超详细！

https://zhuanlan.zhihu.com/p/148062852

https://blog.csdn.net/cpluss/article/details/88418353

代码目录说明

```
├── chineseGLUEdatasets #  存放数据
|  └── inews
|  └── lcqmc
├── metrics                 # metric计算
|  └── glue_compute_metrics.py
├── outputs # 模型输出保存
|  └── inews_output
|  └── lcqmc_output
├── prev_trained_model  # 预训练模型
|  └── albert_base
|  └── bert-wwm
├── processors          # 数据处理
|  └── glue.py
|  └── utils.py
├── tools               #  通用脚本
|  └── progressbar.py
|  └── commom.py
├── transformers      # 模型
|  └── modeling_albert.py
|  └── modeling_bert.py # 预训练模型结果
├── convert_albert_original_tf_checkpoint_to_pytorch.py  #  模型文件转换
├── run_classifier.py # 主程序，微调阶段分类模型
├── run_classifier_inews.sh #  任务运行脚本
```

代码解构

batch_size 16,suqence_length 32,word_embedding_d/hidden_size/all_head_size 768，attention_head_size 64，num_attention_heads 12,num_hidden_layer 12, intermediate_size 3072

BertForSequenceClassification

forward：运行BertModel，得到输出[last_hidden_state, pooler_output, hidden_states, attentions]，对pooled_output[16,768] 进行dropout并经过classifier(768->num_labels) 得logits[16,nlabels]，若num_labels=1做logits和label的MSELoss回归，否则求交叉熵损失函数，返回[loss, logits, hidden_states, attentions]

BertModel

1BertEmbeddings: attention_mask：由**word,token_type,position**生成embeddings

refer-nn.Embedding:The input to the module is a list of indices, and the output is the corresponding word embeddings

forward：接收input_ids,token_type_ids,position_ids [batch_size,seq_length]，分别经过nn.Embedding类之后相加经过layerNorm，dropout

2BertEncoder：建立了整个transformer架构，encoder操作，返回 BertModel 类作为 encoder_outputs，维度大小 torch.Size([16, 32, 768]) 768：隐状态数

forward:对每一一个layer计算attentions hidden_states,可添加输出中间过程，输入hidden_states(**embedding_output**) [16,32,768],attention_mask[16,1,1,32]，输出hidden_states as encoder_outputs

BertLayer(最重要的地方

BertAttention：计算z=att(Q,K,V)

BertSelfAttention(key 返回[16, 32, 768]维度的Z值，传入BertSelfOutput

forward:输入 hidden_states[16,32,768],attention_mask[16,1,1,32],通过线性变换声明可学习的[16,32,768]的QKV,接着转化为多头[16,12 num_head,32,64]，QK相乘得

attentino_scores[16,head12,32,32](32*32即为自注意力)并归一化，之后加上attention_mask[16,1,1,32]并经过softmax、dropout，乘head_mask????[layers12,heads12,1,1],最后乘上V得attention_probs[16,heads12,32,64]，[reshape值得借鉴]转化为out：context_layer[16,32,768], attention_probs(可选)

？？nn.Linear(hidden_states,all_heads_size)如何生成[16,32,768]数据：在最后一维，将768映射为768？？head_mask?

BertSelfOutput 三步linear、dropout、归一化，得输出hidden_states(维度不变

BertIntermediate 经过一个Linear（3072维feed-forward、一个Gelu激活函数，得到hidden_states [16, 32, 3072]

BertOutput Liner[-768]+BertLayerNorm+Dropout,[16, 32, 768]，返回hidden_states给BertEncoder类

3BertPooler：输入hidden_states[16,32,768]，通过取和第一个token相关的隐状态（即将每个句子的第一个单词的表示作为整个句子的表示）来池化模型[16,768]，经过同形Linear，与Tanh激活函数得到pooled_output[16,768] 做分类等下游任务

forward函数：

step1：将attention_mask拓展并将0换为较大负值[batch_size, 1, 1, seq_length]；整理head_mask类似为[num_hidden_layers, num_heads,1,1]；构造**position_ids，和input_ids、token_type_ids**[batch_size,seq_length]输入到**embeddings/BertEmbeddings**层(为什么传入同一个nn.embedding?：信号处理)并相加、layernorm、dropout

step2：进入transformer层）从embeddings层得到输出，然后送进encoder层，encoder层由12层BertLayer(config)构成，得到最后的输出encoder_layers，

输出：[last_hidden_state, pooler_output, hidden_states, attentions]

以上为整个encoding过程

refer else：

BertLayerNorm:torch.nn.LayerNorm，在小批量输入中做层归一化

CrossEntropyloss:交叉熵将Softmax(得分类概率)–Log(映射0以下)–NLLLoss合并，NLLLoss将样本ln后的值取出label对应位置的值取负并求平均

(https://blog.csdn.net/qq_22210253/article/details/85229988

BertConfig

保存BERT的各种参数配置

BertOnlyMLMHead

使用mask 方法训练语言模型时用的，返回预测值

过程：调用BertLMPredictionHead，返回的就是prediction_scores

BertLMPredictionHead

decode功能

过程：调用BertPredictionHeadTransform -> linear层，输出维度是vocab_size

BertPredictionHeadTransform

过程：dense -> 激活(gelu or relu or swish) -> LayerNorm

BertOnlyNSPHead

NSP策略训练模型用的，返回0或1

过程：添加linear层输出size是2，返回seq_relationship_score

BertPreTrainingHeads

MLM和NSP策略都写在里面，输入的是Encoder的输出sequence_output, pooled_output

返回（prediction_scores, seq_relationship_score）分别是MLM和NSP下的分值

BertPreTrainedModel

从全局变量BERT_PRETRAINED_MODEL_ARCHIVE_MAP加载BERT模型的权重

继承关系：nn.Module -> PreTrainedModel -> BertPreTrainedModel -> BertModel以及后面所有BertPreTrainedModel

BertForPreTraining

计算score和loss

通过BertPreTrainingHeads，得到prediction后计算loss，然后反向传播。

BertForMaskedLM

只有MLM策略的loss

BertForNextSentencePrediction

只有NSP策略的loss

BertForSequenceClassification

计算句子分类任务的loss

BertForMultipleChoice

计算句子选择任务的loss

BertForTokenClassification

计算对token分类or标注任务的loss

BertForQuestionAnswering

计算问答任务的loss

全局变量

BERT_START_DOCSTRING

BERT_INPUTS_DOCSTRING

BERT_PRETRAINED_CONFIG_ARCHIVE_MAP

BERT_PRETRAINED_MODEL_ARCHIVE_MAP

装饰器

？？？？？？？？？？？

BertModel的返回为 outputs = (sequence_output, pooled_output,) + encoder_outputs[1:]

sequence_output：torch.Size([16, 32, 768])

pooled_output：torch.Size([16, 768]) 是cls的输出经过一个pool层（其实就是linear维度不变+tanh）输出

outputs返回给BertForSequenceClassification，也就是对pooled_output 做分类

BertIntermediate

函数查阅

x.permute(0, 2, 1, 3) # 变换成特定维度

batchnorm和layernorm：batchnorm用来加速网络训练的Reducing Internal Covariate Shift（减小内部协变量偏移），在sigmoid激活函数中为避免数值往两边偏移导致导数变小梯度消失；LayerNorm在特征维度进行归一化，不依赖于batchsize。两者均有可学习参数