#

1. 原理概述

1.1 提示学习

提示学习（Prompt Learning）是自然语言处理（NLP）领域中的一种创新方法，特别是在利用大规模预训练语言模型（如 BERT、GPT-3 等）进行各种下游任务时的一种技术。它模仿人类向模型提出问题的方式，通过设计合适的“提示”——一种特定形式的输入，使得模型能更好地理解并执行任务。参考论文：The Power of Scale for Parameter-Efficient Prompt Tuning。

1.2 硬提示学习

硬提示学习通常涉及将预定义的标记（通常是词或短语）添加到模型的输入中。这些标记被称为“硬提示”，因为它们是固定不变的文本片段。这种方法简单直接，类似于提问答题的形式。

例如，我们有一条评论：“电影剧情紧凑，非常精彩。”为了使用硬提示学习进行情感分析，我们可能会人为设计一个提示词或短语，比如“情感是正面的吗？”然后将这个硬提示与评论一起构成输入语句：“情感是正面的吗？电影剧情紧凑，非常精彩。”这种硬提示尝试指导模型理解其任务是判断评论的情感倾向。模型之后会根据这个包含硬提示的输入来预测情感。

- 原始输入：`[CLS] 电影剧情紧凑，非常精彩。 [SEP]`
- 添加硬提示：`[CLS] 情感是正面的吗？ 电影剧情紧凑，非常精彩。 [SEP]`

硬提示学习由于其简单性，在一些任务中可以获得不错的性能，尤其是当我们能够设计出一个非常有效的硬提示，但是这往往包含着运气成分。

1.3 软提示学习

与硬提示学习不同，软提示学习不涉及使用预设的文本片段作为输入。相反，它会在模型接收到实际输入之前加入一串可训练的嵌入向量。这些嵌入向量是模型参数的一部分，并且会在训练过程中根据下游任务的需求进行优化和更新。因此，软提示是“软”的，因为它们是可以调整和学习的，不是固定的文本。

例如，在进行情感分析任务时，不是在“电影剧情紧凑，非常精彩。”之前加上“情感是正面的吗？”这样的硬提示，我们先得到原始输入的嵌入层输出结果，然后加上一组软提示嵌入向量，作为新的嵌入层输出结果。这些软提示嵌入向量在训练过程中自适应地学习，修改它们的值以提高模型识别正面或负面情感的能力。

$$
\begin{array}{rl}
\text{原始输入：} & [CLS] \quad \text{电影剧情紧凑，非常精彩。} \quad [SEP] \\
\text{原始输入的嵌入表示：} & \vec{x}_{cls}, \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m, \vec{x}_{sep} \\
\text{添加软提示嵌入：} & \overbrace{\vec{e}_1, \vec{e}_2, \vec{e}_3, \ldots, \vec{e}_n}^{\text{Soft Prompt}}, \vec{x}_{cls}, \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m, \vec{x}_{sep}
\end{array}
$$

在实际应用中，软提示通常表现出更大的灵活性和效率，因为它们允许模型在不同任务和领域间灵活转换，并在固定模型参数的情况下进行微调。

2.实验过程

2.1 数据准备

2.1.1 定义配置

```
cfg = edict({
    'name': 'movie review',
    'pre_trained': True,
    'num_classes': 2,
    'batch_size': 16,
    'epoch_size': 3,
    'weight_decay': 3e-5,
    'data_path': "./data/prompt tuning/data/",
    'checkpoint_path': 'soft-prompt.pth',
    'device_name':"cuda" if torch.cuda.is_available() else "cpu",
    'gpt2_model':'./gpt2',
    'prompt_len':10,
    'max_len' : 100,
    'classes':[['positive'],['negative']],
    'split': 0.8
})
```

2.1.2 定义数据集

实验使用的评论数据集包含评论文本和对应的情感标签（正面或负面）。数据集被划分为训练集和测试集。

`MovieDataset` 类用于处理影评数据集，通过指定的根目录加载数据，检查数据有效性，并将正面和负面的影评分别读取和预处理成模型可接受的格式。随后，该类根据设定的比例将数据集分割为训练集和测试集，并提供方法返回这些数据集对象。整个过程中，类还计算了句子的长度信息，并确保数据的一致性和有效性。

```
class MovieDataset:
    '''
    影评数据集
    '''
    def __init__(self, root_dir, maxlen, split):
        ... ...
    def read_data(self, filePath):
        ... ...
    def process_data(self, data_set, tag):
        ... ...
    def split_dataset(self, split):
        ... ...
    def get_dict_len(self):
        ... ...
    def train_dataset(self):
        ... ...
    def test_dataset(self):
        ... ...
```

