预训练模型应用案例

实验报告

姓名:王雨静

班级 ： 1期NLP训练营

日期:2021年4月21日

目录

一、案例简介

2018年，Google提出了预训练语言模型BERT，该模型在各种NLP任务上都取得了很好的效果。与此同时，它的使用十分方便，可以快速地对于各种NLP任务进行适配。因此，BERT已经被广泛地使用到了各种NLP任务当中。在本案例中，我们会亲手将BERT适配到长文本关系抽取任务DocRED上，从中了解BERT的基本原理和技术细节。关系抽取是自然语言处理领域的重要任务，DocRED中大部分关系需要从多个句子中联合抽取，因此需要模型具备较强的获取和综合文章中信息的能力，尤其是抽取跨句关系的能力。

BERT

BERT是目前最具代表性的预训练语言模型，如今预训练语言模型的新方法都是基于BERT进行改进的。研究者如今将各种预训练模型的使用代码整合到了transformers这个包当中，使得我们可以很方便快捷地使用各种各样的预训练语言模型。在本实验中，我们也将调用transformers来使用BERT完成文档级别关系抽取的任务。

基于transformers的基础后，我们的主要工作就是将数据处理成BERT需要的输入格式，以及在BERT的基础上搭建一个能完成特定任务的模型。在本次实验中，我们的重点也将放在这两个方面。

首先是对于数据的处理，对于给定的文本，我们需要使用BERT的tokenizer将文本切成subword，然后转换成对应的id输入进模型中。通常来说这个过程是比较简单的，但是针对于DocRED这个任务，我们需要有一些额外注意的事情。文档级关系抽取的目标是从一段话中确定两个实体之间的关系，为了让模型知道我们关心的两个实体是什么，我们需要在文本中插入四个额外的符号，将实体标注出来。这一部分是数据处理的关键，也是需要同学们自己去动手实现的部分。

与此同时，BERT模型是一个语言模型，为了能使其适配关系抽取任务，我们需要加入额外的神经网络，使得模型能够进行关系预测。通常来说这个神经网络就是将文本中的第一个字符拿出来输入到一个线性层中进行分类。实现一个BertDocRED模型是本次作业的第二个任务，

数据和代码

本案例使用了DocRED的数据，并提供了一个简单的模型实现，包括数据的预处理、模型的训练、以及简单的评测。中间有两部分代码需要同学进行填充。数据预处理的代码

在preprocessing.ipynb，里面有一个TODO标识的位置需要进行代码填充。在处理完数据之后，再运行DocRED.ipynb进行训练，这里也有一个TODO位置需要进行代码填充。

注意由于预训练模型很大，因此需要调整batch size使得GPU能够放得下，于此同时为了提高batch size的绝对大小，可以使用梯度累积的技术。我们也提供了对应的实现，具体数值留给同学们进行探索。

评分要求

分数由两部分组成。首先，完成对于代码的填空，并且训练模型，评测模型在开发集上的结果，这部分占80%，评分依据为模型的开发集性能和代码的实现情况。第二部分，进行进一步的探索和尝试，我们将在下一小节介绍可能的尝试。同学需要提交代码和报告，在报告中对于两部分的实验都进行介绍，主要包括开发集的结果以及尝试的具体内容。

探索和尝试

- 完成对于测试数据的评测，并且提交到DocRED的评测系统中。(推荐先完成该任务，主要考察同学将模型真正应用起来的能力）
- 使用别的预训练语言模型完成该实验，例如RoBERTa等。
- 对于模型进行改进，提升关系抽取的能力，这里可以参考一些DocRED最新工作，进行复现。

参考资料

- DocRED: A Large-Scale Document-Level Relation Extraction Dataset. ACL 2019.

二、代码填充

2.1 preprocessing中增加token

```
# TODO 添加在 new_sents 里面加入 [unused0] [unused1] [unused2] [unused3]
的代码
for e in entity1_:
  tokens = tokenizer.tokenize(e['name'])
  length = len(tokens)
  sent = new_sents[e['sent_id']]
  for index in range(len(sent)):
      if sent[index:index+length] == tokens:
          sent[index:index+length] = ['[unused0]'] +
sent[index:index+length] + ['[unused1]']
          break

for e in entity2_:
  tokens = tokenizer.tokenize(e['name'])
  length = len(tokens)
  sent = new_sents[e['sent_id']]
  for index in range(len(sent)):
      if sent[index:index+length] == tokens:
          sent[index:index+length] = ['[unused2]'] +
sent[index:index+length] + ['[unused3]']
          break
```

