基于transformer的文本摘要实验

学员信息

- 姓名：纪兴柱
- 学号：231017000028

实验简述

命名实体识别 （NER）是NLP里的一项很基础的任务，旨在定位非结构化文本中提到的命名实体并将其分类为预定义的类别，例如人名、组织、位置、医疗代码、时间表达式、数量、货币价值、百分比等。该实验通过NER更好地认识NLP的相关技术与流程。

实验环境

操作系统：win10虚拟机
python=3.9
torch=1.10.1 (cpu)
transformers=4.18.0
datasets=2.4.0

实验过程

- 分词器

分词器使用预训练的`'distilbert-base-uncased'`。DistilBERT是一种基于BERT（Bidirectional Encoder Representations from Transformers）模型的轻量级版本。它具有较小的模型尺寸和计算资源需求，同时在许多自然语言处理任务上表现良好。

```python
#加载编码器
tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased',
                                          use_fast=True)

print(tokenizer)
```

```
nFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '61930' '--' 'C:\temp\231017000028\src\scripts\named_entity_ identification.py'
PreTrainedTokenizerFast(name_or_path='distilbert-base-uncased', vocab_size=30522, model_max_len=512, is_fast=True, padding_side='right',
'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

- 数据集

数据集使用了'conll2003'， 该数据集是一个常用的英文命名实体识别（Named Entity Recognition，NER）数据集。
在数据处理函数中，首先对每个样本进行分词和标签对齐，然后移除不需要的列，如ID、tokens、pos_tags、chunk_tags和ner_tags。最后返回经过处理后的数据集，用于模型训练或评估。

```python
def get_dataset():

    #加载本地已处理好的数据集
    if load_from_local:
        dataset = load_from_disk('./src/dataset/')
        return dataset

    #远程加载需要预处理
    dataset = load_dataset(path='conll2003')

    print('查看数据样例')
    print(dataset, dataset['train'][0])

    #数据处理函数
    def tokenize_and_align_labels(data):
        #分词
        data_encode = tokenizer.batch_encode_plus(data['tokens'],
                                                  truncation=True,
                                                  is_split_into_words=True)

        data_encode['labels'] = []
        for i in range(len(data['tokens'])):
            label = []
            for word_id in data_encode.word_ids(batch_index=i):
                if word_id is None:
                    label.append(-100)
                else:
                    label.append(data['ner_tags'][i][word_id])

            data_encode['labels'].append(label)

        return data_encode

    dataset = dataset.map(
        tokenize_and_align_labels,
        batched=True,
        batch_size=1000,
        num_proc=1,
        remove_columns=['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'])

    return dataset
```

```
Reusing dataset conll2003 (C:\Users\Administrator\.cache\huggingface\datasets\conll2003\conll2003\1.0.0\9a4d16a94f8674ba3466315300359b0acd891b68b6c874
100%|                                                                                                                  | 3
查看数据样例
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3453
    })
}) {'id': '0', 'tokens': ['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.'], 'pos_tags': [22, 42, 16, 21, 35, 37, 16, 21, 7]
2, 0], 'ner_tags': [3, 0, 7, 0, 0, 0, 7, 0, 0]}
```

- 数据加载器

  通过torch.utils.data.DataLoader创建了一个数据加载器。参数dataset指定了要加载的训练集(dataset['train'])。batch_size设置为8，表示每个批次中有8个样本。collate_fn参数指定了用于处理批次的函数，这里使用了DataCollatorForTokenClassification，它会将批次中的样本整理为模型需要的格式。shuffle=True表示在每个轮次开始时打乱数据顺序，drop_last=True表示如果最后一个批次样本数量不足8个，则丢弃该批次。

```python
#数据加载器
loader = torch.utils.data.DataLoader(
    dataset=dataset['train'],
    batch_size=8,
    collate_fn=DataCollatorForTokenClassification(tokenizer),
    shuffle=True,
    drop_last=True,
)

for i, data in enumerate(loader):
    break

for k, v in data.items():
    print(k, v.shape, v[:2])

