自然语言处理课程报告

姓名：刘铠源

班级：电创1701

学号：201783107

2021/1/8

# 任务介绍

知识库问答（knowledge base question answering,KB-QA）即给定自然语言问题，通过对问题进行语义理解和解析，进而利用知识库进行查询、推理得出答案。

# 数据格式介绍

每条数据由4行组成，第一行为问题，第二行为提取好的三元组[实体，属性，答案]，第三行为答案：

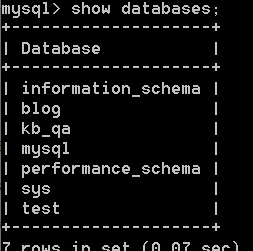


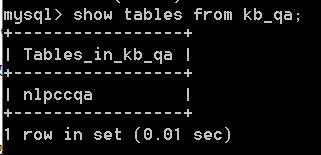
# 使用的模型与优化

BertForTokenClassification+crf+BertForSequenceClassification

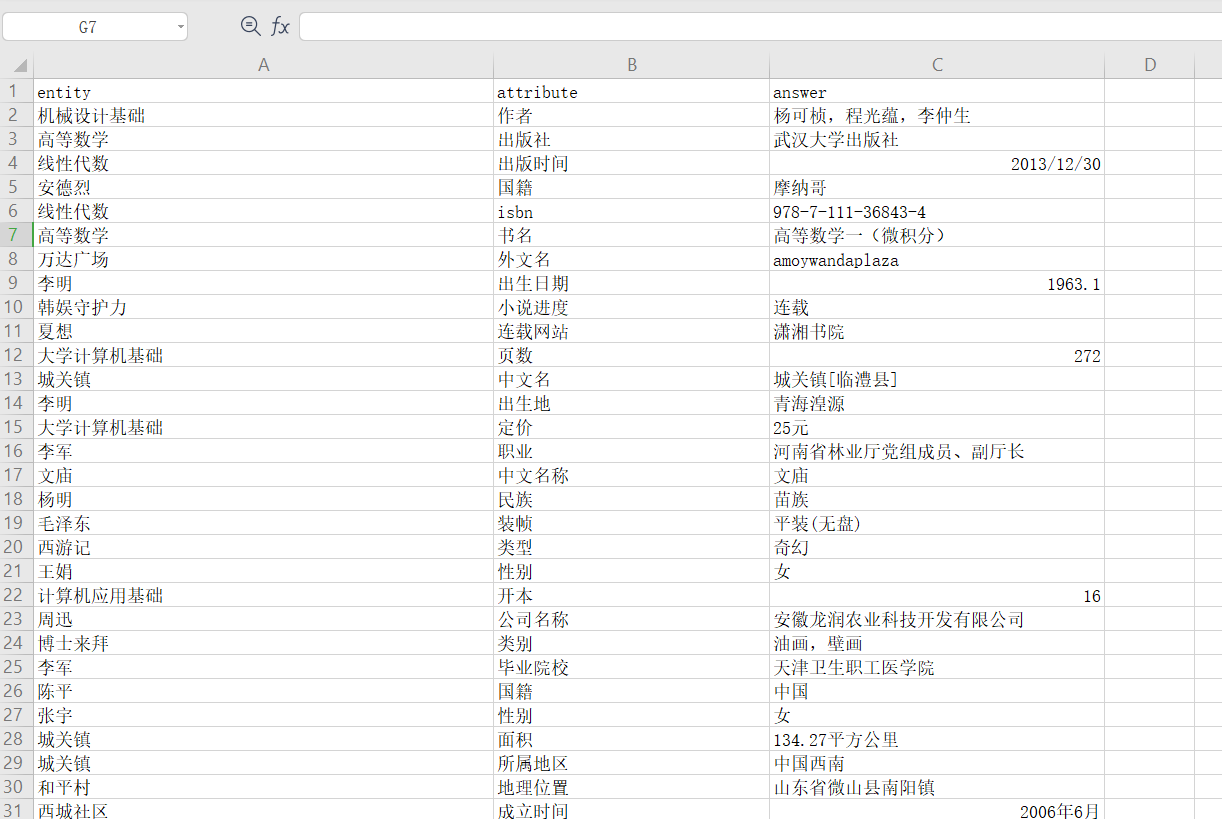
## 3.1 构建数据库

我使用的是mysql数据库，数据库命名为KB\_QA，表名为nlpccqa。





表的内容为：



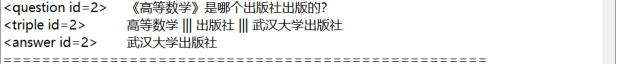
## 3.2 BertForTokenClassification+CRF

首先使用BertForTokenClassification分类器对问题进行命名实体识别序列标注得到实体，并通过CRF层学习潜在约束。

