~ Pytorch-使用Bert预训练模型微调中文文本分类

目录

- 1. 导包和设定超参数
- 2. 数据预处理
  - 2.1 读取文件
  - 2.2 BertTokenizer进行编码，将每一句转成数字
  - 2.3 attention_masks, 在一个文本中，如果是PAD符号则是0，否则就是1
  - 2.4 划分训练集和测试集
  - 2.5 创建DataLoader，用来取出一个batch的数据
- 3. 创建模型、优化器
  - 3.1 创建模型
  - 3.2 定义优化器
  - 3.3 学习率预热，训练时先从小的学习率开始训练
- 4. 训练、评估模型
  - 4.1 模型准确率
  - 4.2 计算模型运行时间
  - 4.3 训练模型
  - 4.4 评估模型
  - 4.5 运行训练模型和评估模型
- 5. 预测

笔记摘抄

语料链接：https://pan.baidu.com/s/1YxGGYrneByuAlRdAVoy_ZLg
提取码：tzao

neg.txt和pos.txt各5000条酒店评论，每条评论一行。

回到顶部

1. 导包和设定超参数

```
import numpy as np
import random
import torch
import matplotlib.pyplot as plt
from torch.nn.utils import clip_grad_norm_
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
from transformers import get_linear_schedule_with_warmup

SEED = 123
BATCH_SIZE = 16
learning_rate = 2e-5
weight_decay = 1e-2
epsilon = 1e-8

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
```

回到顶部

2. 数据预处理

2.1 读取文件

```
def readFile(filename):
    with open(filename, encoding='utf-8') as f:
        content = f.readlines()
        return content

pos_text, neg_text = readFile('./hotel/pos.txt'), readFile('./hotel/neg.txt')
sentences = pos_text + neg_text

# 设定标签
pos_targets = np.ones((len(pos_text)))  # (5000, )
neg_targets = np.zeros((len(neg_text))) # (5000, )
targets = np.concatenate((pos_targets, neg_targets), axis=()).reshape(-1, 1) # (10000, 1)
total_targets = torch.tensor(targets)
```

2.2 BertTokenizer进行编码，将每一句转成数字

```
model_name = 'bert-base-chinese'
cache_dir = './sample_data/'

tokenizer = BertTokenizer.from_pretrained(model_name, cache_dir=cache_dir)
print(pos_text[:3])
print(tokenizer.tokenize(pos_text[:3]))
print(tokenizer.encode(pos_text[:3]))
print(tokenizer.convert_ids_to_tokens(tokenizer.encode(pos_text[:3])))
```

```
不错，下次还来惠人住，交通也方便，在餐厅吃的不错。

['不', '错', '，', '下', '次', '还', '来', '惠', '人', '住', '，', '交', '通', '也', '方', '便', '，', '在', '餐', '厅', '吃', '的
[101, 679, 7231, 8024, 678, 3613, 6820, 5440, 5991, 1057, 857, 511, 769, 6858, 738, 3175, 912, 802
['[CLS]', '不', '错', '，', '下', '次', '还', '来', '惠', '人', '住', '，', '交', '通', '也', '方', '便', '，', '在', '餐', '厅',
```

为了使每一句的长度相等，稍作处理：

```
# 将每一句转成数字（大于120截断，小于120补 Padding，加上首位两个标识，长度总共等于128）
def convert_text_to_token(tokenizer, sentence, limit_size = 126):
    tokens = tokenizer.encode(sentence[:limit_size])  # 直接截断
    if len(tokens) < limit_size + 2:
        tokens.extend([0] * (limit_size + 2 - len(tokens)))
    return tokens

input_ids = [convert_text_to_token(tokenizer, sen) for sen in sentences]

input_tokens = torch.tensor(input_ids)
print(input_tokens.shape)                # torch.Size([10000, 128])
```

2.3 attention_masks, 在一个文本中，如果是PAD符号则是0，否则就是1

```
# 建立mask
def attention_maske(input_ids):
    atten_masks = []
    for seq in input_ids:
        seq_mask = [float(i > 0) for i in seq]  # PAD: 0, 否则: 1
        atten_masks.append(seq_mask)
    return atten_masks

atten_masks = attention_maske(input_ids)
attention_tokens = torch.tensor(atten_masks)
print(attention_tokens.shape)            # torch.Size([10000, 128])
```

