9102年，再不会用bert就out了，看这篇基于pytorch的bert文本分类的完整代码

conghuang
机器学习，广告算法

由于pytorch_pretrained_bert库是transformers的老版库，不再进行更新了。所以以下对原文章代码进行了更新，换成以transformers为框架的代码，并且将打印输出设置的更加简约。本文章适用于初学者，有兴趣的可上手尝试。

---------- 分割线 ----------

之前用bert一直都是根据keras-bert封装库操作的，操作非常简便（可参考苏剑林大佬博客当Bert遇上Keras：这可能是Bert最简单的打开姿势），这次想要来尝试一下基于pytorch的bert实践。

最近pytorch大火，而目前很少有博客完整的给出基于pytorch的bert的应用代码，本文从最简单的中文文本分类入手，一步一步的给出每段代码~ （代码简单清晰，读者有兴趣可上手实践）

（1）首先安装transformers库， 即：pip install transformers==4.4.2（版本为4.4.2）；

（2）然后下载预训练模型权重，这里下载的是 chinese_roberta_wwm_ext_pytorch ，下载链接为中文BERT-wwm系列模型 （这里可选择多种模型），如果下载不了，可在我的百度网盘下载：链接: pan.baidu.com/s/10BCm_q... 密码: 1upi；

（3）数据集选择的THUCNews，自行下载并整理出10w条数据，内容是10类新闻文本标题的中文分类问题（10分类），每类新闻标题数据量相等，为1w条。数据集可在我的百度网盘自行下载：链接: pan.baidu.com/s/1Crj4EL... 密码: p0wj。

下图为数据集展示（最后10行），格式为"title \t label"，标题和所属类别两列用'\t'分隔。

```
99991   银行员工办68张卡透支58万 被控诈骗或坐10年监 9
99992   安信证券：国美和苏宁08年报和09年一季报对比   8
99993   申请秘诀：如何为上常青藤大学做准备?      3
99994   中能电气：先进配电设备制造商      8
99995   以蓝色或红色搭配白色居多(图)      2
99996   胜爵士桑帅为沃尔开妙方 主教练：他就该随心所欲
99997   英国经济再次陷入衰退的可能性为20%    8
99998   生死一战数据暴涨四倍 雷霆板凳上坐着真正的主力
99999   基金再现人事变动潮 涉及近三成公司     9
100000  内地房贷风控升级 个人负资产进入监测范围 9
```

废话少说，下面进入代码阶段。（训练环境为Google Colab，GPU为T4，显存大约15G）

1 导入必要的库

```
import pandas as pd
import numpy as np
import json, time
from tqdm import tqdm
from sklearn.metrics import accuracy_score, classification_report
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, Sequent
from transformers import BertModel, BertConfig, BertTokenizer, AdamW, get_cosi
import warnings
warnings.filterwarnings('ignore')


bert_path = "bert_model/"    # 该文件夹下存放三个文件（'vocab.txt', 'pytorch_model.
tokenizer = BertTokenizer.from_pretrained(bert_path)   # 初始化分词器
```

2 预处理数据集

```
input_ids, input_masks, input_types,  = [], [], []  # input char ids, segment
labels = []          # 标签
maxlen = 30      # 取30即可覆盖99%

with open("news_title_dataset.csv", encoding='utf-8') as f:
    for i, line in tqdm(enumerate(f)):
        title, y = line.strip().split('\t')

        # encode_plus会输出一个字典，分别为'input_ids', 'token_type_ids', 'attenti
        # 根据参数会短则补齐，长则切断
        encode_dict = tokenizer.encode_plus(text=title, max_length=maxlen,
                                            padding='max_length', truncation=T

        input_ids.append(encode_dict['input_ids'])
        input_types.append(encode_dict['token_type_ids'])
        input_masks.append(encode_dict['attention_mask'])
        labels.append(int(y))

input_ids, input_types, input_masks = np.array(input_ids), np.array(input_type
labels = np.array(labels)
print(input_ids.shape, input_types.shape, input_masks.shape, labels.shape)
```

