语义匹配任务
参考课程所讲知识和B站视频编写而成
张大庆_2201914

数据准备

In [2]:
```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import math
import pandas as pd
import numpy as np
import config
import load_data
import model
```

In [3]:
```
class dataset(Dataset):
    def __init__(self,path):
        data = pd.read_csv(path, sep='\t', names=["seq1", "seq2", "target"]).dropna
        #排序
        my_index = (data.seq1.str.len()+data.seq2.str.len()).sort_values().index
        data = data.reindex(my_index)


        seq1 = [[j for j in str(i).replace(" ", "")] for i in data["seq1"]]
        seq2 = [[j for j in str(i).replace(" ", "")] for i in data["seq2"]]

        self.seq1 = seq1
        self.seq2 =seq2
        self.target = data["target"]
        self.vocab = np.load("data/vocab.npy", allow_pickle=True).tolist()

    def __getitem__(self, index):
        seq1 = [self.vocab.get(i, self.vocab["UNK"]) for i in self.seq1[index]]
        seq2 = [self.vocab.get(i, self.vocab["UNK"]) for i in self.seq2[index]]

        return seq1,seq2,self.target[index]
    def __len__(self):
        return len(self.seq1)

    def batch_data_pro(self, batch_datas):
        DEVICE = config.DEVICE
        seq=[]
        target = []
        mask = []
        max_len = 0
        for i, j, k in batch_datas:
            if (len(i)+len(j)) > max_len:
                max_len = (len(i)+len(j))

        for i, j, k in batch_datas:
            seq.append([self.vocab["CLA"]]+i+[self.vocab["SEP"]]+j + [self.vocab["P
            target.append(k)


        seq = torch.tensor(seq, device=DEVICE)
        target = torch.tensor(target, device=DEVICE)
        mask = get_attn_pad_mask(seq, seq)
```

```
        return seq,target,mask
```

获得mask

In [4]:
```
def get_attn_pad_mask(seq_q, seq_k):
    '''
    seq_q: [batch_size, seq_len]
    seq_k: [batch_size, seq_len]
    seq_len could be src_len or it could be tgt_len
    seq_len in seq_q and seq_len in seq_k maybe not equal
    '''
    batch_size, len_q = seq_q.size()
    batch_size, len_k = seq_k.size()
    # eq(zero) is PAD token
    pad_attn_mask = seq_k.data.eq(0).unsqueeze(1)  # [batch_size, 1, len_k], False i
    return pad_attn_mask.expand(batch_size, len_q, len_k)  # [batch_size, len_q, le
```

结果：

In [5]:
```
dataset = dataset(config.PATH+"train.tsv")
dataloader = DataLoader(dataset, batch_size=config.BatchSize, shuffle=False,collate
```

结果返回三部分：
1:编码后的文本，
其中1编码是留着用于分类的、4编码是分隔符，0编码是pad
2：mask矩阵
3：target矩阵

In [6]:
```
for seq ,mask, target in dataloader:
    print("合并后的文本，4是分隔符，1是分类")
    print(seq)

    print("mask矩阵")
    print(mask)

    print("target矩阵")
    print(target)
    break
```

合并后的文本，4是分隔符，1是分类

```
tensor([[   1, 4487, 2083,    4, 4487, 2083,    0,    0,    0,    0],
        [   1,  725, 4504, 4322,    4, 4504,  725,  586,    0,    0],
        [   1, 3425, 3392, 2630,    4, 3425, 3392, 4181, 3090,    0],
        [   1,  942, 1959,  903, 2630,    4, 3154, 1961, 2630,    0],
        [   1, 2365, 4191,    4, 2762,  586, 1386, 2365, 2365,    0],
        [   1, 2428,  665,  586, 2365,    4, 2428,  665, 2365,    0],
        [   1, 3485, 1176, 2841, 2630,    4, 3154, 1961, 2630,    0],
        [   1, 4402,  325, 4029, 3207,    4, 4402,  325,   46, 3182],
        [   1, 4225, 2947,  558, 1204,    4, 4225, 2947, 1852, 2209],
        [   1, 2966, 3999, 4831,  202,    4, 3735,  564,  754, 3782],
        [   1, 4701, 3870, 2563, 2788,    4, 2563, 2788, 4701, 3870],
        [   1, 2845, 2407, 3076, 4739,    4,  906, 4078, 4771, 3519],
        [   1, 2257, 4051, 3851, 1319,    4, 3843, 4991, 3851, 1319],
        [   1, 4191, 4191, 4191, 3133,    4,  348, 3372, 1387, 3133],
        [   1, 2510, 2490, 1680, 1673,    4, 4953, 4446, 4790, 2186],
        [   1,  189, 1936, 4739, 2630,    4, 3768, 4924, 4739, 2630],
        [   1,  806, 3285, 3739, 3578,    4,  806, 3285, 5028, 3695],
        [   1, 1553,  586, 2567, 2919,    4, 1553, 4072, 2567, 2919],
        [   1, 4924, 4223, 4701, 3870,    4, 4253, 4927, 4701, 3870],
        [   1, 2584, 3977,  806, 3285,    4,  806, 3285, 5028, 3695],
        [   1, 1553, 2762,  278,  797,    4, 1553,  278,  797, 2936],
        [   1, 2239, 3792,  393,  661,    4, 2239, 3792,  393, 3133],
        [   1,   46, 2506,  805, 4789,    4, 2091, 2371,   46, 2506],
        [   1, 2347, 4412, 4562,  586,    4, 2347, 4412, 4562, 2630],
        [   1,  522,  196, 3072, 4272,    4, 1143, 2095, 3072, 4272],
        [   1, 2978, 1004,  619, 4012,    4, 2335, 4269,  619, 4012]],
       device='cuda:0')
```

mask矩阵

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

target矩阵

```
tensor([[[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,  True,  True],
         [False, False, False,  ..., False,  True,  True],
         [False, False, False,  ..., False,  True,  True],
         ...,
         [False, False, False,  ..., False,  True,  True],
         [False, False, False,  ..., False,  True,  True],
         [False, False, False,  ..., False,  True,  True]],

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

        ...,

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

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

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

         [[False, False, False,  ..., False, False, False],
          [False, False, False,  ..., False, False, False],
          [False, False, False,  ..., False, False, False],
          ...,
          [False, False, False,  ..., False, False, False],
          [False, False, False,  ..., False, False, False],
          [False, False, False,  ..., False, False, False]]], device='cuda:0')
```

