pytorch-style的数据集构建

梁子

<2021-03-28 周日>

Table of Contents

[1. pytorch定义数据集的基本样式](#org2815cbd)

[1.1. 在此之前, 看一看数据集](#orgf87f480)

[1.2. pytorch对此的封装](#orgac22cb5)

[1.3. 调用数据集](#org731b38a)

[2. 对于NLP使用过程中的特殊说明](#org039bb00)

这篇文字写给实习生.

训练集是进行深度学习训练的最重要环节。 现以NLP领域的例子展示做实验过程中涉及到对数据集进行处理的各种操作。

# pytorch定义数据集的基本样式

## 在此之前, 看一看数据集

数据集是什么? 数据集是这样的一种迭代器(简单理解,数据集就是一个列表), 这个迭代器里的每一个元素都是一个二元组(data, label). 比如,对于分类问题, 我们的数据就是图片,文本等等, 标签就是最终的分类结果, 比如二分类问题,或许就是0和1.

所以,对于要做的实验,重点知道数据和标签是什么. 对于深度学习, 大概率 数据就是输入,标签就是输出.

## pytorch对此的封装

理解了数据集的基本格式,其实封装一个数据集就比较简单了. 需要在初始化的时候处理好数据, 然后在此基础上封装一个迭代器. 下面的代码是对一个基本例子的展示

class NLG\_Normal\_dataset(Dataset):

"""NLG input format for normal model"""

def \_\_init\_\_(self, tokenizer, mode='train'):

self.max\_sentence\_length=tokenizer.get\_max\_seq\_length()

self.tokenizer=tokenizer

self.path=os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

with open("./dicts\_data\_save\_for\_dataset.pk",'rb') as f:

traindict,testdict,tunedict=pickle.load(f)

self.dataset=[]

# 此处单拣训练集作为测试例子,其他地方从略.

for response in traindict.keys():

acts=traindict[response]

string\_act=""

for ele\_ls in acts:

per\_seq=""

for ele in ele\_ls:

per\_seq+=ele

string\_act+=per\_seq+","

output1=self.tokenizer.single\_convert\_text\_into\_indextokens\_and\_segment\_id(response)[0]

mask1=self.tokenizer.single\_convert\_text\_into\_indextokens\_and\_segment\_id(response)[2]

input2=self.tokenizer.single\_convert\_text\_into\_indextokens\_and\_segment\_id(string\_act)[0]

mask2=self.tokenizer.single\_convert\_text\_into\_indextokens\_and\_segment\_id(string\_act)[2]

self.dataset.append(((input2,mask2),(output1,mask1)))

def \_\_getitem\_\_(self,i):

return self.dataset[i]

def \_\_len\_\_(self):

return len(self.dataset)

我们不要先去看具体的细节, 这个数据集在封装模式上需要包括以下三个特点:

1. 继承自Dataset. 实际上是继承自torch.utils.data.Dataset
2. 需要有方法\_\_getitem\_\_ . 这个方法输入一个必要的索引i, 然后去返回迭代器的第i个样本. 也就是数据集里的第i个样本.
3. 需要有方法\_\_len\_\_ . 这个方法输出整个迭代器的长度.

那么, 进行初始化的时候,主要是在干嘛? 其实就是在干一件事, 获得数据集样本列表. 也就是上面代码里的self.dataset变量.

## 调用数据集

比较简单,基本语法如下面第二行展示结果.

test\_dataset=NLG\_GPT2\_dataset(tokenizer=a\_tokenizer, mode='test')

test\_loader=DataLoader(dataset=test\_dataset, batch\_size=args.batch\_size, shuffle=True)

DataLoader 来自于torch.utils.data.Dataloader

参数解释:

* batch\_szie: 规定batchsize
* shuffle: 规定是否进行shuffle
* dataset: 输入dataset

# 对于NLP使用过程中的特殊说明

NLP中, 使用huggingface transformers的话, 基本的流程(尤其对于中文)是这样的:

1. 将自然语言文本分成token, 得到token list;
2. 对token list 里的每一个元素, 都查询其在vocabulary list中的索引, 得到index tensor;
3. 将index tensor输入到模型里.

上述整个过程在上面的数据集封装中, 上述方法被self.tokenizer.single\_convert\_text\_into\_indextokens\_and\_segment\_id()所实现.

