# 樣型識別期末報告

# 論文分類

# B0529001 馮少迪

Data是從參加這個比賽下載

一張含有 螢幕擷取畫面 的圖片

自動產生的描述一張含有 螢幕擷取畫面 的圖片

自動產生的描述

最後排在第41名，總共420隊

下載原本的train data，是如下一個csv檔案，總共7000比資料

一張含有 文字, 報紙 的圖片

自動產生的描述

通過python講資料做一個統計，轉成更容易被讀取的形式。

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

一開始最初想實作的nlp模型。通過LSTM對title和 abstract提取向量，再將其他種類等特征放一起做DNN。

如下圖：

一張含有 文字, 收據 的圖片

自動產生的描述

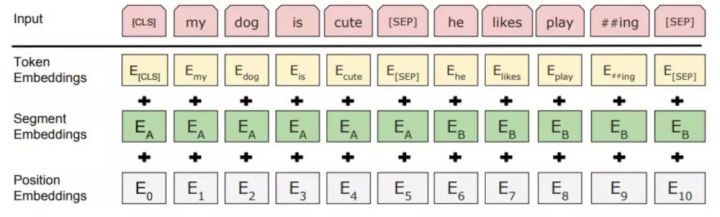
後來發現除了title和 abstract 其他值不如不用還比較准。

用lstm不如直接用google 的最新模型bert。

BERT的全称是Bidirectional Encoder Representation from Transformers，即双向Transformer的Encoder，因为decoder是不能获要预测的信息的。模型的主要创新点都在pre-train方法上，即用了Masked LM和Next Sentence Prediction两种方法分别捕捉词语和句子级别的representation。



这里的Embedding由三种Embedding求和而成：



Token Embeddings是词向量，第一个单词是CLS标志，可以用于之后的分类任务

Segment Embeddings用来区别两种句子，因为预训练不光做LM还要做以两个句子为输入的分类任务

Position Embeddings和之前文章中的Transformer不一样，不是三角函数而是学习出来的

以上來自： <https://zhuanlan.zhihu.com/p/46652512>

使用Bert有很多方法，用pytorch，fastbert，kertain等等。不過我要用的是這個最快速簡潔的fast ai。

文檔：<https://docs.fast.ai/>

The fastai library simplifies training fast and accurate neural nets using modern best practices. It's based on research in to deep learning best practices undertaken at [fast.ai](http://www.fast.ai/), including "out of the box" support for [vision](https://docs.fast.ai/vision.html#vision), [text](https://docs.fast.ai/text.html#text), [tabular](https://docs.fast.ai/tabular.html#tabular), and [collab](https://docs.fast.ai/collab.html#collab) (collaborative filtering) models. If you're looking for the source code, head over to the [fastai repo](https://github.com/fastai/fastai) on GitHub. For brief examples, see the [examples](https://github.com/fastai/fastai/tree/master/examples) folder; detailed examples are provided in the full documentation (see the sidebar).

當然自己讀文檔不如讀別人的教程，所以我找到了這個教程，教學如何在fastai中使用bert這個模型：

<https://mlexplained.com/2019/05/13/a-tutorial-to-fine-tuning-bert-with-fast-ai/>

然後要感謝這位大大在比賽討論區分享的fastai，不然我還不知道這麼牛逼的工具。

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

最後訓練n個epoch的結果，資料是分成5600/1400，因為不知道怎麼微調，感覺已經到極限了。不過這個結果用來跑test data 是過擬合的。

Code 鏈接：

<https://colab.research.google.com/drive/1ZzZuKK3UbGqNQweqT90lKBNSJ5C29zcX>

In this notebook, we'll be exploring how to use BERT with fastai for sentence classification.

import numpy as np  
import pandas as pd  
  
from pathlib import Path  
from typing import \*  
  
import torch  
import torch.optim as optim

from fastai import \*  
from fastai.vision import \*  
from fastai.text import \*  
from fastai.callbacks import \*

class Config(dict):  
 def \_\_init\_\_(self, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
 for k, v in kwargs.items():  
 setattr(self, k, v)  
   
 def set(self, key, val):  
 self[key] = val  
 setattr(self, key, val)  
  
config = Config(  
 testing=False,  
 bert\_model\_name="bert-base-uncased",  
 max\_lr=3e-5,  
 epochs=1,  
 use\_fp16=False,  
 bs=4,  
 discriminative=False,  
 max\_seq\_len=128,  
)

We'll be using the pytorch-pretrained-bert package, so install it if you do not have it yet!

