基于BERT 的语义匹配 项目

一、项目简介

本项目采用自己复现的BERT模型，来对语义相似度进行判别。例如判断“看图猜一电影名”和“看图猜电影”语义是否相同。

项目结合《自然语言处理》课程中所讲的Bert模型框架、B站李宏毅《深度学习》课程等所学知识，参考谷歌开源代码编写完成。如有不到位的地方，欢迎指正。

关于老师提出的思考题，可以在这里查看。

项目代码已上传到github，地址为：1349064125/nlp_homework (github.com)

二：语义匹配

语义匹配是NLP的基础任务之一，通常表现为判断两句话是否表达了相同或相似。此类任务目的在抓取文本的含义，故研究的结果很容易移植到NLP任务当中，例如搜索、QA、推荐任务等。是一个很重要的研究方向。具体如下所示：

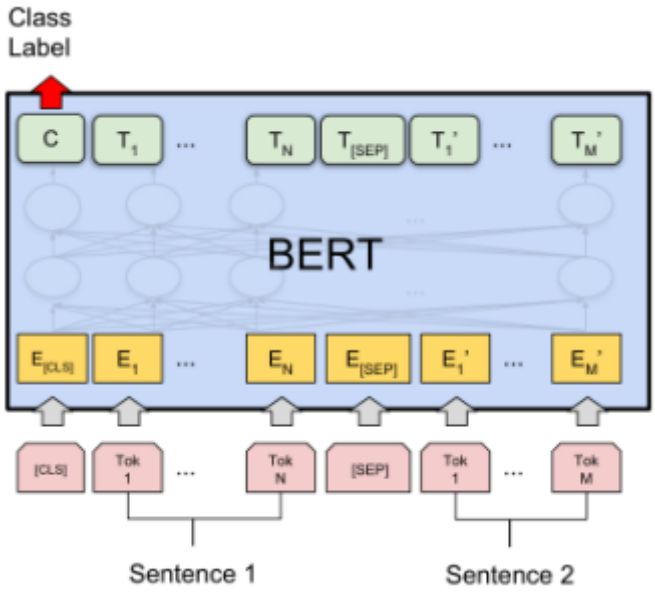
类型	文本1	文本2
相似文本	看图猜一电影名	看图猜电影
不相似文本	无线路由器怎么无线上网	无线上网卡和无线路由器怎么用

采用自注意力等机制的BERT 模型，在NLP领域取得巨大成功，我们也Bert解析文本的语义。

三：模型结构

1.BERT总体结构

BERT 使用 Transformer，这是一种注意力机制，可以学习文本中单词（或sub-word）之间的上下文关系。Transformer 包括两个独立的机制——一个读取文本输入的Encoder和一个为任务生成预测的Decoder。由于 BERT 的目标是生成语言模型，因此只需要Encoder。本项目使用如下的BERT结构：



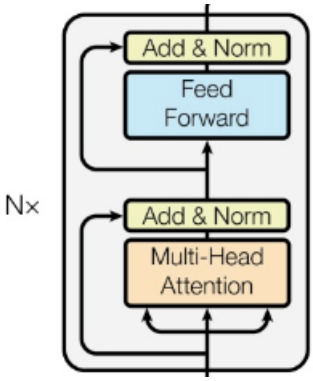
代码如下所示：

```
class BERT(nn.Module):
    def __init__(self):
        super(BERT, self).__init__()
        self.src_emb = nn.Embedding(config.Vocab_Size, config.EmbedSize)
        self.pos_emb = PositionalEncoding(config.EmbedSize)
        self.layers = nn.ModuleList([Layer() for _ in range(config.N_Layers)])
        self.fc = nn.Linear(config.EmbedSize,2)
    def forward(self, enc_inputs,mask):
        # 1.词嵌入
        enc_outputs = self.src_emb(enc_inputs) # [batch_size, src_len, d_model]
        # 2.增加位置编码
        enc_outputs = self.pos_emb(enc_outputs.transpose(0, 1)).transpose(0, 1)
        enc_self_attn_mask = mask
        # 3.多层Layer： 自注意力+前馈网络（多头合成一头）
        for layer in self.layers:
            enc_outputs = layer(enc_outputs, enc_self_attn_mask)
        # 4.返回二分类结果
        return self.fc(enc_outputs)
```