### 3.2.1 构建NER数据集

（1）具体操作

以下面这条数据为例：



标注后的结果为：

['O', 'B-LOC', 'I-LOC', 'I-LOC', 'I-LOC', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

对应：

['《','高','等','数','学','》','是','哪','个','出','版','社','出','版','的','？']

1. 结果



### 3.2.2 BertForTokenClassification+CRF训练

（1）具体操作

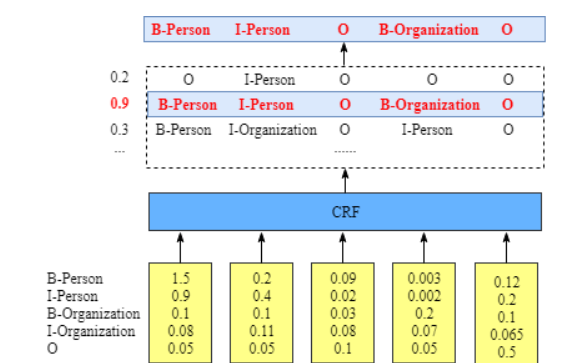
使用BertTokenizer.encode\_plus()函数得到[input\_ids,token\_type\_ids]，并在此基础上得到[input\_ids,attention\_mask,token\_type\_ids]。

labels\_ids通过NER数据集得到。

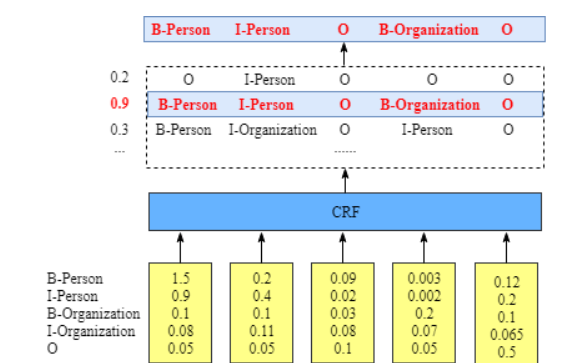
最终形成每一条数据的input为：[input\_ids,attention\_mask,token\_type\_ids,labels\_ids]