`__init__` 方法初始化 `MovieDataset` 类实例，接受影评数据目录路径、句子最大长度和训练/评估比例作为参数。先检查路径的有效性，确保目录中包含两个文件（正面和负面影评）。然后读取文件内容，统计句子长度信息，将数据预处理为模型可接受的格式，并根据设定比例将数据集分割为训练集和评估集。

```
def __init__(self, root_dir, maxlen, split):
    '''
        input:
            root_dir: 影评数据目录
            maxlen: 设置句子最大长度
            split: 设置数据集中训练/评估的比例
        '''
    self.path = root_dir
    self.files = []

    self.doConvert = False

    mypath = Path(self.path)
```

```
    if not mypath.exists() or not mypath.is_dir():
        print("please check the root_dir!")
        raise ValueError

        # 在数据目录中找到文件
        for root,_,filename in os.walk(self.path):
            for each in filename:
                self.files.append(os.path.join(root,each))
                break

                # 确认是否为两个文件.neg与.pos
                if len(self.files) != 2:
                    print("There are {} files in the
root_dir".format(len(self.files)))
                    raise ValueError

                    # 读取数据
                    self.word_num = 0
                    self.maxlen = 0
                    self.minlen = float("inf")
                    self.maxlen = float("-inf")
                    self.Pos = []
                    self.Neg = []
                    self.sentences = []
                    self.isShuffle = True

                    for filename in self.files:
                        f = codecs.open(filename, 'r')
                        ff = f.read()
                        file_object = codecs.open(filename, 'w',
'utf-8')
                        file_object.write(ff)
                        self.read_data(filename)


                        self.Pos = self.process_data(self.Pos,
cfg.classes[0][0])
                        self.Neg = self.process_data(self.Neg,
cfg.classes[1][0])

                        #self.text2vec(maxlen=maxlen)
                        self.split_dataset(split=split)
```

`read_data` 方法用于读取指定文件路径中的影评数据。它逐行读取文件内容，并对每行进行预处理。 然后将处理后的句子按正面或负面分类，分别添加到 `self.Pos` 或 `self.Neg` 列表中，并记录每个句子的单词数量。

```
def read_data(self, filePath):
    with open(filePath,'r') as f:
        for sentence in f.readlines():
            sentence = sentence.replace('\n','')
                …… # 省略
                .replace('%','')
            if sentence:
                self.word_num += len(sentence.split(' '))
                self.maxlen = max(self.maxlen, len(sentence.split('
')))
                self.minlen = min(self.minlen, len(sentence.split('
')))
                if 'pos' in filePath:
                    self.Pos.append([sentence,
self.feelMap['pos']])
                else:
                    self.Neg.append([sentence,
self.feelMap['neg']])
```

`process_data` 方法用于将影评数据集中的句子预处理为模型可接受的格式。对于每个句子，首先使用 `tokenizer` 对其进行编码，生成输入张量并添加特殊标记，同时设置目标标签。接着，将编码结果中的张量维度压缩，并保存句子的原文本、输入ID、标签、注意力掩码和目标标签。计算句子的实际长度并更新注意力掩码，以确保模型处理时正确关注特定部分。最后，将处理后的结果添加到返回列表中，并返回该列表。

```
def process_data(self, data_set, tag):
    ret = []
    for line in data_set:
        res = tokenizer(
            line.strip('\n'),
            return_tensors="pt",
            text_target=tag,
            padding='max_length',
            max_length=cfg.max_len + cfg.prompt_len,
            add_special_tokens=True,
        )
        res['text'] = line
        res['input_ids'] = res['input_ids'].squeeze(0)
```

```
        res['labels'] = res['labels'].squeeze(0)
        res['attention_mask'] = res['attention_mask'].squeeze(0)
        res['answer'] = tag
        res['len'] = res['attention_mask'].sum()
        res['attention_mask'][res['len']:res['len'] +
cfg.prompt_len] = 1
        ret.append(res)
    return ret
```