- 构造input_ids 和 atten_masks 的目的 和 前面一节中提到的 encode_plus 函数返回的 input_ids 和 attention_mask 一样
- input_type_ids 和 本次任务无关，它是针对每个训练集有两个句子的任务（如问答任务）。

2.4 划分训练集和测试集

- 两个划分函数的参数 random_state 和 test_size 值要一致，才能使得 train_inputs 和 train_masks一一对应。

```
from sklearn.model_selection import train_test_split

train_inputs, test_inputs, train_labels, test_labels = train_test_split(input_tokens, total_targets,
                                                                        random_state=666, test_size=0.2)
train_masks, test_masks, _, _ = train_test_split(attention_tokens, input_tokens,
                                                 random_state=666, test_size=0.2)
print(train_inputs.shape, test_inputs.shape)    # torch.Size([8000, 128]) torch.Size([2000, 128])
print(train_masks.shape)                        # torch.Size([8000, 128])和train_inputs形状一样

print(train_inputs[0])
print(train_masks[0])


torch.Size([8000, 128]) torch.Size([2000, 128])
torch.Size([8000, 128])
tensor([  101,  2769,  6370,  4638,  3221,  3299,  1039,  4638,   117,   853,
         2769,  6230,  2533,  8821,  1039,  4638,  7599,  3419,  3291,  1962,
          671,   763,   117,  3300,   671,  2476,  1377,   809,  1298,  1309,
         4638,  3763,  1355,   119,  2456,  6379,  1920,  2197,  6379,  3249,
         6858,  7313,   106,   102,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0)
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

3.1 创建模型

```
model = BertForSequenceClassification.from_pretrained(model_name, num_labels = 2) # num_lab
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

▶ 详情

3.2 定义优化器

参数eps是为了 **提高数值稳定性** 而添加到分母的一个项(默认: 1e-8)。

```
optimizer = AdamW(model.parameters(), lr = learning_rate, eps = epsilon)
```

更通用的写法：bias和LayNorm.weight没有用权重衰减

```
no_decay = ['bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {'params' : [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)],
     'weight_decay' : weight_decay
    },
    {'params' : [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)],
     'weight_decay' : 0.0
    }
]

optimizer = AdamW(optimizer_grouped_parameters, lr = learning_rate, eps = epsilon)
```

3.3 学习率预热，训练时先从小的学习率开始训练

```
epochs = 2
# training steps 的数量: [number of batches] x [number of epochs].
total_steps = len(train_dataloader) * epochs

# 设计 learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps = 0,
                                            num_training_steps = total_steps)
```

回到顶部

4.训练、评估模型

4.1 模型准确率

```
def binary_acc(preds, labels):# preds.shape = [16, 2] labels.shape = [16, 1]
    # torch.max 返回最大值 (比赛中最重要!)
    correct = torch.eq(torch.max(preds, dim=1)[1], labels.flatten()).float()
    acc = correct.sum().item() / len(correct)
    return acc
```

4.2 计算模型运行时间

```
import time
import datetime

def format_time(elapsed):
    elapsed_rounded = int(round(elapsed))
    return str(datetime.timedelta(seconds = elapsed_rounded))# 返回 hh:mm:ss 形式的时间
```