len(loader)
```

```
input_ids torch.Size([8, 54]) tensor([[  101, 14824,  1005,  1055,  7794,  7920,  2003,  2025,  2092,  2438,
          2000,  5463,  2021,  1037,  4471,  2013,  2014,  2097,  2022,  3191,
          2041,  2011,  1996, 19938,  1005,  1055,  2882,  1011,  2684, 17508,
          6229,  2386,  2076,  1996,  3116,   102,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0],
        [  101,  7367,  4244,  1010,  5479,  1011,  2039,  2000, 22160,  2197,
          2095,  1010,  2003, 13916,  2000,  2448,  2046,  3587,  1011,  4396,
          2446,  2019,  3489,  9594,  2121,  1999,  1996,  4284,  1011,  4399,
          2007,  2959,  6534,  9530,  5428,  2696, 10337,  2030,  5964,  1011,
         13916,  4386,  3410, 12110, 16273,  2559,  2066,  2014,  2087,  3497,
         16797,  7892,  1012,   102]])
attention_mask torch.Size([8, 54]) 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, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0],
        [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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1]])
labels torch.Size([8, 54]) tensor([[-100,    1,    0,    0,    0,    1,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    1,    2,    2,    0,    0,    0, -100,
         -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
         -100, -100, -100, -100, -100, -100],
        [-100,    1,    1,    0,    0,    0,    0,    0,    1,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    7,    1,    1,    2,
            2,    0,    0,    0,    0,    0,    0,    0,    0,    1,    1,    1,
            2,    0,    0,    0,    0,    7,    0,    1,    2,    0,    0,    0,
            0,    0,    0,    0,    0, -100]])

1755
```

- 下游任务模型

  这个类继承自PreTrainedModel，其中config_class属性设置为PretrainedConfig。

  在**init**方法中，首先加载了预训练的DistilBERT模型(distilbert-base-uncased)作为self.pretrained。然后，定义了一个全连接层self.fc，包含一个dropout层和一个线性层，将输入特征维度768映射到输出类别数9。接下来，通过加载预训练模型的参数，将self.fc的权重初始化为预训练模型的分类器权重。最后，定义了交叉熵损失函数self.criterion。

  在forward方法中，首先将输入数据传递给预训练模型，得到最后一层隐藏状态logits。然后，将logits传递给全连接层self.fc进行分类。如果提供了标签labels，计算交叉熵损失。最后，返回一个字典，包含损失和预测的logits。

```python
#定义下游任务模型
class Model(PreTrainedModel):
    config_class = PretrainedConfig

    def __init__(self, config):
        super().__init__(config)

        self.pretrained = DistilBertModel.from_pretrained(
            'distilbert-base-uncased')

        #9 = len(dataset['train'].features['ner_tags'].feature.names)
        self.fc = torch.nn.Sequential(torch.nn.Dropout(0.1),
                                      torch.nn.Linear(768, 9))

        #加载预训练模型的参数
        parameters = AutoModelForTokenClassification.from_pretrained(
            'distilbert-base-uncased', num_labels=9)
        self.fc[1].load_state_dict(parameters.classifier.state_dict())

        self.criterion = torch.nn.CrossEntropyLoss()

    def forward(self, input_ids, attention_mask, labels=None):
        logits = self.pretrained(input_ids=input_ids,
                                 attention_mask=attention_mask)
        logits = logits.last_hidden_state

        logits = self.fc(logits)

        loss = None
        if labels is not None:
            loss = self.criterion(logits.flatten(end_dim=1), labels.flatten())

        return {'loss': loss, 'logits': logits}
```

- train

  使用AdamW优化器和线性学习率调度器进行模型的参数优化和学习率更新。在每个批次中，将数据移动到设备上，通过模型进行前向传播并计算损失。然后，执行反向传播和梯度裁剪，更新模型参数和学习率。最后，将模型移回CPU。

```python
#训练
def train():
    optimizer = AdamW(model.parameters(), lr=2e-5)
    scheduler = get_scheduler(name='linear',
                              num_warmup_steps=0,
                              num_training_steps=len(loader),
                              optimizer=optimizer)

    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    model.train()
    model.to(device)
    for i, data in enumerate(loader):
        for k in data.keys():
            data[k] = data[k].to(device)

        out = model(**data)
        loss = out['loss']

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
        scheduler.step()

        optimizer.zero_grad()
        model.zero_grad()
        if i % 50 == 0:
            labels = []
            outs = []
            out = out['logits'].argmax(dim=2)
            for j in range(8):
                select = data['attention_mask'][j] == 1
                labels.append(data['labels'][j][select][1:-1])
                outs.append(out[j][select][1:-1])