（2）模型

因为BertForTokenClassification得到的是概率：

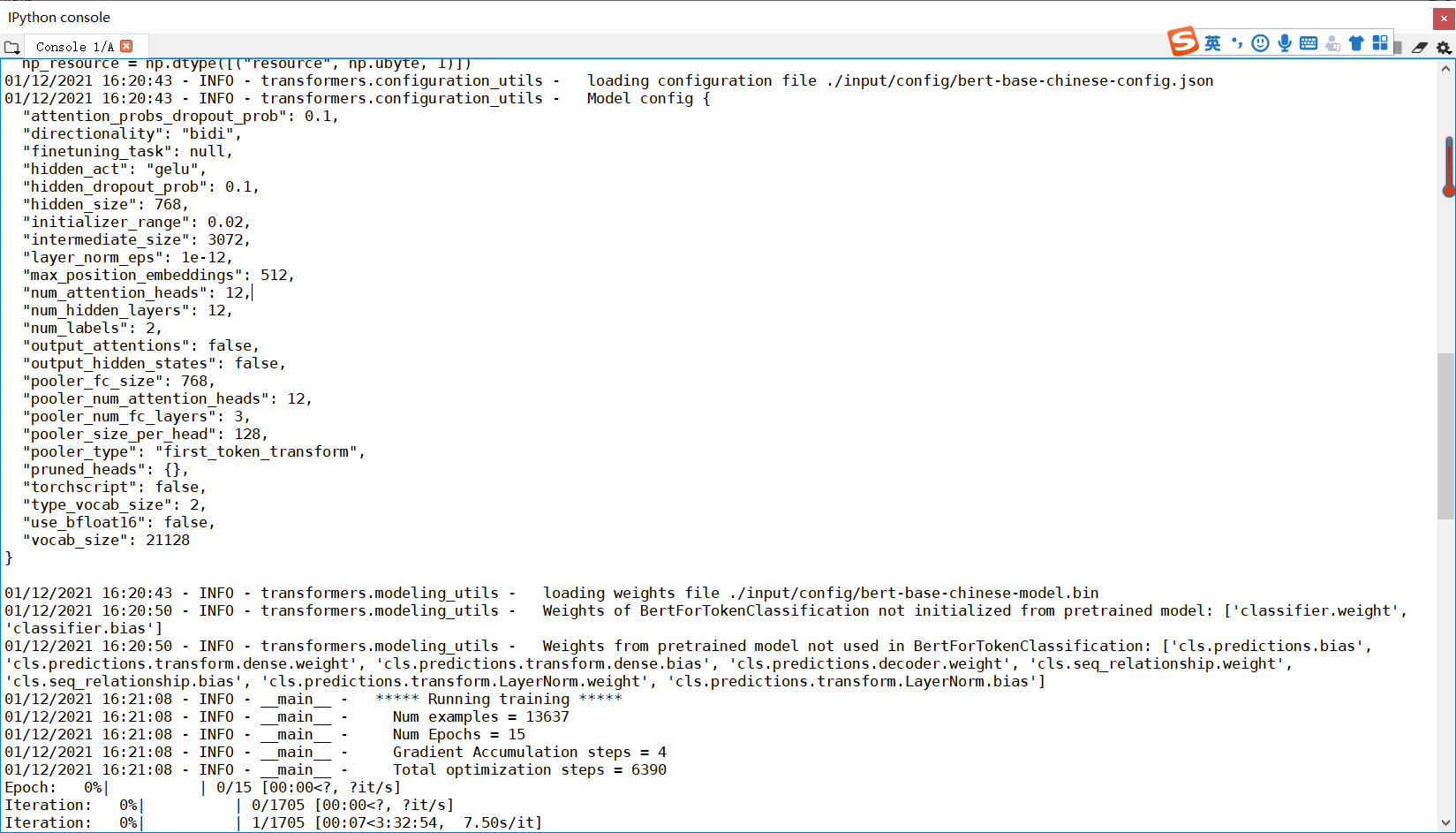


所以使用CRF来学习一些潜在的约束规则：

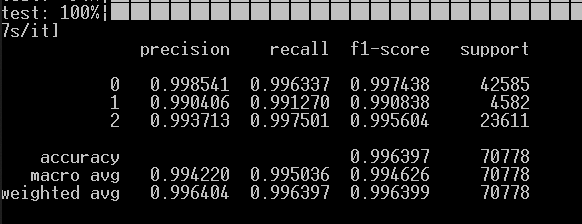


1. 结果

训练：



测试：



## 3.3 BertForSequenceClassification

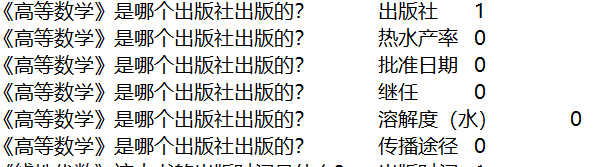
对于一个问题，如果前一步得到的实体存在于数据库中，且数据库中的属性也存在于问题中，那即得到了回答，但属性可能和问题中的提法不一致，所以使用BertForSequenceClassification进行相似判断。