加好后的例子:

```
E1: ['Zest Airways, Inc.', 'Asian Spirit and Zest Air', 'AirAsia Zest',
'AirAsia Zest']
E2: ['Ninoy Aquino International Airport', 'Ninoy Aquino International
Airport']

tokens: ['[CLS]', '[unused0]', 'ze', '##st', 'airways', ',', 'inc',
'.', '[unused1]', 'operated', 'as', '[unused0]', 'air', '##asia', 'ze',
'##st', '[unused1]', '(', 'formerly', '[unused0]', 'asian', 'spirit',
'and', 'ze', '##st', 'air', '[unused1]', ')', ',', 'was', 'a', 'low',
```

```
'-', 'cost', 'airline', 'based', 'at', 'the', '[unused2]', 'nino',
'##y', 'aquino', 'international', 'airport', '[unused3]', 'in', 'pas',
'##ay', 'city', ',', 'metro', 'manila', 'in', 'the', 'philippines',
'.', 'it', 'operated', 'scheduled', 'domestic', 'and', 'international',
'tourist', 'services', ',', 'mainly', 'feeder', 'services', 'linking',
'manila', 'and', 'cebu', 'with', '24', 'domestic', 'destinations',
'in', 'support', 'of', 'the', 'trunk', 'route', 'operations', 'of',
'other', 'airlines', '.', 'in', '2013', ',', 'the', 'airline',
'became', 'an', 'affiliate', 'of', 'philippines', 'air', '##asia',
'operating', 'their', 'brand', 'separately', '.', 'its', 'main',
'base', 'was', '[unused2]', 'nino', '##y', 'aquino', 'international',
'airport', '[unused3]', ',', 'manila', '.', 'the', 'airline', 'was',
'founded', 'as', 'asian', 'spirit', ',', 'the', 'first', 'airline',
'in', 'the', 'philippines', 'to', 'be', 'run', 'as', 'a',
'cooperative', '.', 'on', 'august', '16', ',', '2013', ',', 'the',
'civil', 'aviation', 'authority', 'of', 'the', 'philippines', '(',
'ca', '##ap', ')', ',', 'the', 'regulating', 'body', 'of', 'the',
'government', 'of', 'the', 'republic', 'of', 'the', 'philippines',
'for', 'civil', 'aviation', ',', 'suspended', 'ze', '##st', 'air',
'flights', 'until', 'further', 'notice', 'because', 'of', 'safety',
'issues', '.', 'less', 'than', 'a', 'year', 'after', 'air', '##asia',
'and', 'ze', '##st', 'air', "'", 's', 'strategic', 'alliance', ',',
'the', 'airline', 'has', 'been', 'rebranded', 'as', '[unused0]', 'air',
'##asia', 'ze', '##st', '[unused1]', '.', 'the', 'airline', 'was',
'merged', 'into', 'air', '##asia', 'philippines', 'in', 'january',
'2016', '.', '[SEP]']
```

2.2 DocRED中forward函数

```
def forward(self, input_ids, token_type_ids, attention_mask,
relation_multi_label=None):

    # TODO
    # 实现DocRED模型的训练和预测过程
    # 可以参考普通文本分类的BERT实现，十分类似
```

```
pooler_logits = self.bert(input_ids, token_type_ids,
attention_mask, output_hidden_states=False).pooler_output
final_logits = self.decoder(pooler_logits)

if relation_multi_label is not None:
    if relation_multi_label.dtype != final_logits.dtype:
        relation_multi_label = relation_multi_label.half()
    train_loss = BCEWithLogitsLoss()(final_logits,
relation_multi_label)
    return train_loss
else:
    return torch.sigmoid(final_logits)
```

三、preprocessing

Max_seq_length = 512
训练集包括样本1189412个，其中正例35416个，负例1153996个。（6g）
开发集包括样本392062个，其中正例11437个，负例380625个。(2g)
测试集样本共388230个。(2g)

四、训练模型

3.1 首次训练

3.1.1 参数调整

Max_seq_length = 512时，train_annotated_cls_data.tx有6.5G，memory中装不下。
Max_seq_length = 256时，train_annotated_cls_data.tx仍有2G，memory依旧不能全部装下。
但是Max_seq_length = 128时，没有sample留下了（0G）。
因此，实验中选取Max_seq_length = 256，只读了train的前300000行和dev的前30000行，
train : dev 约为 10 ： 1。