> 输出：（27秒，速度较快）
> 100000it [00:27, 3592.75it/s]
> (100000, 30) (100000, 30) (100000, 30) (100000,)

3 切分训练集、验证集和测试集

```
# 随机打乱索引
idxes = np.arange(input_ids.shape[0])
np.random.seed(2019)   # 固定种子
np.random.shuffle(idxes)
print(idxes.shape, idxes[:10])


# 8:1:1 划分训练集、验证集、测试集
input_ids_train, input_ids_valid, input_ids_test = input_ids[idxes[:80000]], in
input_masks_train, input_masks_valid, input_masks_test = input_masks[idxes[:80
input_types_train, input_types_valid, input_types_test = input_types[idxes[:80

y_train, y_valid, y_test = labels[idxes[:80000]], labels[idxes[80000:90000]],

print(input_ids_train.shape, y_train.shape, input_ids_valid.shape, y_valid.sha
      input_ids_test.shape, y_test.shape)
```

> 输出：

```
(100000,) [84968 96544   870 71358 89287 95337 74539 26224 80363 15792]
(80000, 30) (80000,) (10000, 30) (10000,) (10000, 30) (10000,)
```

4加载到pytorch的DataLoader

```
BATCH_SIZE = 64  # 如果会出现OOM问题，减小它
# 训练集
train_data = TensorDataset(torch.LongTensor(input_ids_train),
                           torch.LongTensor(input_masks_train),
                           torch.LongTensor(input_types_train),
                           torch.LongTensor(y_train))
train_sampler = RandomSampler(train_data)
train_loader = DataLoader(train_data, sampler=train_sampler, batch_size=BATCH_S

# 验证集
valid_data = TensorDataset(torch.LongTensor(input_ids_valid),
                           torch.LongTensor(input_masks_valid),
                           torch.LongTensor(input_types_valid),
                           torch.LongTensor(y_valid))
valid_sampler = SequentialSampler(valid_data)
valid_loader = DataLoader(valid_data, sampler=valid_sampler, batch_size=BATCH_S

# 测试集（是没有标签的）
test_data = TensorDataset(torch.LongTensor(input_ids_test),
                           torch.LongTensor(input_masks_test),
                           torch.LongTensor(input_types_test))
test_sampler = SequentialSampler(test_data)
test_loader = DataLoader(test_data, sampler=test_sampler, batch_size=BATCH_SIZE
```

5 定义bert模型

```
# 定义model
class Bert_Model(nn.Module):
    def __init__(self, bert_path, classes=10):
        super(Bert_Model, self).__init__()
        self.config = BertConfig.from_pretrained(bert_path)  # 导入模型超参数
        self.bert = BertModel.from_pretrained(bert_path)     # 加载预训练模型权重
        self.fc = nn.Linear(self.config.hidden_size, classes)  # 直接分类


    def forward(self, input_ids, attention_mask=None, token_type_ids=None):
        outputs = self.bert(input_ids, attention_mask, token_type_ids)
        out_pool = outputs[1]   # 池化后的输出 [bs, config.hidden_size]
        logit = self.fc(out_pool)   #  [bs, classes]
        return logit
```

可以发现，bert模型的定义由于高效简易的封装库存在，使得定义模型较为容易，如果想要在bert之后加入cnn/lstm等层，可在这里定义。

6 实例化bert模型

```
def get_parameter_number(model):
    #  打印模型参数量
    total_num = sum(p.numel() for p in model.parameters())
```

```
    trainable_num = sum(p.numel() for p in model.parameters() if p.requires_gra
    return 'Total parameters: {}, Trainable parameters: {}'.format(total_num, t

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
EPOCHS = 5
model = Bert_Model(bert_path).to(DEVICE)
print(get_parameter_number(model))
```