### 3.3.1 构建SIM数据集

（1）具体操作

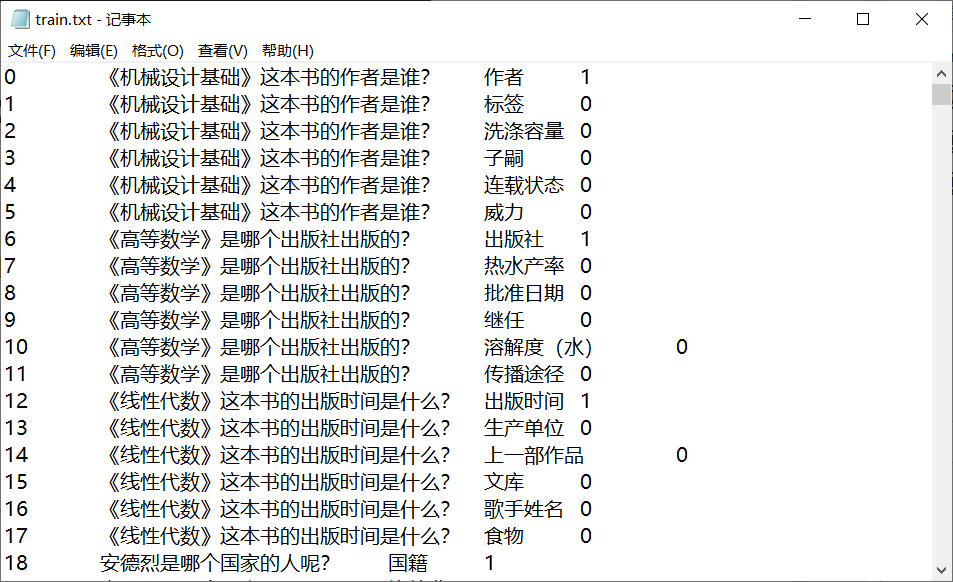
每一条数据都找五个不同的属性当做负例。

还是以上文的高等数学为例：



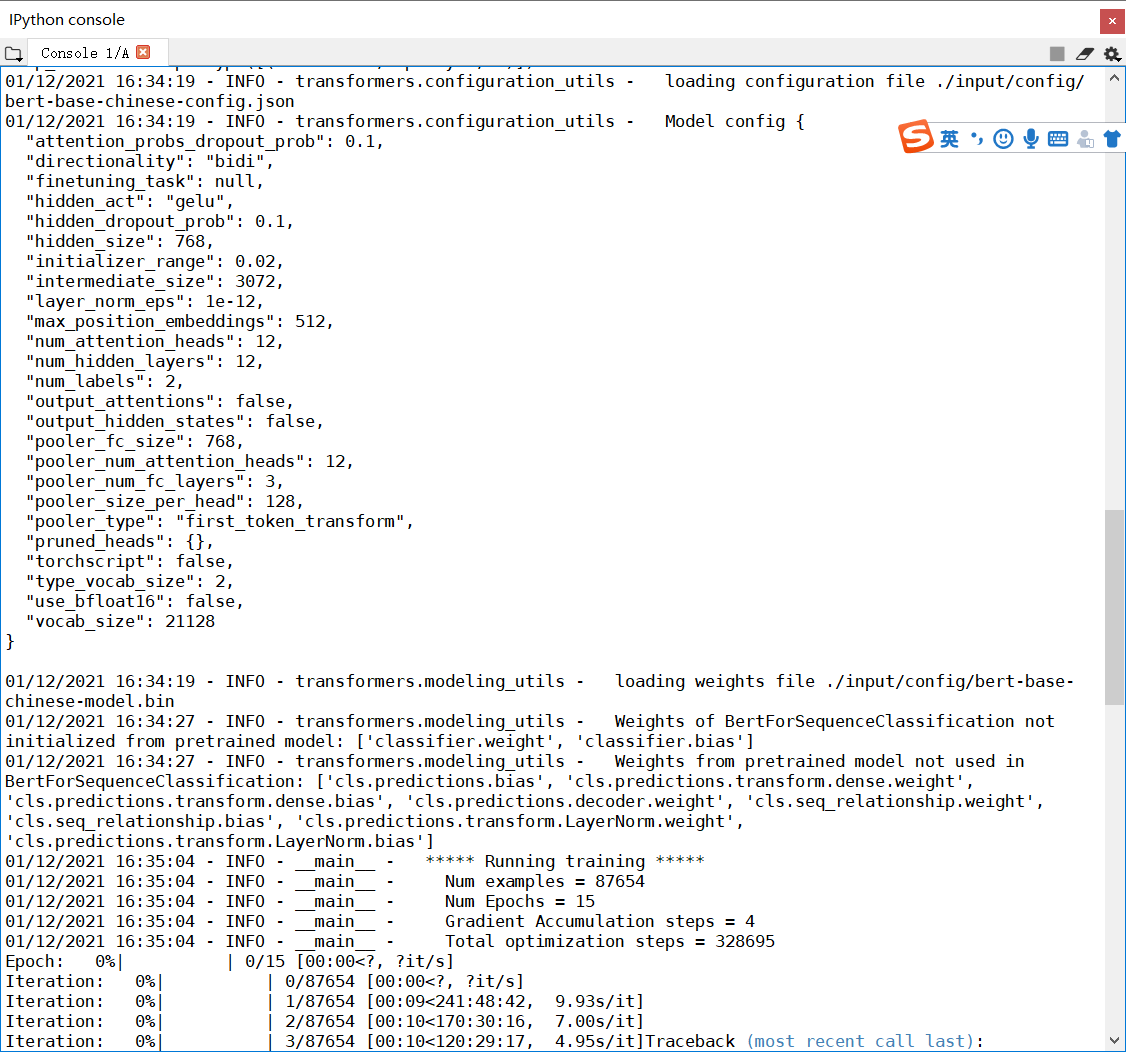
（2）结果

每一个正例，有五个负例。



### 3.3.2 BertForSequenceClassification训练

训练：



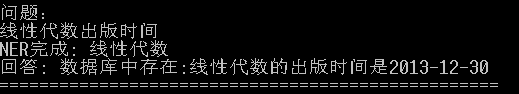
结果：



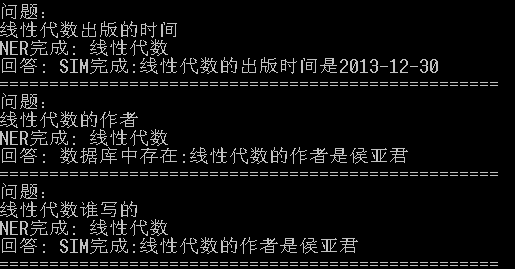
# 代码与实验效果

## 4.1 实验效果

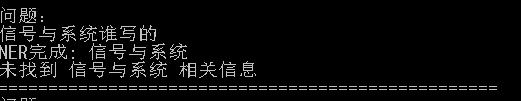
（1）对于数据库有的实体且属性存在于问题中，直接从数据库得到答案：



1. 对于数据库有的实体，属性不一致：



1. 对于数据库中不存在的实体：



## 4.2 主要代码

**CRF模型代码：**

from typing import List, Optional

import torch

import torch.nn as nn

class CRF(nn.Module):

def \_\_init\_\_(self,num\_tags : int = 2, batch\_first:bool = True) -> None:

if num\_tags <= 0:

raise ValueError(f'invalid number of tags: {num\_tags}')

super().\_\_init\_\_()

self.num\_tags = num\_tags

self.batch\_first = batch\_first

# start 到其他tag(不包含end)的得分

self.start\_transitions = nn.Parameter(torch.empty(num\_tags))

# 到其他tag(不包含start)到end的得分

self.end\_transitions = nn.Parameter(torch.empty(num\_tags))