下面给出自己对已有接口的一个自定义化的工程上的封装示例.

class MyTokenizer():

def \_\_init\_\_(self, max\_sentence\_length=64):

path=os.path.dirname(os.path.abspath(\_\_file\_\_))

self.max\_sentence\_length=max\_sentence\_length

self.tokenizer=BertTokenizer.from\_pretrained(path+'/Novel\_GPT')

self.tokenizer.add\_special\_tokens({"additional\_special\_tokens":["[DOMAIN]","[NAME]",

"[NAME\_ATBTE]",

"[AT\_VALUE]",

"[POSITIVE]",

"[NEGATIVE]",

"[MAYBE\_YOU\_LIKE]",

"[DOYOU\_LIKE\_IT?]",

"[DOYOU\_LIKE\_IT?]"]})

def \_\_len\_\_(self):

return len(self.tokenizer)

def single\_convert\_text\_into\_indextokens\_and\_segment\_id(self,text):

tokeniz\_text = self.tokenizer.tokenize(text)

indextokens = self.tokenizer.convert\_tokens\_to\_ids(tokeniz\_text)

# indextokens.append(self.tokenizer.convert\_tokens\_to\_ids('[EndOfResponse]'))

input\_mask = [1] \* len(indextokens)

if self.max\_sentence\_length<len(indextokens):

indextokens=indextokens[:self.max\_sentence\_length]

segment\_id=[0]\*self.max\_sentence\_length

input\_mask=input\_mask[:self.max\_sentence\_length]

else:

pad\_indextokens = [0]\*(self.max\_sentence\_length-len(indextokens))

indextokens.extend(pad\_indextokens)

input\_mask\_pad = [0]\*(self.max\_sentence\_length-len(input\_mask))

input\_mask.extend(input\_mask\_pad)

segment\_id = [0]\*self.max\_sentence\_length

indextokens=torch.tensor(indextokens,dtype=torch.long)

segment\_id=torch.tensor(segment\_id,dtype=torch.long)

input\_mask=torch.tensor(input\_mask,dtype=torch.long)

return indextokens,segment\_id,input\_mask

def single\_convert\_text\_into\_tokeniz\_textes(self,text):

tokenize\_text=self.tokenizer.tokenize(text)

return tokenize\_text

def convert\_text\_into\_indextokens\_and\_segment\_id(self,text1,text2,spl='[SEP]'):

tokeniz\_text1 = self.tokenizer.tokenize(text1)

indextokens1 = self.tokenizer.convert\_tokens\_to\_ids(tokeniz\_text1)

length\_first=self.max\_sentence\_length//2

input\_mask1 = [1] \* len(indextokens1)

if length\_first<len(indextokens1):

indextokens1=indextokens1[:length\_first]

input\_mask1=input\_mask1[:length\_first]

else:

pad\_indextokens1 = [0]\*(length\_first-len(indextokens1))

indextokens1.extend(pad\_indextokens1)

input\_mask\_pad1 = [0]\*(length\_first-len(input\_mask1))

input\_mask1.extend(input\_mask\_pad1)

text2=spl+text2

tokeniz\_text2 = self.tokenizer.tokenize(text2)

indextokens2 = self.tokenizer.convert\_tokens\_to\_ids(tokeniz\_text2)

length\_second=self.max\_sentence\_length-length\_first

length2\_begin=length\_first

input\_mask2 = [1] \* len(indextokens2)

if length\_second<len(indextokens2):

indextokens2=indextokens2[:length\_second]

input\_mask2=input\_mask2[:length\_second]

else:

pad\_indextokens2 = [0]\*(length\_second-len(indextokens2))

indextokens2.extend(pad\_indextokens2)

input\_mask\_pad2 = [0]\*(length\_second-len(input\_mask2))

input\_mask2.extend(input\_mask\_pad2)

indextokens1.extend(indextokens2)

input\_mask1.extend(input\_mask2)

indextokens=torch.tensor(indextokens1,dtype=torch.long)

# segment\_id=torch.tensor(segment\_id,dtype=torch.long)

input\_mask=torch.tensor(input\_mask1,dtype=torch.long)

return indextokens,None,input\_mask

def convert\_ids\_to\_tokens(self, index):

return self.tokenizer.convert\_ids\_to\_tokens(index)

def convert\_prediction\_result2\_sentence(self,output\_prediction):

output\_index=output\_prediction.cpu()[0].numpy().tolist()

# print(output\_index)

return self.tokenizer.convert\_ids\_to\_tokens(output\_index)

def indextoken2wordtoken(self,index):

return self.tokenizer.convert\_ids\_to\_tokens(index)

def convert\_tokens\_to\_ids(self,token):

return self.tokenizer.convert\_tokens\_to\_ids(token)

def get\_max\_seq\_length(self):

return self.max\_sentence\_length