模型总体分为四步：（1）将输入的句子按字进行词嵌入，（2）添加位置编码，（3）多层注意力及前馈网络，（4）二分类返回结果。

2.多层Layer

BERT一般由24层自注意力+前馈网络组成，不同层有不同的作用，例如有的层用于编码词的表示，有的层用于提取句子语义等。



代码如下所示：`

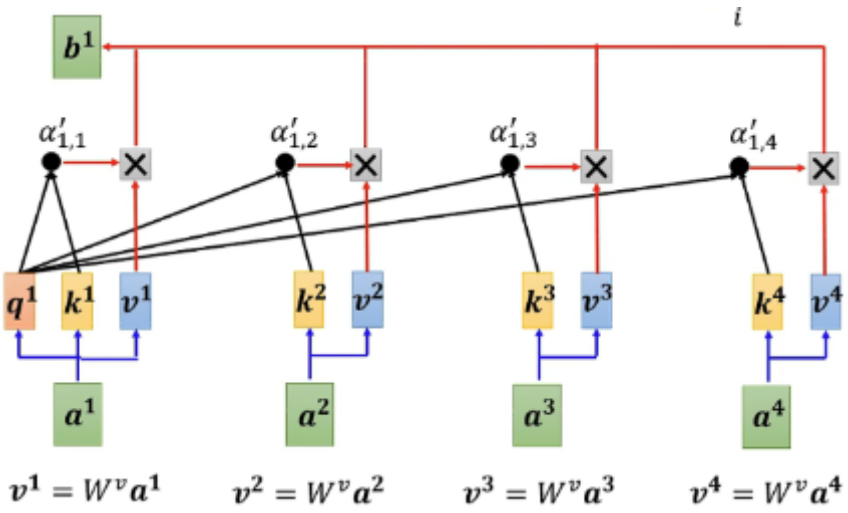
```
class Layer(nn.Module):
    def __init__(self):
        super(Layer, self).__init__()
        self.enc_self_attn = MutiHeadAttation(config.EmbedSize,
                                              config.N_Heads,config.Dropout)
        self.pos_ffn = PoswiseFeedForwardNet()

    def forward(self, enc_inputs, enc_self_attn_mask):
        enc_outputs = self.enc_self_attn(enc_inputs, enc_inputs,
                                              enc_inputs,enc_self_attn_mask)
        enc_outputs = self.pos_ffn(enc_outputs)
        return enc_outputs
```

多层Layer分为2步：（1）多头自注意力机制，（2）前馈网络多头合成一头

3.多头自注意力机制

相较于传统的RNN/LSTM模型，BERT模型采用了自注意力机制，更擅长捕捉长距离依赖。自注意力机制也可以说是BERT的灵魂。自注意力机制计算了每个token和其它token的关系，然后和更新自己得出的V，相比于循环神经网络，能更好的提取文本信息。而多头注意力实际上就是多个自注意力机制的堆叠，不过因为多层的缘故，最后所有的自注意力机制会生成多个大小相同的矩阵，处理方式是把这些矩阵拼接起来，然后通过乘上一个参数矩阵得到最后的计算结果。自注意力机制如下图所示：