4.3 训练模型

- 传入model的参数必须是tensor类型的；
- nn.utils.clip_grad_norm_(parameters, max_norm, norm_type=2) 用于解决神经网络训练过程合的方法：
  - 输入是（NN参数，最大梯度范数，范数类型=2）一般默认为L2 范数；
  - Tip：注意这个方法只在训练的时候使用，在测试的时候不用；

```
def train(model, optimizer):
    t0 = time.time()
    avg_loss, avg_acc = [],[]

    model.train()
    for step, batch in enumerate(train_dataloader):

        # 每隔40个batch 输出一下进度时间
        if step % 40 == 0 and not step == 0:
            elapsed = format_time(time.time() - t0)
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {}'.format(step, len(train_dataloader), elapsed))

        b_input_ids, b_input_mask, b_labels = batch[0].long().to(device), batch[1].long().to(device), ba

        output = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_
        loss, logits = output[0], output[1]      # loss: 损失; logits: predict

        avg_loss.append(loss.item())

        acc = binary_acc(logits, b_labels)     # (predict, label)
        avg_acc.append(acc)

        optimizer.zero_grad()
        loss.backward()
        clip_grad_norm_(model.parameters(), 1.0) # 大于1的梯度将其设为1.0, 以防梯度爆炸
        optimizer.step()                         # 更新模型参数
        scheduler.step()                         # 更新learning rate

    avg_acc = np.array(avg_acc).mean()
    avg_loss = np.array(avg_loss).mean()
    return avg_loss, avg_acc
```

- 此处 output的形式为（元组类型，第0个元素是loss值，第1个元素是每个batch中好评和差评的概率）：

```
(tensor(0.0210, device='cuda:0', grad_fn=<NllLossBackward>),
tensor([[-2.9815,  2.6923],
        [-3.2350,  3.1939],
        [-3.0775,  3.0713],
        [ 2.0191, -2.3688],
        [ 3.1146, -2.7957],
        [ 3.7798, -2.7410],
        [-0.3273,  0.8227],
        [ 2.5012, -1.5535],
        [-3.0231,  3.0162],
        [ 3.4146, -2.5583],
        [ 3.3104, -2.2134],
        [ 3.3776, -2.5192],
        [-2.6553,  2.5108],
        [-3.3891,  2.9516],
        [ 3.2397, -2.0473],
        [-2.8622,  2.7395]], device='cuda:0', grad_fn=<AddmmBackward>))
```

4.4 评估模型

调用model模型时不传入label值。

```
def evaluate(model):
    avg_acc = []
    model.eval()       # 表示进入测试模式

    with torch.no_grad():
        for batch in test_dataloader:
            b_input_ids, b_input_mask, b_labels = batch[0].long().to(device), batch[1].long().to(device)

            output = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask)

            acc = binary_acc(output[0], b_labels)
            avg_acc.append(acc)

    avg_acc = np.array(avg_acc).mean()
    return avg_acc
```

此处output的形式为（元组类型，**第0个元素是每个batch中好评和差评的概率**）：(区别可label的model)

```
(tensor([[ 3.8217, -2.7519],
         [-2.7686,  2.6035],
         [-2.6317,  2.9050],
         [-3.3734,  3.2591],
         [-2.8692,  2.6741],
         [-3.2784,  2.9276],
         [-3.4846,  -2.8891],
         [-3.7855, -2.8623],
         [-2.2249,  2.4338],
         [-2.4257,  2.4604],
         [-3.3795, -2.5700],
         [-3.1386,  3.0841],
         [-3.6883, -2.9492],
         [-3.2883, -2.3600],
         [-2.6725, -2.0778],
         [-3.1868,  3.1106]], device='cuda:0'),)
```

4.5 运行训练模型和评估模型

```
for epoch in range(epochs):

    train_loss, train_acc = train(model, optimizer)
    print('epoch={},训练准确率={}, 损失={}'.format(epoch, train_acc, train_loss))

    test_acc = evaluate(model)
    print("epoch={},测试准确率={}".format(epoch, test_acc))
```

▶ 详情

回到顶部

5. 预测

```
def predict(sen):

    input_id = convert_text_to_token(tokenizer, sen)
    input_token =  torch.tensor(input_id).long().to(device)          #torch.Size([128])

    atten_mask = [float(i>0) for i in input_id]
    attention_token = torch.tensor(atten_mask).long().to(device)      #torch.Size([128])

    output = model(input_token.view(1, -1), token_type_ids=None, attention_mask=attention_token
    print(output[0])

    return torch.max(output[0], dim=1)[1]

label = predict('酒店位置绝佳，环境不大好，隔音差，下次不会再来的。')
print('好评 !' if label==1 else '差评')

label = predict('酒店还可以，值得入质量热情，卫生合格，空间也比较大，不足的地方就是没有窗户')
print('好评 !' if label==1 else '差评')

label = predict("服务各方面没有不满到的地方，各方面没有没想到的细节")
print('好评 !' if label==1 else '差评')


tensor([[ 2.3774, -4.0351]], device='cuda:0', grad_fn=<AddmmBackward>)
差评
tensor([[-2.5653,  2.7316]], device='cuda:0', grad_fn=<AddmmBackward>)
好评
tensor([[-2.0390,  1.8002]], device='cuda:0', grad_fn=<AddmmBackward>)
好评
```