二、构建模型

1.bert

bert分为三大部分：

1.将编号变为词向量，并添加位置信息。

2.对数据进行 n层 attation

3.对每个词向量进行全连接：（config.EmbedSize,2），将每个向量长度由config.EmbedSize，变成2，并取第一个词向量

In [7]:
```
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):
        #第一步！！！！
        enc_outputs = self.src_emb(enc_inputs) # [batch_size, src_len, d_model]
        enc_outputs = self.pos_emb(enc_outputs.transpose(0, 1)).transpose(0, 1) # [b

        #第二步！！！！
        for layer in self.layers:
            enc_outputs = layer(enc_outputs, mask)

        #第三步！！！！
        fl = self.fc(enc_outputs[:,0,:])


        print("模型返回值")
        print(fl)
        return fl
```

2.layer

layer分为两部分：

1:进行Attation

2:在进行一次PoswiseFeedForwardNet 前馈

In [8]:
```
class Layer(nn.Module):
    def __init__(self):
        super(Layer, self).__init__()
        self.enc_self_attn = MutiHeadAttation(config.EmbedSize,config.N_Heads,confi
        self.pos_ffn = PoswiseFeedForwardNet()

    def forward(self, enc_inputs, enc_self_attn_mask):
        # enc_outputs: [batch_size, src_len, d_model], attn: [batch_size, n_heads, s

        #第一部分！！！！
        enc_outputs = self.enc_self_attn(enc_inputs, enc_inputs, enc_inputs,
                                         enc_self_attn_mask)  # enc_inputs to

        #第二部分！！！！
        enc_outputs = self.pos_ffn(enc_outputs)  # enc_outputs: [batch_size, src_len
        return enc_outputs
```

3.Attation

Attationr分三两部分：

1:求Q、K、V矩阵

2:Q、 K(T)相乘，添加掩码，再与V相乘

3:进行词向量长度变换，将词向量由《多头长度》变成《一头长度》并返回

In [9]:
```
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):
        #第一部分！！！！
        residual, batch_size = query, query.size(0)
        qlist = self.Q(query).view(batch_size,-1,self.n_heads,self.hid_dim).transpo
        klist = self.K(key).view(batch_size,-1,self.n_heads,self.hid_dim).transpose
        vlist = self.V(value).view(batch_size,-1,self.n_heads,self.hid_dim).transpo
```

```
        #第二部分！！！！
        scores= torch.matmul(qlist,klist.transpose(-1, -2)) / np.sqrt(self.hid_dim)

        mask = mask.unsqueeze(1).repeat(1, config.N_Heads, 1, 1)

        scores.masked_fill_(mask, -1e9)

        A = self.sfmx(scores)

        list = torch.matmul(A,vlist).transpose(1, 2).reshape(batch_size, -1, self.n

        #第三部分！！！！！
        output = self.fc(list)

        return nn.LayerNorm(config.EmbedSize).to(config.DEVICE)(output+residual)
```

位置编码等

In [10]:
```
class PoswiseFeedForwardNet(nn.Module):
    def __init__(self):
        super(PoswiseFeedForwardNet, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(config.EmbedSize, config.d_ff, bias=False),
            nn.ReLU(),
            nn.Linear( config.d_ff, config.EmbedSize, bias=False)
        )
    def forward(self, inputs):
        '''
        inputs: [batch_size, seq_len, d_model]
        '''
        residual = inputs
        output = self.fc(inputs)
        return nn.LayerNorm(config.EmbedSize).to(config.DEVICE)(output+residual) #



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
        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)
```

三.train

In [1]:
```
dataset = load_data.dataset(config.PATH+"train.tsv")
```

```
loader = DataLoader(dataset, batch_size=config.BatchSize, shuffle=False,
                                    collate_fn=dataset.batch_data_pro)
model = BERT().to(config.DEVICE)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.005, momentum=0.99)
i=0



for epoch in range(15):
    for seq, target ,mask in loader:

      # outputs: [batch_size , [二维向量]]
      outputs = model(seq, mask)
      loss = criterion(outputs,target)

      i = (i + 1) % 100
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
      trainloss +=loss.item()*len(target)
      trainnum+=len(target)

      input()

torch.save(model,"model.pkl")
```

test

In [17]:
```
from sklearn.metrics import accuracy_score
dataset = load_data.dataset(config.PATH+"dev.tsv")
loader = DataLoader(dataset, batch_size=config.BatchSize, shuffle=False,
                                    collate_fn=dataset.batch_data_pro)
model = torch.load("model.pkl").eval()
i=0
for seq, target, mask in loader:

    # outputs: [batch_size * tgt_len, tgt_vocab_size]
    outputs = model(seq, mask)

    pre_label = torch.argmax(outputs[:,0,:],-1)
    acc = accuracy_score(target.cpu(),pre_label.cpu())
    print("acc:",acc)
    if i == 100:
        break
    i+=1
```

```
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
```