代码如下：

```
class MutiHeadAttation(nn.Module):
    def __init__(self,hid_dim,n_heads,dropout):
        super().__init__()

        assert hid_dim % n_heads == 0

        self.hid_dim = hid_dim    ##词向量维度
        self.n_heads =n_heads     #头数
        self.dropout = dropout

        self.Q = nn.Linear(hid_dim,n_heads*hid_dim)
        self.K = nn.Linear(hid_dim,n_heads*hid_dim)
        self.V = nn.Linear(hid_dim,n_heads*hid_dim)

        self.sfmx = nn.Softmax(dim=-1)
        self.fc = nn.Linear(n_heads*hid_dim,hid_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self,query,key,value,mask):
        residual, batch_size = query, query.size(0)

        # 1.生产 Q K V矩阵。
        qlist =
self.Q(query).view(batch_size,-1,self.n_heads,self.hid_dim).transpose(1,2)
        klist =
self.K(key).view(batch_size,-1,self.n_heads,self.hid_dim).transpose(1,2)
        vlist =
self.V(value).view(batch_size,-1,self.n_heads,self.hid_dim).transpose(1,2)
```

```
        # 2.计算每两个Token的关系
        scores= torch.matmul(qlist,klist.transpose(-1, -2)) /
np.sqrt(self.hid_dim)

        # 3.将补充的文本[PAD]和其他token的关系变成0
        mask = mask.unsqueeze(1).repeat(1, config.N_Heads, 1, 1)
        scores.masked_fill_(mask, -1e9)

        # 4.计算每个token注意力后的编码
        A = self.sfmx(scores)

        # 5.多头变一头
        list = torch.matmul(A,vlist).transpose(1, 2).reshape(batch_size, -1,
self.n_heads * self.hid_dim)
        output = self.fc(list)
        #残差网络返回这一层答案。
        return nn.LayerNorm(config.EmbedSize).to(config.DEVICE)(output+residual)
```

4.位置编码

不同于循环神经网络因为其编码方式使其天然就考虑了位置信息，由于BERT模型中的自注意力网络同时考虑了整个句子，所以该模型没有能力对词语的位置信息进行建模。换言之就算把一个句子的顺序打乱之后再输入到BERT模型中，模型最终得到的结果与把正确句子作输入得到的结果是相类似的。为了解决这一问题，必须人为地为序列中的词语加入位置信息，即引入相对或绝对的位置编。

公式:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k) \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases}$$

代码:

```
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-
math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        '''
        x: [seq_len, batch_size, d_model]
        '''
        x = x + self.pe[:x.size(0), :]
        return self.dropout(x)
```

四：实验

1.数据集

本项目采用百度知道领域的中文问题匹配数据集，该数据集从百度知道不同领域的用户问题中抽取构建数据。该数据集的任务定义如下：给定两个问题Q，判定该问题对语义是否匹配。

类型	文本1	文本2	标签（label）
相似文本	看图猜一电影名	看图猜电影	1
不相似文本	无线路由器怎么无线上网	无线上网卡和无线路由器怎么用	0

本数据集由训练集、验证集和测试集组成，具体统计数据如下表所示：

数据集名称	训练集大小	验证集大小	测试集大小
LCQMC	238766	8802	12500

2.模型输入：

文本数据：1代表[cls] 最终用于二分类；4代表[sep] 用于两分隔个句子；0代表[PAD]用于统一句子长度。

```
tensor([[   1, 2661, 4588, 4012, 3002, 4178, 2948,    4, 2661, 4588, 4012, 3002,
         4607, 4178, 2948,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0],
        [   1, 2876, 1682, 4012, 3002, 4607, 4178, 2948,    4, 2876, 1682, 4012,
         3002, 4178, 2948,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0],
        [   1, 3425, 4449, 2762, 4756, 3153,  806, 3285, 3475, 1456,  586, 3182,
          249,    4, 3425, 4449, 3182,  249,  402,  756,    0,    0,    0,    0,
            0,    0,    0,    0,    0],
        [   1, 4181, 3090, 2563, 2788,  743, 2965, 3851, 1319,    4,  743, 2965,
         3851, 1319, 3154, 2563, 2788,  395,  756,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0],
        [   1,  806, 3285, 2091, 3717, 4607, 4078, 1465, 4997, 1673, 2762, 2617,
            4,  806, 3285, 2091, 3717, 4607, 4078, 1465, 4997, 1673, 2762,   47,
            0,    0,    0,    0,    0],
        [   1, 3425, 4997, 3529, 2088, 2762,  683,  756,    4, 3425, 4997, 3529,
         2088, 2762,  683,  756, 4181, 4768,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0],
```