# 从 \_compute\_normalizer 中 next\_score = broadcast\_score + self.transitions + broadcast\_emissions 可以看出

# transitions[i][j] 表示从第j个tag 到第 i 个 tag的分数

# 更正 ：transitions[i][j] 表示从第i个tag 到第 j 个 tag的分数

self.transitions = nn.Parameter(torch.empty(num\_tags,num\_tags))

self.reset\_parameters()

def reset\_parameters(self):

init\_range = 0.1

nn.init.uniform\_(self.start\_transitions,-init\_range,init\_range)

nn.init.uniform\_(self.end\_transitions,-init\_range,init\_range)

nn.init.uniform\_(self.transitions, -init\_range, init\_range)

def \_\_repr\_\_(self):

return f'{self.\_\_class\_\_.\_\_name\_\_}(num\_tags={self.num\_tags})'

def forward(self, emissions:torch.Tensor,

tags:torch.Tensor = None,

mask:Optional[torch.ByteTensor] = None,

reduction: str = "mean") -> torch.Tensor:

self.\_validate(emissions, tags = tags ,mask = mask)

reduction = reduction.lower()

if reduction not in ('none','sum','mean','token\_mean'):

raise ValueError(f'invalid reduction {reduction}')

if mask is None:

mask = torch.ones\_like(tags,dtype = torch.uint8)