> 输出：Total parameters: 102275338, Trainable parameters: 102275338

7 定义优化器

```
optimizer = AdamW(model.parameters(), lr=2e-5, weight_decay=1e-4) #AdamW优化器
scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=len(tra
                                            num_training_steps=EPOCHS*len(trai
# 学习率先线性warmup一个epoch，然后cosine式下降。
# 这里给个小提示，一定要加warmup（学习率从0慢慢升上去），如果把warmup去掉，可能收敛不了。
```

8 定义训练函数和验证测试函数

```
# 评估模型性能，在验证集上
def evaluate(model, data_loader, device):
    model.eval()
    val_true, val_pred = [], []
    with torch.no_grad():
        for idx, (ids, att, tpe, y) in (enumerate(data_loader)):
            y_pred = model(ids.to(device), att.to(device), tpe.to(device))
            y_pred = torch.argmax(y_pred, dim=1).detach().cpu().numpy().tolist(
            val_pred.extend(y_pred)
            val_true.extend(y.squeeze().cpu().numpy().tolist())

    return accuracy_score(val_true, val_pred)  #返回accuracy


# 测试集没有标签，需要预测提交
def predict(model, data_loader, device):
    model.eval()
    val_pred = []
    with torch.no_grad():
        for idx, (ids, att, tpe) in tqdm(enumerate(data_loader)):
            y_pred = model(ids.to(device), att.to(device), tpe.to(device))
            y_pred = torch.argmax(y_pred, dim=1).detach().cpu().numpy().tolist(
            val_pred.extend(y_pred)
    return val_pred


def train_and_eval(model, train_loader, valid_loader,
                   optimizer, scheduler, device, epoch):
    best_acc = 0.0
    patience = 0
    criterion = nn.CrossEntropyLoss()
    for i in range(epoch):
        """训练模型"""
        start = time.time()
        model.train()
        print("***** Running training epoch {} *****".format(i+1))
        train_loss_sum = 0.0
        for idx, (ids, att, tpe, y) in enumerate(train_loader):
            ids, att, tpe, y = ids.to(device), att.to(device), tpe.to(device),
            y_pred = model(ids, att, tpe)
            loss = criterion(y_pred, y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            scheduler.step()   # 学习率变化

            train_loss_sum += loss.item()
            if (idx + 1) % (len(train_loader)//5) == 0:    # 只打印五次结果
                print("Epoch {:04d} | Step {:04d}/{:04d} | Loss {:.4f} | Time {
                        i+1, idx+1, len(train_loader), train_loss_sum/(idx+1)
                # print("Learning rate = {}".format(optimizer.state_dict()['par

        """验证模型"""
        model.eval()
        acc = evaluate(model, valid_loader, device)  # 验证模型的性能
        ## 保存最优模型
        if acc > best_acc:
            best_acc = acc
            torch.save(model.state_dict(), "best_bert_model.pth")

        print("current acc is {:.4f}, best acc is {:.4f}".format(acc, best_acc)
        print("time costed = {}s \n".format(round(time.time() - start, 5)))
```

9 开始训练和验证模型

```
# 训练和验证评估
train_and_eval(model, train_loader, valid_loader, optimizer, scheduler, DEVICE,
```

> 输出：（训练时间较长，500s左右一个epoch，这里只训练了2个epoch，验证集便得到了0.9680的accuracy）

```
***** Running training epoch 1 *****
Epoch 0001 | Step 0250/1250 | Loss 1.6724 | Time 92.7192
Epoch 0001 | Step 0500/1250 | Loss 1.0117 | Time 184.9402
Epoch 0001 | Step 0750/1250 | Loss 0.7427 | Time 276.9679
Epoch 0001 | Step 1000/1250 | Loss 0.6025 | Time 369.3089
Epoch 0001 | Step 1250/1250 | Loss 0.5148 | Time 460.9221
current acc is 0.9607, best acc is 0.9607
time costed = 509.92137s

***** Running training epoch 2 *****
Epoch 0002 | Step 0250/1250 | Loss 0.1184 | Time 92.4572
Epoch 0002 | Step 0500/1250 | Loss 0.1164 | Time 184.3489
Epoch 0002 | Step 0750/1250 | Loss 0.1156 | Time 275.9755
Epoch 0002 | Step 1000/1250 | Loss 0.1142 | Time 368.0724
Epoch 0002 | Step 1250/1250 | Loss 0.1120 | Time 459.7428
current acc is 0.9680, best acc is 0.9680
time costed = 480.61548s
```