标签：表示两个文本语义是否相同

```
tensor([0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1,
        1, 1], device='cuda:0')
```

mask掩饰掉[PAD]，以防q*v的结果矩阵矩阵归一化时，占比重。

```
[[False, False, False,  ...,  True,  True,  True],
 [False, False, False,  ...,  True,  True,  True],
 [False, False, False,  ...,  True,  True,  True],
 ...,
 [False, False, False,  ...,  True,  True,  True],
 [False, False, False,  ...,  True,  True,  True],
 [False, False, False,  ...,  True,  True,  True]],

[[False, False, False,  ..., False, False, False],
 [False, False, False,  ..., False, False, False],
 [False, False, False,  ..., False, False, False],
```

3.实验结果

使用Adam优化器，loss稳步下降

```
Epoch: 0001 loss = 0.766605
Epoch: 0001 loss = 0.730736
Epoch: 0001 loss = 0.716783
Epoch: 0001 loss = 0.717496
Epoch: 0001 loss = 0.710880
Epoch: 0001 loss = 0.706449
Epoch: 0001 loss = 0.703190
Epoch: 0001 loss = 0.701020
Epoch: 0001 loss = 0.698758
Epoch: 0001 loss = 0.697495
Epoch: 0001 loss = 0.696643
Epoch: 0001 loss = 0.695177
Epoch: 0001 loss = 0.694809
Epoch: 0001 loss = 0.693870
Epoch: 0001 loss = 0.693361
Epoch: 0001 loss = 0.692844
Epoch: 0001 loss = 0.692616
Epoch: 0001 loss = 0.692352
Epoch: 0001 loss = 0.691668
```

以10批640条数据为一次检验，准确率约为62%左右浮动，结果较稳定。

```
acc:0.6215384615153846
acc:0.6384615384615384
acc:0.6153846153846154
acc:0.6238461538461539
acc:0.6215384615384156
acc:0.6415384615384656
acc:0.6115384615346156
acc:0.6230769230769231
acc:0.6384615384615384
acc:0.6046153846538464
acc:0.6153846153846154
```

五：结果分析

1.实验所得

本次实验最终准确率为62%左右，估计是因为所采用的模型比较还比较简单的缘故，只用到了多头自注意力、残差网络等机制，而且由于机器原因，词嵌入的纬度和前馈神经网络的层数有限。这些以后可以继续探索优化。

2.课下思考问题

通过本次实验，我进一步加深了对Transfomer和BERT模型的理解，也能回答出师兄和老师上课所留的思考题的答案了。

问题1：为何q*k得到的关系矩阵，mask要用1e-9,而不是0?

```
scores.masked_fill_(mask, -1e9)
```

答：因为关系矩阵a归一化时使用了幂函数，e的负无穷次方为0，而e的0次方为1。我们希望mask地方的值为0，故选择用1e-9。

问题2：为何q*k得到的关系矩阵，是按行归一化还是按列归一化?

答：都可以，因为关系矩阵是关于主对角线对称的，因为axb ==bxa，按行按列结果一样。

问题3：Transfomer中，decoder的输入来自于什么。

答：Q来自上层输出，K/V来自Encoder输入。

总结

在本门课程中，从词嵌入到LSTM模型，再到Transfomer、BERT模型，我系统学习了自然语言处理的相关知识，也进行很多实验，诗歌生成等，同时还有幸听到了朱老师的建议，人生就得做一些让自己激情澎湃的事情，对我真的很有意义。非常感谢老师和师兄们的指导，也希望老师和师兄的研究可以一切顺利！课程也能越办越好！