BERT requires a special wordpiece tokenizer and a vocabulary to go along with that. Thankfully, the pytorch-pretrained-bert package provides all of that within the handy BertTokenizer class.

!pip install pytorch\_pretrained\_bert

# Data

BERT使用自己的词件标记器。 BERT需要将[CLS]和[SEP]令牌添加到每个序列。 BERT使用自己的预建词汇表。 让我们看一下如何处理这些问题。

使用字词标记器并处理特殊标记

编写我们自己的词件标记器并处理从词件到id的映射将是一个很大的麻烦。幸运的是，出色的pytorch-pretrained-bert软件包在其BertTokenizer中为我们提供了所有必要的信息。

from pytorch\_pretrained\_bert import BertTokenizer  
bert\_tok = BertTokenizer.from\_pretrained(  
 config.bert\_model\_name,  
)

100%|██████████| 231508/231508 [00:00<00:00, 430829.40B/s]

FastAI has its own conventions for handling tokenization, so we'll need to wrap the tokenizer within a different class. This is a bit confusing but shouldn't be that much of a hassle.

Notice we add the [CLS] and [SEP] special tokens to the start and end of the sequence here.

BERT具有多种风格，因此我们为类传递了将要使用的BERT模型的名称（在本文中，我们将使用无大小写的较小版本）。

Fastai有关于标记化的内部约定，因此我们将此标记化器包装在其自己的Tokenizer类中。这有点令人困惑，但不应该太麻烦。 Fastai有关于标记化的内部约定，因此我们将此标记化器包装在其自己的Tokenizer类中。这有点令人困惑，但不应该太麻烦。

class FastAiBertTokenizer(BaseTokenizer):  
 """Wrapper around BertTokenizer to be compatible with fast.ai"""  
 def \_\_init\_\_(self, tokenizer: BertTokenizer, max\_seq\_len: int=128, \*\*kwargs):  
 self.\_pretrained\_tokenizer = tokenizer  
 self.max\_seq\_len = max\_seq\_len  
  
 def \_\_call\_\_(self, \*args, \*\*kwargs):  
 return self  
  
 def tokenizer(self, t:str) -> List[str]:  
 """Limits the maximum sequence length"""  
 return ["[CLS]"] + self.\_pretrained\_tokenizer.tokenize(t)[:self.max\_seq\_len - 2] + ["[SEP]"]

Slightly confusingly, we further need to wrap the tokenizer above in a Tokenizer object. 如您所见，我们将在此处添加[CLS]和[SEP]令牌，并限制令牌化序列的长度。

有点令人困惑，我们需要将以上代码包装在另一个Tokenizer中，以传递给预处理器。

fastai\_tokenizer = Tokenizer(tok\_func=FastAiBertTokenizer(bert\_tok, max\_seq\_len=config.max\_seq\_len), pre\_rules=[], post\_rules=[])

Now, we need to make sure fastai uses the same mapping from wordpiece to integer as BERT originally did. Again, fastai has its own conventions on vocabulary so we'll be passing the vocabulary internal to the BertTokenizer and constructing a fastai Vocab object to use for preprocessing.

使用BERT词汇

bert令牌生成器还包含词汇，作为从单词到id的字典映射。与令牌生成器一样，由于fastai具有自己的词汇约定，因此我们需要根据bert词汇构造一个fastai Vocab对象。幸运的是，这很简单–我们可以简单地通过在词汇表中传递标记列表来做到这一点。

fastai\_bert\_vocab = Vocab(list(bert\_tok.vocab.keys()))

Now we have all the pieces we need to make BERT work with fastai! We'll load the data into dataframes and construct a validation set.

放在一起

现在，我们拥有构建数据束所需的一切。在本教程中，我们将从数据帧构建数据绑定，但是有多种加载数据的方法（请参阅官方文档）。

首先，我们使用熊猫将数据加载到数据帧中，并在其中处理验证集。

from sklearn.model\_selection import train\_test\_split  
  
DATA\_ROOT = Path("") / "data" / "jigsaw"  
  
train, test = [pd.read\_csv(DATA\_ROOT / fname) for fname in ["traindata.csv", "testdata.csv"]]  
train, val = train\_test\_split(train)  
  
if config.testing:  
 train = train.head(1024)  
 val = val.head(1024)  
 test = test.head(1024)

Now, we can build the databunch using the tokenizer and vocabulary we build above. Notice we're passing the include\_bos=False and include\_eos=False options. This is to prevent fastai from adding its own SOS/EOS tokens that will interfere with BERT's SOS/EOS tokens.