Num split examples = 9964 Num split examples = 9964
Batch size = 2

3.1.2 结果（节选）

第一个epoch：
Epoch = 1/10 steps = 20 loss = 0.453748
Epoch = 1/10 steps = 500 loss = 0.101070
Epoch = 1/10 steps = 9960 loss = 0.050426
Precision:0.000, Recall:0.000, F1-score:0.000
Precision_ignore:0.000, Recall_ignore:0.000, F1-score_ignore:0.000

第10个epoch后：
Epoch = 10/10 steps = 9960 loss = 0.047708
Evaluating: 0%| | 0/938 [00:00<?, ?it/s]Start evaluating

Evaluating: 100%|██████████| 938/938 [08:59<00:00, 1.74it/s, F1=0.00000]
Precision:0.000, Recall:0.000, F1-score:0.000
Precision_ignore:0.000, Recall_ignore:0.000, F1-score_ignore:0.000

3.1.3 分析

样本中正例样本数为9964，而负例样本有288956个，非常不均衡（unbalanced）。
在第一个epoch中，loss就下降得很快，第一个epoch结束时的loss已经非常接近第10个epoch结束时的loss值了。这很可能是因为负例样本占的比例非常大，导致模型将几乎所有的都预测为0了，这样整体loss值也会比较小。也因此模型的预测能力很差，precision、recall、F1-score都非常低（为0）。

Batch size = 2 比较小，模型训练比较慢，可以考虑增大batch size。

3.2 第二次训练

3.2.1调整

减小负例样本的数量：随机抽取与正例样本样本相同数量的负例样本。

```
import random

new_neg_set = []
for i in range(len(pos_features)):
  new_neg_set.append(neg_features[random.randint(0,
len(neg_features)-1)])

neg_features = new_neg_set
```

为发挥GPU的作用，并利用梯度累积增大batch size，将Batch size 改为32，gradient_accumulation_steps = 4

3.2.2 结果

```
Batch size = 32
Epoch = 1/1 steps = 1 loss = 0.697223
Epoch = 1/1 steps = 20 loss = 0.433323
Epoch = 1/1 steps = 40 loss = 0.354643
```

……

```
Epoch = 1/1 steps = 620 loss = 0.084170
Evaluating:   0%|          | 0/938 [00:00<?, ?it/s]Start evaluating
Evaluating: 100%|██████████| 938/938 [15:18<00:00,  1.02it/s,
F1=0.00000]
Precision:0.000, Recall:0.000, F1-score:0.000
Precision_ignore:0.000, Recall_ignore:0.000, F1-score_ignore:0.000
```

3.2.3 分析

样本不均衡这个问题在下面这一步已经解决了，不需要在前面改。且下面这个方法更好，能保证每个负例在每个epoch中至多只使用一次。

```
neg_batch = tuple(t[0:train_batch_size // 2].to(device) for t in
neg_batch)
```

但是precision、recall还是非常低。这可能是相比较于类别数（97），样本数量较少（9000+）**造成的？也可能是模型**实现得有问题。具体哪里有问题，我也没有找到。

3.3 第三次训练

Max_seq_length = 512

为解决memory装不下所有样本的问题，在加载训练集时，随机留下和正例数量相同的负例。

代码：

```
print('loading data...')
train_features = []
with open(train_feat_dir, 'r') as f:
  for i, line in enumerate(f):
    train_features.append(json.loads(line))
    if (i+1) == 50000:
      pos_features, neg_features = split_data(train_features)
      neg_list = []
      for i in range(len(pos_features)):
        neg_list.append(neg_features[random.randint(0,
len(neg_features) - 1)])
      neg_features = neg_list
      neg_list = []
      train_features = []
    elif (i+1) % 50000 == 0:
      print(i)
      new_pos_features, new_neg_features = split_data(train_features)
```

```
        neg_list = []
        for i in range(len(new_pos_features)):
          neg_list.append(new_neg_features[random.randint(0,
len(new_neg_features) - 1)])
        new_neg_features = neg_list
        pos_features = pos_features + new_pos_features
        neg_features = neg_features + new_neg_features
        neg_list = []
        train_features = []
train_features = []
```

另外，code**中一**处错误的改正：

原本：

```
input_ids, input_mask, segment_ids, relation_multi_label = pos_batch
```

改动后：

```
input_ids, input_mask, segment_ids, relation_multi_label = batch #改动
```