# a.shape (seq\_len,batch\_size)

# a[0] shape ? batch\_size

if self.batch\_first:

# emissions.shape (seq\_len,batch\_size,tag\_num)

emissions = emissions.transpose(0,1)

tags = tags.transpose(0,1)

mask = mask.transpose(0,1)

# shape: (batch\_size,)

numerator = self.\_computer\_score(emissions=emissions,tags=tags,mask=mask)

# shape: (batch\_size,)

denominator = self.\_compute\_normalizer(emissions=emissions,mask=mask)

# shape: (batch\_size,)

llh = denominator - numerator

if reduction == 'none':

return llh

elif reduction == 'sum':

return llh.sum()

elif reduction == 'mean':

return llh.mean()

assert reduction == 'token\_mean'

return llh.sum() / mask.float().sum()

def decode(self,emissions:torch.Tensor,

mask : Optional[torch.ByteTensor] = None) ->List[List[int]]:

self.\_validate(emissions=emissions,mask=mask)

if mask is None:

mask = emissions.new\_ones(emissions.shape[:2],dtype=torch.uint8)

if self.batch\_first:

emissions = emissions.transpose(0,1)

mask = mask.transpose(0,1)

return self.\_viterbi\_decode(emissions,mask)

def \_validate(self,

emissions:torch.Tensor,

tags:Optional[torch.LongTensor] = None ,

mask:Optional[torch.ByteTensor] = None) -> None:

if emissions.dim() != 3:

raise ValueError(f"emissions must have dimension of 3 , got {emissions.dim()}")

if emissions.size(2) != self.num\_tags:

raise ValueError(

f'expected last dimension of emissions is {self.num\_tags},'

f'got {emissions.size(2)}'

)

if tags is not None:

if emissions.shape[:2] != mask.shape:

raise ValueError(

'the first two dimensions of and mask must match,'

f'got {tuple(emissions.shape[:2])} and {tuple(mask.shape)}'

)

no\_empty\_seq = not self.batch\_first and mask[0].all()

no\_empty\_seq\_bf = self.batch\_first and mask[:,0].all()

if not no\_empty\_seq and not no\_empty\_seq\_bf:

raise ValueError('mask of the first timestep must all be on')

def \_computer\_score(self,

emissions:torch.Tensor,

tags:torch.LongTensor,

mask:torch.ByteTensor) -> torch.Tensor:

# batch second

assert emissions.dim() == 3 and tags.dim() == 2

assert emissions.shape[:2] == tags.shape

assert emissions.size(2) == self.num\_tags

assert mask.shape == tags.shape

assert mask[0].all()

seq\_length,batch\_size = tags.shape

mask = mask.float()

# self.start\_transitions start 到其他tag(不包含end)的得分

score = self.start\_transitions[tags[0]]

# emissions.shape (seq\_len,batch\_size,tag\_nums)

score += emissions[0,torch.arange(batch\_size),tags[0]]

for i in range(1,seq\_length):

# if mask[i].sum() == 0:

# break

score += self.transitions[tags[i-1], tags[i]] \* mask[i]

score += emissions[i, torch.arange(batch\_size), tags[i]] \* mask[i]

# 这里是为了获取每一个样本最后一个词的tag。

# shape: (batch\_size,) 每一个batch 的真实长度

seq\_ends = mask.long().sum(dim=0) - 1

# 每个样本最火一个词的tag

last\_tags = tags[seq\_ends,torch.arange(batch\_size)]

# shape: (batch\_size,) 每一个样本到最后一个词的得分加上之前的score

score += self.end\_transitions[last\_tags]

return score

def \_compute\_normalizer(self,

emissions:torch.Tensor ,

mask: torch.ByteTensor) -> torch.Tensor:

# emissions: (seq\_length, batch\_size, num\_tags)

# mask: (seq\_length, batch\_size)

assert emissions.dim() == 3 and mask.dim() == 2

assert emissions.shape[:2] == mask.shape

assert emissions.size(2) == self.num\_tags

assert mask[0].all()

seq\_length = emissions.size(0)

# shape : (batch\_size,num\_tag)

# self.start\_transitions start 到其他tag(不包含end)的得分

# start\_transitions.shape tag\_nums emissions[0].shape (batch\_size,tag\_size)

score = self.start\_transitions + emissions[0]

for i in range(1,seq\_length):

# shape : (batch\_size,num\_tag,1)

broadcast\_score = score.unsqueeze(dim=2)

# shape: (batch\_size,1,num\_tags)

broadcast\_emissions = emissions[i].unsqueeze(1)

next\_score = broadcast\_score + self.transitions + broadcast\_emissions

next\_score = torch.logsumexp(next\_score,dim = 1)

score = torch.where(mask[i].unsqueeze(1),next\_score,score)

# shape (batch\_size,num\_tags)

score += self.end\_transitions

# shape: (batch\_size)

return torch.logsumexp(score,dim=1)

def \_viterbi\_decode(self,emissions : torch.FloatTensor ,

mask : torch.ByteTensor) -> List[List[int]]:

# emissions: (seq\_length, batch\_size, num\_tags)

# mask: (seq\_length, batch\_size)

assert emissions.dim() == 3 and mask.dim() == 2

assert emissions.shape[:2] == mask.shape

assert emissions.size(2) == self.num\_tags

assert mask[0].all()

seq\_length , batch\_size = mask.shape

# self.start\_transitions start 到其他tag(不包含end)的得分

score = self.start\_transitions + emissions[0]

history = []

# for i in range(1,seq\_length):

#

# # shape : (batch\_size,num\_tag,1)

# broadcast\_score = score.unsqueeze(dim=2)

#

# # shape: (batch\_size,1,num\_tags)

# broadcast\_emissions = emissions[i].unsqueeze(1)

#

# next\_score = broadcast\_score + self.transitions + broadcast\_emissions

#

# next\_score = torch.logsumexp(next\_score,dim = 1)

#

# score = torch.where(mask[i].unsqueeze(1),next\_score,score)

for i in range(1,seq\_length):

broadcast\_score = score.unsqueeze(2)

broadcast\_emission = emissions[i].unsqueeze(1)

next\_score = broadcast\_score + self.transitions + broadcast\_emission

next\_score, indices = next\_score.max(dim=1)

score = torch.where(mask[i].unsqueeze(1), next\_score, score)

history.append(indices)

score += self.end\_transitions

seq\_ends = mask.long().sum(dim=0) - 1

best\_tags\_list = []

for idx in range(batch\_size):

\_,best\_last\_tag = score[idx].max(dim = 0)

best\_tags= [best\_last\_tag.item()]

# history[:seq\_ends[idx]].shape (seq\_ends[idx])

for hist in reversed(history[:seq\_ends[idx]]):

best\_last\_tag = hist[idx][best\_tags[-1]]

best\_tags.append(best\_last\_tag.item())

best\_tags.reverse()

best\_tags\_list.append(best\_tags)

return best\_tags\_list

# 遇到的问题与解决

1. mysql忘记root密码

这是一个比较常见的问题，简单的百度一下就解决了。

<https://blog.csdn.net/greywolf0824/article/details/80216379>

（2）MySQL 提示The service already exists! The current server installed: D:\mysql-8.0.17-wi

由于我忘记之前安装过mysql，所以我再一次安装时会报这个错误，通过下面的博客删除一下就解决了。

<https://blog.csdn.net/qq_42411214/article/details/104827308>

# 思考感悟

以前总听说学机器学习，人工智能要学好线性代数和概率与统计，但是由于之前我做尝试的领域是CV，所以并没有遇见很多概率知识。通过本门课的学习，我终于发现了概率与统计在人工智能领域的应用。理论课上我复习并学到了许多有用的知识和算法，实验课上我学会了面对nlp问题的大概处理流程，并实践了RNN，Bert的使用。感谢老师和助教将我领入了NLP的大门。