现在，我们在TextDataBunch上调用from\_df方法 注意，我们传递了include\_bos = False和include\_eos = False选项。这是因为fastai默认情况下会添加自己的bos和eos令牌，这会干扰BERT添加的[CLS]和[SEP]令牌。请注意，此选项是新选项，可能不适用于旧版本的fastai。

label\_cols = ["THEORETICAL", "ENGINEERING", "EMPIRICAL", "OTHERS"]  
  
databunch = TextDataBunch.from\_df(".", train, val, test,  
 tokenizer=fastai\_tokenizer,  
 vocab=fastai\_bert\_vocab,  
 include\_bos=False,  
 include\_eos=False,  
 text\_cols="Title",  
 label\_cols=label\_cols,  
 bs=config.bs,  
 collate\_fn=partial(pad\_collate, pad\_first=False, pad\_idx=0),  
 )

Alternatively, we can pass our own list of Preprocessors to the databunch (this is effectively what is happening behind the scenes)

让我们更深入地了解幕后到底发生了什么。上面的代码使用词片标记器和BERT词汇表初始化TokenizerProcessor和NumericizeProcessor，然后将其应用于数据框中的每个文本。

实际上，我们还可以初始化自己的TokenizerProcessor和NumericizeProcessor并将它们传递给数据束，而不是传递配置。

class BertTokenizeProcessor(TokenizeProcessor):  
 def \_\_init\_\_(self, tokenizer):  
 super().\_\_init\_\_(tokenizer=tokenizer, include\_bos=False, include\_eos=False)  
  
class BertNumericalizeProcessor(NumericalizeProcessor):  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, vocab=Vocab(list(bert\_tok.vocab.keys())), \*\*kwargs)  
  
def get\_bert\_processor(tokenizer:Tokenizer=None, vocab:Vocab=None):  
 """  
 Constructing preprocessors for BERT  
 We remove sos/eos tokens since we add that ourselves in the tokenizer.  
 We also use a custom vocabulary to match the numericalization with the original BERT model.  
 """  
 return [BertTokenizeProcessor(tokenizer=tokenizer),  
 NumericalizeProcessor(vocab=vocab)]

To use our own custom preprocessors, we'll need to modify the from\_df method to call the function above. Not the cleanest code but it will suffice.

我们只是包装标记器和词汇表，然后将它们放到管道中以预处理文本。要使用此管道，我们需要稍微修改数据绑定代码以在内部调用get\_bert\_processor。

class BertDataBunch(TextDataBunch):  
 @classmethod  
 def from\_df(cls, path:PathOrStr, train\_df:DataFrame, valid\_df:DataFrame, test\_df:Optional[DataFrame]=None,  
 tokenizer:Tokenizer=None, vocab:Vocab=None, classes:Collection[str]=None, text\_cols:IntsOrStrs=1,  
 label\_cols:IntsOrStrs=0, label\_delim:str=None, \*\*kwargs) -> DataBunch:  
 "Create a `TextDataBunch` from DataFrames."  
 p\_kwargs, kwargs = split\_kwargs\_by\_func(kwargs, get\_bert\_processor)  
 # use our custom processors while taking tokenizer and vocab as kwargs  
 processor = get\_bert\_processor(tokenizer=tokenizer, vocab=vocab, \*\*p\_kwargs)  
 if classes is None and is\_listy(label\_cols) and len(label\_cols) > 1: classes = label\_cols  
 src = ItemLists(path, TextList.from\_df(train\_df, path, cols=text\_cols, processor=processor),  
 TextList.from\_df(valid\_df, path, cols=text\_cols, processor=processor))  
 src = src.label\_for\_lm() if cls==TextLMDataBunch else src.label\_from\_df(cols=label\_cols, classes=classes)  
 if test\_df is not None: src.add\_test(TextList.from\_df(test\_df, path, cols=text\_cols))  
 return src.databunch(\*\*kwargs)