`split_dataset` 方法用于将影评数据集按设定比例分割为训练集和测试集。首先，计算正面和负面影评中训练集所需的样本数量，并确定分割的次数。然后，将正面和负面影评按计算的数量进行分块存储在临时列表中。接着，选择其中一个分块作为测试集，其余的作为训练集，并将这些分块组合成最终的训练集和测试集。最后，对训练集进行随机打乱，以确保训练过程中数据的随机性。

```
def split_dataset(self, split):
    '''
    分割为训练集与测试集

    '''
    trunk_pos_size = math.ceil((1-split)*len(self.Pos))
    trunk_neg_size = math.ceil((1-split)*len(self.Neg))
    trunk_num = int(1/(1-split))
    pos_temp=list()
    neg_temp=list()
    for index in range(trunk_num):
        pos_temp.append(self.Pos[index*trunk_pos_size:
(index+1)*trunk_pos_size])
        neg_temp.append(self.Neg[index*trunk_neg_size:
(index+1)*trunk_neg_size])
    self.test = pos_temp.pop(2)+neg_temp.pop(2)
    self.train = [i for item in pos_temp+neg_temp for i in item]

    random.shuffle(self.train)
    # random.shuffle(self.test)
```

最后用`CustomDataset`类的构造函数，列表转数据集，方便后续训练。

```
def train_dataset(self):
    return CustomDataset(self.train)

def test_dataset(self):
    return CustomDataset(self.test)
```

MovieDataset 会根据软提示和硬提示有变化

2.2 硬提示学习实验

2.2.1 设计硬提示

为情感分析任务设计硬提示，例如`is it positive or negative`。

硬提示比较简单，在原有的句子上面加一行提示即可，故修改`class MovieDataset`

```
prompt = f"Is the sentiment positive or negative"  # 硬提示
res = tokenizer(
    (prompt + line).strip('\n'),
    return_tensors="pt",
    text_target=tag,
    padding='max_length',
    max_length=cfg.max_len + cfg.prompt_len,
    add_special_tokens=True,
    )
```

2.2.2 构建输入数据

将硬提示与原始评论文本结合，构建测试集。

```
instance = MovieDataset(cfg.data_path, maxlen=cfg.max_len, split =
cfg.split)
test_dataset = instance.test_dataset()
```

2.2.3 模型测试

在测试集上评估模型性能。 提取 logits 中最后一个 token 的输出，计算概率分布，确定生成词

```
def test():
    cfg.batch_size = 1
    data_loader = DataLoader(test_dataset, 
batch_size=cfg.batch_size, )
    total = 0
    correct = 0
    # 预先编码 'positive' 和 'negative' 以减少循环中的计算
    positive_token_id = tokenizer.encode('positive')[0]
    negative_token_id = tokenizer.encode('negative')[0]
    for batch in data_loader:
        inputs, labels = batch['input_ids'].to(cfg.device_name), 
batch['labels'].to(cfg.device_name)
        output = model(inputs, labels=labels)
        logits = output.logits[:, -1, :]  # 取最后一个token的输出
        # 选取最后一个提示词对应的生成词
        AnswerPlace = (batch["len"] + cfg.prompt_len - 
1).to(cfg.device_name)
        probabilities = 
torch.nn.functional.softmax(output.logits[:, :, :], dim=-1)
        answer_pb = 
probabilities[torch.arange(probabilities.shape[0]), AnswerPlace]
        predicted_tokens = [tokenizer.decode(s).strip() for s
                            in torch.argmax(answer_pb, dim=-1)]
        batch['result'] = predicted_tokens
        for i in range(cfg.batch_size):
            ……打印结果
```

因为未进行训练效果不佳,考虑使用猜测下一个是`positive` or `negative`的概率

```
# 获取 'positive' 和 'negative' token 的概率
        probabilities = torch.nn.functional.softmax(logits, dim=-1)
        positive_probs = probabilities[:, positive_token_id]
        negative_probs = probabilities[:, negative_token_id]
        # 计算每个样本的结果
        results = (positive_probs > negative_probs).long()
        batch['result'] = [cfg.classes[0][0] if result.item() == 1
else cfg.classes[1][0] for result in results]
```