疑惑：这说明原本的code中没有使用neg_batch，那为什么预测结果准确率那么低呢？

Google colab GPU**跑上限了**，这一次没有GPU加速，训练速度较慢，并没有能训练完。

五、测试数据评测

代码：（因为上面的训练没跑完，这个代码没机会运行过，可能存在很多bug)

```
test_feat_dir = '/content/drive/MyDrive/Colab Notebooks/xtzx/nlp4/预训练模型应用案例/dataset/test_cls_data.txt'
test_features = []
with open(test_feat_dir, 'r') as f:
  for i, line in enumerate(f):
    dev_features.append(json.loads(line))
    if i > 20000 and i%10000 == 0:
      print("test load at", i)
rel2id = json.load(open('/content/drive/MyDrive/Colab Notebooks/xtzx/nlp4/预训练模型应用案例/dataset/rel2id.json', 'r'))

id2rel = {}

for key, item in rel2id.items:
  id2rel[item] = key

test_info = []
test_info_dir = '/content/drive/MyDrive/Colab Notebooks/xtzx/nlp4/预训练模型应用案例/dataset/test.json'
with open(test_info_dir, 'r') as f:
  for i, line in enumerate(f):
    dev_features.append(json.loads(line))
    if i > 20000 and i%10000 == 0:
      print("test load at", i)


def to_official(preds, i):

    f = test_info[i]:
    hts = f["label"]
    h_idx = hts['h']
    t_idx = hts['t']
    title = f["title"]

    p = np.argmax(preds)

    res={'title': title,
          'h_idx': h_idx,
```

```
            't_idx': t_idx,
            'r': id2rel[p]}

    return res



def test_pred(model, test_features, device):
    test_dataloader = DataLoader(test_features,
batch_size=predict_batch_size, collate_fn=data_gen,
                                    shuffle=False)
    all_res = []

    model.eval()
    index = 0

    print("Start predicting")
    with tqdm(total=len(test_dataloader), desc='Predicting') as pbar:
        for batch_data in eval_dataloader:
            input_ids, input_mask, segment_ids, _ = batch_data
            input_ids = input_ids.to(device)
            input_mask = input_mask.to(device)
            segment_ids = segment_ids.to(device)

            with torch.no_grad():
                preds = model(input_ids, segment_ids, input_mask)

            preds = preds.cpu().numpy()
            for i in range(preds.shape[0]):
              res = to_official(preds[i], index)
              all_res.append(res)
              index += 1

    return all_res

all_res = test_pred(model, test_features, device)
output_file = '/content/drive/MyDrive/Colab Notebooks/xtzx/nlp4/预训练模
型应用案例/result.json'
with open(output_file, 'w') as w:
  for res in all_res:
    w.write(json.dumps(res, ensure_ascii=False) + '\n')
```

六、模型改进

6.1 其他bert模型

可以考虑使用更大的bert模型，例如'bert-large-uncased-whole-word-masking',
链接 :https://github.com/ewrfcas/DocRED_Bert
(我的电脑上apex装不上，代码运行出错）（下面是作者的结果）

Result in 10 epoch:

Threshold=0.5 in sigmoid

All: Precision:46.336, Recall:78.334, F1-score:58.228

Ignore: Precision:42.544, Recall:76.520, F1-score:54.684

Threshold=0.9 in sigmoid

All: Precision:56.772, Recall:70.718, F1-score:62.981

Ignore: Precision:52.836, Recall:68.814, F1-score:59.776

6.2 RoBERTa模型

可以考虑使用RoBERTa模型替换bert。RoBERTa模型探索了多种预训练技巧，例如动态遮盖、输入格式等，并在大量数据上进行了大量实验，它可能可以实现比bert更好的效果。

6.3 CorefBERT

可以考虑使用CorefBERT
链接 :https://github.com/thunlp/CorefBERT
（这份代码在我的电脑上跑不通，报错，Runtime error）

6.4 其他改进

Adaptive Thresholding and Localized Context Pooling
链接：https://github.com/wzhouad/ATLOP
Edge-oriented Graph：https://github.com/fenchri/edge-oriented-graph

七、反思小结

在这次实验中尝试了使用预训练模型bert来进行长文本关系抽取任务，但是结果不是很理想。
计算机算力的飞跃带给NLP领域的变化是巨大的，使得很多庞大的模型能够进行训练，并取得了不错的效果。
但可惜我的电脑很难带动这些模型。如果可以，希望能在云平台上（BaiduAI）尝试跑动这些模型。
硬件是一大限制（如memory、GPU**等），但是我本身写**code、debug、**配置**环境的能力不足导致**我无法在短**时间内复现他人写好的模型。这是我自身需要不断尝试实践、积攒经验努力的地方。