# 现在，我们可以像这样构建数据仓库。  
# this will produce a virtually identical databunch to the code above  
# databunch = BertDataBunch.from\_df(".", train, val, test,  
# tokenizer=fastai\_tokenizer,  
# vocab=fastai\_bert\_vocab,  
# text\_cols="comment\_text",  
# label\_cols=label\_cols,  
# bs=config.bs,  
# collate\_fn=partial(pad\_collate, pad\_first=False, pad\_idx=0),  
# )

# Model

Now with the data in place, we will prepare the model and loss functions. Again, the pytorch-pretrained-bert package gives us a sequence classifier based on BERT straight out of the box.

from pytorch\_pretrained\_bert.modeling import BertConfig, BertForSequenceClassification  
bert\_model = BertForSequenceClassification.from\_pretrained(config.bert\_model\_name, num\_labels=4)

Since this is a multilabel classification problem, we're using BCEWithLogitsLoss

loss\_func = nn.BCEWithLogitsLoss()  
#about bcewithlogistloss https://blog.csdn.net/qq\_22210253/article/details/85222093

And now we can build the Learner.

from fastai.callbacks import \*  
  
learner = Learner(  
 databunch, bert\_model,  
 loss\_func=loss\_func,  
)

And we're done! All the rest is fastai magic. For example, you can use half-precision training simply by calling learner.to\_fp16()

#我们还可以利用fastai必须提供的其他一些功能。例如，我们可以通过以下代码轻松地使用半精度训练。  
if config.use\_fp16: learner = learner.to\_fp16()

We can also use the learning rate finder.

learner.lr\_find()

LR Finder is complete, type {learner\_name}.recorder.plot() to see the graph.

learner.recorder.plot()

png

png

And now to actually train the model.

learner.fit\_one\_cycle(config.epochs, max\_lr=config.max\_lr)

<div>  
 <style>  
 /\* Turns off some styling \*/  
 progress {  
 /\* gets rid of default border in Firefox and Opera. \*/  
 border: none;  
 /\* Needs to be in here for Safari polyfill so background images work as expected. \*/  
 background-size: auto;  
 }  
 .progress-bar-interrupted, .progress-bar-interrupted::-webkit-progress-bar {  
 background: #F44336;  
 }  
 </style>  
 <progress value='0' class='' max='1', style='width:300px; height:20px; vertical-align: middle;'></progress>  
 0.00% [0/1 00:00<00:00]  
</div>

epoch

train\_loss

valid\_loss

time

<div>  
 <style>  
 /\* Turns off some styling \*/  
 progress {  
 /\* gets rid of default border in Firefox and Opera. \*/  
 border: none;  
 /\* Needs to be in here for Safari polyfill so background images work as expected. \*/  
 background-size: auto;  
 }  
 .progress-bar-interrupted, .progress-bar-interrupted::-webkit-progress-bar {  
 background: #F44336;  
 }  
 </style>  
 <progress value='785' class='' max='1312', style='width:300px; height:20px; vertical-align: middle;'></progress>  
 59.83% [785/1312 26:30<17:47 0.4952]  
</div>

help(learner)

See how simple that was?

# Predictions

Now to generate predictions. This is where you can get tripped up because the databunch does not load data in sorted order. So we'll have to do reorder the generated predictions to match their original order.

如果您使用的是整个数据集，则需要一段时间。 训练完模型后，我们现在要生成预测。这可能有点棘手，因为预测生成仍然不如fastai的其他部分记录在案。我们必须要注意的是，数据加载器不会按排序顺序加载数据。这意味着我们必须对预测进行重新排序以匹配原始排序（此代码是从与文本相关的学习者代码中借用的）。

def get\_preds\_as\_nparray(ds\_type) -> np.ndarray:  
 """  
 the get\_preds method does not yield the elements in order by default  
 we borrow the code from the RNNLearner to resort the elements into their correct order  
 """  
 preds = learner.get\_preds(ds\_type)[0].detach().cpu().numpy()  
 sampler = [i for i in databunch.dl(ds\_type).sampler]  
 reverse\_sampler = np.argsort(sampler)  
 return preds[reverse\_sampler, :]

test\_preds = get\_preds\_as\_nparray(DatasetType.Test)

You can generate a submission if you like, though you'll probably want to use a different set of configurations.

# sample\_submission = pd.read\_csv(DATA\_ROOT / "sample\_submission.csv")  
# if config.testing: sample\_submission = sample\_submission.head(test.shape[0])  
# sample\_submission[label\_cols] = test\_preds  
# sample\_submission.to\_csv("predictions.csv", index=False)