分类: A---自然语言处理 A---pytorch框架 A---深度学习实战

标签: BERT

好文要顶 关注我 收藏该文

douzujun
关注 - 9
粉丝 - 238
+加关注

0 推荐 0 反对

- 上一篇： Pytorch-Bert预训练模型的使用（调用transformers）
- 下一篇： 生成式神经网络

posted @ 2020-08-28 22:40 douzujun 阅读(1748) 评论(2) 编辑 收藏 举报

刷新评论 刷新页面 返回顶部

登录后才能查看或发表评论，立即 登录 或者 逛逛 博客园首页

华为云
如何加速
App 增长和变现？
新手成长课程实操上手
免费学习

编辑推荐：
· 技术管理进阶 —— 关于成本优化与利益分配机制
· 微前端框架single-spa初探
· 并发编程之：深入解析线程池
· 源码解析 .Net 中 Host 主机的构建过程
· 理解ASP.NET Core - Dependency Injection

最新新闻：
· 微软终于2021年10月12日终止对Silverlight的支持 (2021-09-14 20:15)
· 超42万手机、20万件商品 全国首家京东MALL将于开业3日开业 (2021-09-14 20:00)
· 飞猪公布10球状速通比赛冲刺“最强方案”：全程跟跑+最终30% (2021-09-14 19:00)
· 收割智商税的快递盲盒：一线精心策划的骗局 (2021-09-14 18:30)
· 等选手的落笔，等不到Apple Car (2021-09-14 18:20)
· 更多新闻...

历史上的今天：
2016-08-28 Linear regression with multiple variables(多特征的线型回归)算法实例_梯度下降算法(Gradient Des...
2016-08-28 Linear regression with one variable算法实例详解（绘制图像、cost_Function /Gradient Desent) 和...

Page Counter

昵称: douzujun
园龄: 5年11个月
粉丝: 238
关注: 9
+加关注

< 2021年9月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

搜索

找找看

我的标签

pytorch(31)

科研(18)

Paper(18)

深度学习(15)

attention&transformer(11)

BERT(9)

NLP(9)

博客园配置(5)

面试(4)

积分与排名

积分 - 534702

排名 - 826

随笔分类 (979)

A---cs224n_nlp(1)

A---Paper Reading(18)

A---pytorch框架(52)

A---TensorFlow(8)

A---概率论/线代/统计(2)

A---机器学习笔记(42)

A---面试题总结(4)

A---篇章关系分析(11)

A---搭希鹏-rndl Exercise(5)

A---深度学习笔记(87)

A---深度学习实战(19)

A---苏大/LeetCode周赛(12)

A---自然语言处理(21)

B---C++11特性(12)

B---C++STL(3)

B---C++并发网络通信(1)

B---C++面试题总结(20)

B---C++面向对象程序设计(2)

B---C++容器(1)

B---C++易错知识点(17)

B---C++智能指针(4)

B---Docker(2)

B---Linux多线程服务器端编程(4)

B---Linux学习笔记(49)

B---python爬虫(26)

B---TCP/IP网络编程(4)

C---数据挖掘笔记(21)

C---语音识别常识以及基础知识(2)

C---语音信号处理--8.声源面向(1)

C---语音信号处理--6.波束形成(1)

language：Android(8)

language：C#(9)

language：C/C++(45)

language：Java(15)

language：JSP(20)

language：LaTeX(9)

language：Python(121)

language：QT5(10)

language：R语言(2)

language：Shell(13)

online judge：LeetCode(27)

online judge：LeetCode--链表(8)

online judge：LeetCode--树(12)

online judge：LeetCode--搜索(8)

online judge：NYOJ(2)

online judge：POJ(19)

online judge：UVa(4)

online judge：剑指offer(1)