10 加载最优模型进行测试

```
# 加载最优权重对测试集测试
model.load_state_dict(torch.load("best_bert_model.pth"))
pred_test = predict(model, test_loader, DEVICE)
print("\n Test Accuracy = {} \n".format(accuracy_score(y_test, pred_test)))
print(classification_report(y_test, pred_test, digits=4))
```

> 输出：测试集准确率为96.72%

```
157it [00:19,  7.88it/s]
 Test Accuracy = 0.9672

              precision    recall  f1-score   support

           0     0.9947    0.9936    0.9941       938
           1     0.9749    0.9698    0.9723       960
           2     0.9668    0.9788    0.9728      1040
           3     0.9629    0.9590    0.9609      1000
           4     0.9655    0.9636    0.9645       988
           5     0.9952    0.9782    0.9866      1053
           6     0.9432    0.9570    0.9501      1024
           7     0.9405    0.9787    0.9592       985
           8     0.9704    0.9554    0.9629      1031
           9     0.9604    0.9388    0.9495       981

    accuracy                         0.9672     10000
   macro avg     0.9674    0.9673    0.9673     10000
weighted avg     0.9674    0.9672    0.9672     10000
```

经过以上10步，即可建立起较为完整的基于pytorch的bert文本分类体系，代码也较为简单易懂，对读者有帮助记得点个赞支持一下呀（别光收藏呀）~

-完结-

编辑于 2021-09-21 09:17

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

PyTorch　文本分类　BERT

评论千万条，友善第一条

67 条评论　默认　最新

lrcx …
写的很好呀，我一步一步照着下来代码能运行。给乎圈树立了一个良好的风气。现在很多人写代码都不负则呢，也不管能不能运行。给你点赞
2021-05-18　回复　17

Pandas.ipynb …
我也运行了！ 答主👍太棒啦
03-20　回复　2

王车忆霸 …
请问 不管epoch多少 准确率都没有提升 是什么原因呀
11-12　回复　赞

vb巴比龙Pililion …
文件夹vocab是放什么文件🤔

文件夹vocab是放什么文件
11-08 回复 赞

迷路森林
安装pre-bert
一直报错
2020-05-20 回复 赞

赤的眼
请问 最后保存了这个模型 再加载这个模型 得到最后的结果这个过程是不是很慢啊？加载训练👌模型这个过程
2020-05-19 回复 赞

赤的眼 ‣ **conghuang**
请问 用bert做分类不需要加sigmoid或者softmax这些么？
2020-05-20 回复 赞

conghuang 作者
稍微有些慢，毕竟模型挺大
2020-05-19 回复 赞

胡文斌
感谢
2020-05-14 回复 赞

小分析
为啥我预训练模型一直下载不了呀？
2020-04-25 回复 赞

特立独行
大佬有没有兴趣做一个bert关系关系抽取的demo啊
2020-04-10 回复 赞

conghuang 作者
zhuanlan.zhihu.com/p/13... 有了！
2020-04-25 回复 赞

seekingFor
我训练时，acc不变一直为57.9.后面，发现test集输出的全为一个类别，不知道哪里出错了。
2020-04-04 回复 赞

史诗之巨魔 ‣ **M2nNg**
我也是解决了吗
2021-05-26 回复 赞

M2nNg ‣ **鱼香**
好吧。。我最近一直搞这个问题没搞明白 如果后续想起来麻烦告知下 感激不尽
2021-03-16 回复 赞

查看全部 6 条回复

嘻嘻哈哈
大佬 pytorch_pretrained_bert这个库是和transforms合并了吗
2020-04-03 回复 赞

conghuang 作者
是的
2020-04-04 回复 赞

布兰妮
训练过程中bert的参数也会更新吧
2020-03-31 回复 赞

脱发猿
应该可以param.requires_grad = True设为False冻结吧?
2020-03-31 回复 2

脱发猿
大佬，想增加训练批次的时候，优化器里的NUM_EPOCHS也要变吗?
2020-03-30 回复 赞

conghuang 作者
是的 t_total表示total number of training steps for the learning
2020-04-04 回复 赞