            #计算正确率
            labels = torch.cat(labels)
            outs = torch.cat(outs)
            accuracy = (labels == outs).sum().item() / len(labels)

            lr = optimizer.state_dict()['param_groups'][0]['lr']

            print(i, loss.item(), accuracy, lr)

    model.to('cpu')
```

训练过程中会周期性地打印训练信息，包括轮次，损失、准确率和学习率。经过将近两千次的训练，正确率达99.16%。

```
  warnings.warn(
0 2.399038553237915 0.024390243902439025 1.998860398860399e-05
50 0.5870899558067322 0.8269230769230769 1.941880341880342e-05
100 0.51480633020401 0.8372093023255814 1.8849002849002852e-05
150 0.13835808634757996 0.9583333333333334 1.827920227920228e-05
200 0.13117648661136627 0.9669421487603306 1.770940170940171e-05
250 0.23385891318321228 0.94 1.713960113960114e-05
300 0.1472039669752121 0.9512195121951219 1.6569800569800573e-05
350 0.19199655950069427 0.9505494505494505 1.6000000000000003e-05
400 0.0964977890253067 0.9759036144578314 1.5430199430199432e-05
450 0.5718852877616882 0.8689655172413793 1.4860398860398862e-05
500 0.14928022027015686 0.9669421487603306 1.4290598290598293e-05
550 0.04728260636329651 0.9913793103448276 1.3720797720797722e-05
600 0.06350374221801758 0.9777777777777777 1.3150997150997152e-05
650 0.17678901553153992 0.9712230215827338 1.2581196581196581e-05
700 0.12549585103988647 0.96875 1.2011396011396012e-05
750 0.06563395261764526 0.9820627802690582 1.1441595441595444e-05
800 0.24276818335056305 0.917910447761194 1.0871794871794871e-05
850 0.09486348181962967 0.9707602339181286 1.0301994301994303e-05
900 0.08860410004854202 0.9617224880382775 9.732193732193734e-06
950 0.08840496093034744 0.96875 9.162393162393163e-06
1000 0.2855250835418701 0.9044117647058824 8.592592592592593e-06
1050 0.06121617928147316 0.9864864864864865 8.022792022792024e-06
1100 0.025660444051027298 1.0 7.452991452991454e-06
1150 0.16198216378688812 0.9543147208121827 6.883190883190884e-06
1200 0.2566322386264801 0.9340659340659341 6.313390313390314e-06
...
1600 0.019120031967759132 0.9933333333333333 1.7549857549857553e-06
1650 0.042408760637044491 1.0 1.1851851851851854e-06
1700 0.095241978764534 0.9797297297297297 6.153846153846155e-07
1750 0.030119173228740692 0.9915966386554622 4.5584045584045586e-08
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

实验结论

- 测试函数

  测试函数将模型设置为评估模式，并使用测试数据加载器加载数据。在测试循环中，对于每个批次，通过模型进行前向传播并获取预测的标签索引，然后将预测值和标签值添加到列表中。在一定的间隔内打印进度信息，并限制测试批次的数量。最后，将标签和预测值连接起来，并计算预测准确率并打印出来。

```python
#测试
def test():
    model.eval()

    #数据加载器
    loader_test = torch.utils.data.DataLoader(
        dataset=dataset['test'],
        batch_size=16,
        collate_fn=DataCollatorForTokenClassification(tokenizer),
        shuffle=True,
        drop_last=True,
    )

    labels = []
    outs = []
    for i, data in enumerate(loader_test):
        #计算
        with torch.no_grad():
            out = model(**data)

        out = out['logits'].argmax(dim=2)

        for j in range(16):
            select = data['attention_mask'][j] == 1
            labels.append(data['labels'][j][select][1:-1])
            outs.append(out[j][select][1:-1])

        if i % 10 == 0:
            print(i)

        if i == 50:
            break

    #计算正确率
    labels = torch.cat(labels)
    outs = torch.cat(outs)

    print((labels == outs).sum().item() / len(labels))
```

```python
#训练前测试
test()

#训练
train()

#训练前测试
test()
```

- 训练前测试

```
0
10
20
30
40
50
0.016053162099604272
```

- 训练后测试

```
0
10
20
30
40
50
0.973296013314169
```

实验总结

在实验过程中遇到很多问题，包括为环境搭建，数据集的下载，以及模型函数参数设置等等。通过解决以上问题对NLP的处理领域和处理逻辑有了初步认识，感谢老师和热心的同学们。