工具：Git(4)

工具：SVN(1)

蓝桥杯(7)

其他 博客园配置(8)

其他 操作系统笔记(23)

其他 大一玩耍程序(6)

其他 计算机网络笔记(5)

其他 计算机组成原理(1)

其他 软件测试学习笔记(4)

其他 数据结构(17)

其他 数据库(SQL Server)(25)

其他 数学建模笔记(24)

其他 私人笔记(29)

其他 语音识别(14)

算法 并查集(4)

算法 博弈问题(3)

算法 动态规划(7)

算法 二分(5)

算法 枚举(1)

算法 模拟(1)

算法 排序(3)

算法 数论(9)

算法 搜索(21)

算法 贪心(31)

算法 图论(12)

算法 优先队列(1)

算法 字符串(4)

算法竞赛入门经典笔记(29)

阅读：程序员自我修养—链接、装载与库(1)

随笔档案 (760)

2021年9月(5)

2021年8月(3)

2021年7月(2)

2021年5月(4)

2021年4月(4)

2021年3月(3)

2021年2月(3)

2021年1月(2)

2020年12月(2)

2020年11月(13)

2020年10月(23)

2020年9月(24)

2020年8月(15)

2020年7月(32)

2020年6月(50)

2020年5月(9)

2020年4月(2)

2020年3月(8)

2020年2月(23)

2020年1月(23)

2019年12月(1)

2019年6月(8)

2019年5月(31)

2019年4月(54)

2019年3月(33)

2019年2月(20)

2019年1月(32)

2018年12月(12)

2018年8月(2)

2018年7月(1)

2018年6月(9)

2018年5月(9)

2018年4月(4)

2018年3月(41)

2018年2月(16)

2018年1月(13)

2017年12月(5)

2017年11月(13)

2017年10月(7)

2017年9月(18)

2017年8月(18)

2017年7月(4)

2017年6月(3)

2017年5月(18)

2017年4月(28)

2017年3月(48)

2017年2月(21)

2017年1月(5)

2016年12月(8)

2016年11月(4)

2016年10月(8)

2016年9月(4)

2016年8月(8)

2016年7月(6)

2016年6月(8)

2016年5月(3)

2016年4月(3)

相册 (12)

壁纸(12)

好友

豆子

赵yun

天涯

快刀切草莓

阅读排行榜

1. C#_连接数据库实现 登录注册界面(50902)
2. 深入学习c++—多线程编程（一）(42427)
3. Matplotlib中plt.rcParams用法（设置图像细节）(28819)
4. SQL语句（十）查询结果排序(22835)
5. 虚拟机下Linux终端配置网络的方法(20162)
6. PyTorch的nn.Linear(详解)(19380)
7. extern "C" 含义(16171)
8. wav音频文件格式解析【个人笔记】(由用)(15201)
9. SQL语句（二）创建带主键和约束的数据表(15070)
10. "1机大战"游戏_Java(13548)

推荐排行榜

1. 数学建模 碎纸片的拼接复原 灰色关联算法(4)
2. Matplotlib中plt.rcParams用法（设置图像细节）(3)
3. 使用Javar代码发送SMTP邮件(3)
4. SQL语句（十）查询结果排序(3)
5. QT_地图导航 源码下载(3)

1. 导包和设定超参数
2. 数据预处理
   2.1 读取文件
   2.2 BertTokenizer进行编码，将每一句转成数字
   2.3 attention_masks, 在一个文本中，如果...
   2.4 划分训练集和测试集
   2.5 创建DataLoader，用来取出一个batch的数...
3. 创建模型，优化器
   3.1 创建模型
   3.2 定义优化器
   3.3 学习率预热，训练时先从小的学习率开始训练
4. 训练、评估模型
   4.1 模型准确率
   4.2 计算模型运行时间
   4.3 训练模型
   4.4 评估模型
   4.5 运行训练模型和评估模型
5. 预测

Copyright © 2021 douxuyun
Powered by .NET 5.0 on Kubernetes