2.3 软提示学习实验

2.3.1 初始化软提示嵌入向量

随机初始化一组嵌入向量，作为软提示。

```
def create_soft_prompt(length):
    prompt = \
        tokenizer('is it negative', max_length=length,
padding='max_length',
                  return_tensors='np')[
            'input_ids']
    prompt = numpy.array([prompt, ])
    initial_sp =
model.transformer.wte(torch.from_numpy(prompt).to(cfg.device_name))
    sp = torch.nn.Parameter(initial_sp[0], requires_grad=True)
    return sp
soft_prompt = create_soft_prompt(cfg.prompt_len)
```

2.3.2 构建输入数据

定义一个软提示嵌入层

```
prompt_embeddings = soft_prompt.to(cfg.device_name)
```

将原始评论文本的嵌入层输出结果与软提示嵌入向量结合，构建新的嵌入层输出结果。

```
def forward(self, batch):
        input_ids = batch["input_ids"].to(cfg.device_name)
        target_ids = batch["labels"].to(cfg.device_name)
```

```
        # sentence_embeddings = model.transformer.wte(input_ids)
        sentence_embeddings =
model.transformer.wte(input_ids).to(cfg.device_name)

        # 生成 soft prompt embeddings
        prompt_embeddings = soft_prompt.to(cfg.device_name)

        # 将 soft prompt embeddings 插入到输入的结尾
        for i in range(input_ids.shape[0]):
            l = batch["len"][i]
            sentence_embeddings[i, l:l + cfg.prompt_len] =
prompt_embeddings
        # 执行前向传递
        output = model(
            inputs_embeds=sentence_embeddings,  # labels=target_ids

 attention_mask=batch["attention_mask"].to(cfg.device_name)
        )
        # 选取最后一个提示词对应的生成词
        AnswerPlace = (batch["len"] + cfg.prompt_len -
1).to(cfg.device_name)
        probabilities =
torch.nn.functional.softmax(output.logits[:, :, :], dim=-1)
        answer_pb =
probabilities[torch.arange(probabilities.shape[0]), AnswerPlace]
        predicted_tokens = [tokenizer.decode(s).strip() for s in
torch.argmax(answer_pb, dim=-1)]
        batch['result'] = predicted_tokens

        # 计算损失
        answer_logits =
output.logits[torch.arange(probabilities.shape[0]), AnswerPlace, :]
        loss = loss_fn(answer_logits, target_ids[:, 0])

        return answer_logits, loss
```

2.3.3 模型训练

注意只优化soft prompt

```
# 只优化 soft prompt 的参数
optimizer = AdamW([soft_prompt], lr=0.1)
loss_fn = torch.nn.CrossEntropyLoss()
```

2.3.4 模型测试

在测试集上评估模型性能。

3.实验结果

3.1 硬编码实验结果

首先采用的是预测下一个词的结果，但是由于硬编码没有训练过程，效果并不好。

```
correct: 0/1 = 0.0
text: director david fincher and writer david koepp can't sustain it .
result/answer: The/negative

correct: 0/2 = 0.0
text: after sitting through this sloppy , made-for-movie comedy special , it makes me wonder if lawrence hates criticism so
much that he refuses to evaluate his own work .
result/answer: The/negative

correct: 0/3 = 0.0
text: a profoundly stupid affair , populating its hackneyed and meanspirited storyline with cardboard characters and perform
ers who value cash above credibility .
result/answer: The/negative

correct: 0/4 = 0.0
text: a string of rehashed sight gags based in insipid vulgarity .
result/answer: The/negative

correct: 0/5 = 0.0
text: martin and barbara are complex characters -- sometimes tender , sometimes angry -- and the delicate performances by sv
en wollter and viveka seldahl make their hopes and frustrations vivid .
result/answer: The/positive

correct: 0/6 = 0.0
text: alternating between facetious comic parody and pulp melodrama , this smart-aleck movie . . . tosses around some intrig
uing questions about the difference between human and android life .
result/answer: The/positive

correct: 0/7 = 0.0
text: an artsploitation movie with too much exploitation and too little art .
result/answer: The/negative

correct: 0/8 = 0.0
text: the sort of film that makes me miss hitchcock , but also feel optimistic that there's hope for popular cinema yet .
result/answer: The/positive

correct: 0/9 = 0.0
text: woody allen used to ridicule movies like hollywood ending . now he makes them .
result/answer: The/negative

correct: 0/10 = 0.0
text: directed by kevin bray , whose crisp framing , edgy camera work , and wholesale ineptitude with acting , tone and pace
```