AOzhihuVer
大佬运行到优化器的 t_total=len(train_loader) * NUM_EPOCHS报错了 name 'num_epoches' is not defined
2020-03-22 回复 赞

conghuang 作者
抱歉啊，现在补上去了
2020-03-22 回复 赞

wasabi
请问优化器能用其他的吗，我的torch版本不支持AdamW
2021-10-21 回复 赞

conghuang 作者
可以，直接用torch自带的Adam吧
2021-10-22 回复 赞

呵呵哒
大佬，bert外接层是在out_pool后面接吗?
2021-07-30 回复 赞

呵呵哒 ‣ **conghuang**
序列标注是怎么写的，麻烦大佬了
2021-07-30 回复 赞

conghuang 作者
文本分类是，序列标注不是
2021-07-30 回复 赞

哈哈哈没想到吧
您好。我看您说一个epoch只要500s，但是我train一个epoch的时间远大于它，请问是由于我将batch_size调小导致的吗
2021-07-25 回复 赞

conghuang 作者
应该是，另外GPU不大行吧
2021-07-26 回复 赞

白糖水
大哥们，
model=Bert_Model(bert_path).to(DEVICE)
错误如下
unable to parse ---config.json as a url or as a local path
2021-05-25 回复 赞

白糖水 ‣ **Humpty dumpty**
嗦嘎，感谢
2021-07-04 回复 赞

Humpty dumpty
模型里的文件名字改一下就好了，把原来的bert_config.json重命名成config.json
2021-06-15 回复 赞

永远的link
请问下Can't set hidden_size with value 768 for BartConfig
AttributeError: can't set attribute是什么原因，为啥这个参数会报错呀😭
2021-05-21 回复 赞

永远的link ‣ **conghuang**
不好意思，我代码打错了，用成bartmodel了
2021-05-21 回复 赞

conghuang 作者
BartConfig没有这个参数hidden_size
2021-05-21 回复 赞

展开其他 1 条回复

lrcx
请问一下，您自行整理的十万条数据的标签从0-9，依次是什么啊
2021-05-18 回复 赞

conghuang 作者 ‣ **ai169**
👍
03-22 回复 赞

ai169
['体育', '娱乐', '家居', '教育', '时政', '游戏', '社会', '科技', '财经', '股票']
我自己核对了下应该是这个顺序。
03-22 回复 赞

展开其他 1 条回复

世界第二中单
你好请问数据集的格式是什么样子的，我想用我自己的数据集去测试一下
2020-12-30 回复 赞

conghuang 作者
看第一张图
2020-12-30 回复 赞

点击查看全部评论

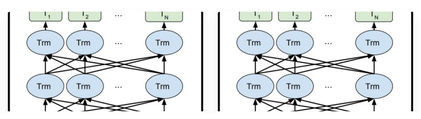
评论千万条，友善第一条

NLP成长之路
一起变得更优秀吧

自然语言

推荐阅读

Bert源码详解（Pytorch版本）
代码链接如下： GitHub - codertimo/BERT-pytorch: Google AI 2018 BERT pytorch implementation该代码在github获



30分钟带你彻底掌握Bert源码(Pytorch)，超详细！！不看...
DASOU 发表于NLP基础...

得了4400stars。如果你想要学习Bert，首先你应该去了解...
Coder... 发表于算法代码解...

一起读Bert文本分类代码(pytorch篇 三)
叫我老周就好了

一起读Bert文本分类代码(pytorch篇 五)
叫我老周就好了