然后尝试比较下一个词是`positive`和`negative`概率，结果基本上和随机一样。

```
text: kinnear gives a tremendous performance .
probability of 'positive': 5.478357212318485e-10
probability of 'negative': 2.735131010567215 5e-10
correct label: positive
```

```
correct: 1/1 = 1.0
text: director david fincher and writer david koepp can't sustain it .
probability of 'positive': 5.371085243233154e-10
probability of 'negative': 2.6735461067239896e-10
correct label: negative
text: director david fincher and writer david koepp can't sustain it .
result/answer: positive/negative
```

```
correct: 81/157 = 0.5159235668789809
text: a gorgeous , somnolent show that is splendidly mummified and thoroughly unsurprising .
result/answer: positive/negative

correct: 81/158 = 0.5126582278481012
text: though the aboriginal aspect lends the ending an extraordinary poignancy , and the story itself could be played out in
any working class community in the nation .
result/answer: positive/positive

correct: 82/159 = 0.5157232704402516
text: a compelling motion picture that illustrates an american tragedy .
result/answer: positive/positive

correct: 83/160 = 0.51875
text: i would have preferred a transfer down the hall to mr . holland's class for the music , or to robin williams's lecture
so i could listen to a teacher with humor , passion , and verve .
result/answer: positive/negative

correct: 83/161 = 0.515527950310559
text: off the hook is overlong and not well-acted , but credit writer-producer-director adam watstein with finishing it at a
ll .
result/answer: positive/negative

correct: 83/162 = 0.5123456790123457
text: a good-natured ensemble comedy that tries hard to make the most of a bumper cast , but never quite gets off the ground
```

3.2 软编码

测试结果1：**batch size = 15**

训练过程

```
Epoch 1/3:   0%|          | 1/533 [00:18<2:40:31, 18.10s/it]
Epoch: 0, Loss: 20.74989128112793
Epoch 1/3:   0%|          | 2/533 [00:33<2:26:06, 16.51s/it]
Epoch: 0, Loss: 16.657255172729492
Epoch 1/3:   1%|          | 3/533 [00:47<2:13:43, 15.14s/it]
Epoch: 0, Loss: 11.952482223510742
Epoch 1/3:   1%|          | 4/533 [01:01<2:10:12, 14.77s/it]
Epoch: 0, Loss: 7.702881336212158
Epoch 1/3:   1%|          | 5/533 [01:14<2:05:17, 14.24s/it]
Epoch: 0, Loss: 3.544212818145752
Epoch 1/3:   1%|          | 6/533 [01:28<2:02:51, 13.99s/it]
Epoch: 0, Loss: 1.1515787839889526
Epoch 1/3:   1%||         | 7/533 [01:42<2:04:38, 14.22s/it]
Epoch: 0, Loss: 0.8505994081497192
Epoch 1/3:   2%||         | 8/533 [01:56<2:04:05, 14.18s/it]
```

```
result/answer: negative/negative

correct: 1957/2666 = 0.7340585146286571
```

根据gpt2的词嵌入层，把软编码转回硬编码查看

```
Extracted Hard Prompts: [' Negative', ' Negative', 'itude', '76561', 'eals', 'umbnails', 'ultimate', 'ert', ' eur
ozone', ' is']
```

测试结果2： batch size =60

```
text: though it is by no means his best work , laissez-passer is a distinguished and distinctive effort by a bona-fide master , a
fascinating film replete with rewards to be had by all willing to make the effort to reap them .
result/answer: positive/positive

correct: 2242/2666 = 0.840960240060015
```

```
Extracted Hard Prompts: [' externalToEVAOnly', ' Negative', 'Newsletter', 'rongh', 'Interstitial', ' eagle', 'reb', 'amon', '%"',
' pro']
```

4.参考资料

SoftPrompting/SoftPrompt-Translation.ipynb at main · 11AbhijithROY/SoftPrompting (github.com)

Hugging Face中GPT2模型应用代码 - 知乎 (zhihu.com)

zejunwang1/gpt2classifier: 基于中文 GPT2 预训练模型的文本分类微调 (github.com)

Prompt-Tuning/prompt_tuning.py at main · advin4603/Prompt-Tuning (github.com)