【NLP作业-05】Seq2Seq模型文本生成

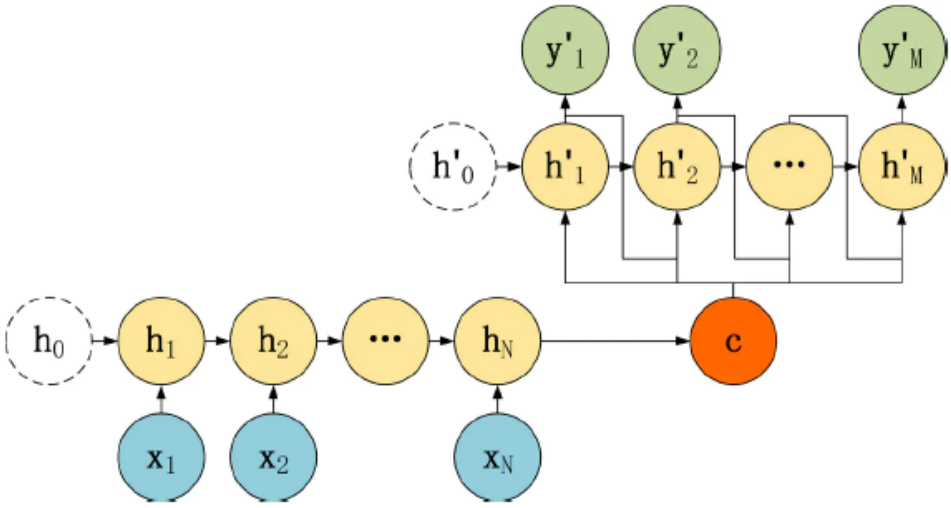
杨思捷　ZY2103533

1. 引言

本作业通过RNN构建了一个编码器和解码器，介绍了注意力机制，但是最终没有实现。最后通过输入文本至Seq2Seq模型生成了一些有趣的文本。

2. Seq2Seq

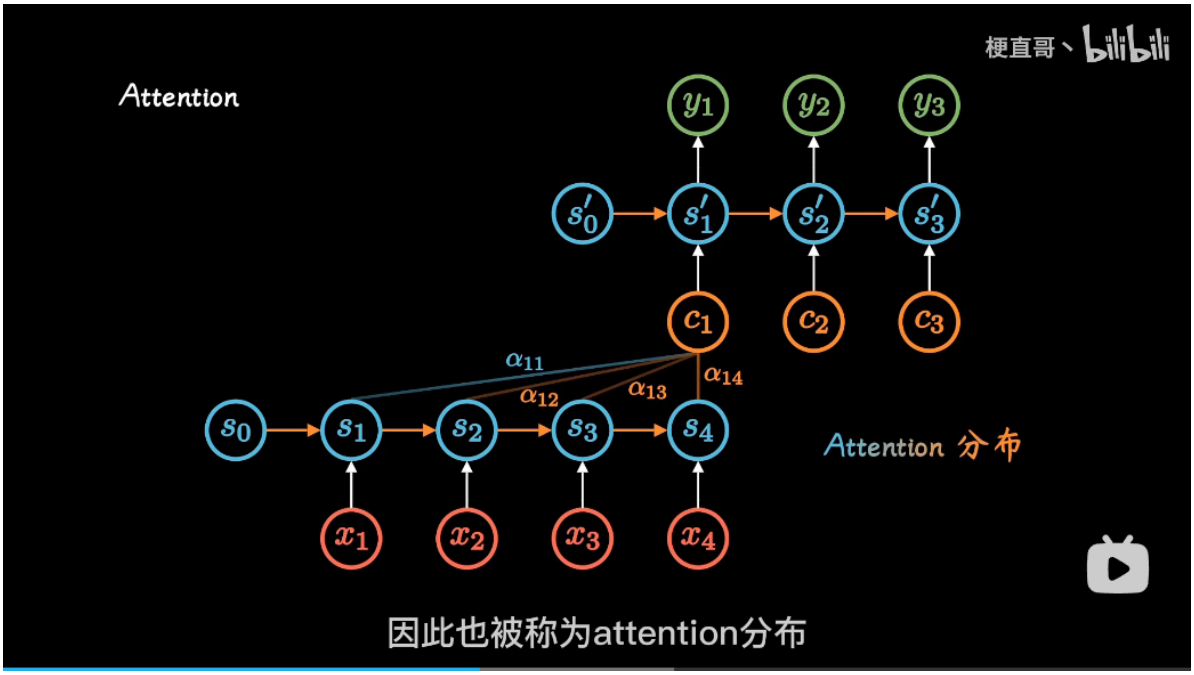
Seq2Seq 是一个Encoder–Decoder 结构的网络，它的输入是一个序列，输出也是一个序列。基本思想就是利用两个RNN，一个RNN作为Encoder，另一个RNN作为Decoder。 Encoder 中将一个可变长度的信号序列变为固定长度的向量表达，Decoder 将这个固定长度的向量变成可变长度的目标的信号序列。



在 Seq2Seq 模型，Encoder 总是将源句子的所有信息编码到一个固定长度的上下文向量c中，然后在 Decoder 解码的过程中向量 c 都是不变的。现在引入注意力机制。

注意力机制，是一种将模型的注意力放在当前翻译单词上的一种机制。例如翻译 "I have a cat"，翻译到 "我" 时，要将注意力放在源句子的 "I" 上，翻译到 "猫" 时要将注意力放在源句子的 "cat" 上。使用了 Attention 后，Decoder 的输入就不是固定的上下文向量 c 了，而是会根据当前翻译的信息，计算当前的 c。

示意图如下，通过对RNN的状态进行加权得到不同的上下文向量，由于权重不同所以为



Attention改进的问题：

- 对于比较长的句子，很难用一个定长的向量 c 完全表示其意义。
- RNN 存在长序列梯度消失的问题，只使用最后一个神经元得到的向量 c 效果不理想。
- 与人类的注意力方式不同，即人类在阅读文章的时候，会把注意力放在当前的句子上。

下面是通过Seq2Seq生成文本的代码实践

3. 代码实现

下面是通过Seq2Seq生成文本的代码实践

3.1. 文本预处理

首先需要进行预处理，将原始文本处理为断好句的列表，特殊符号仅保留逗号，核心代码如下和作业4的代码类似，不同的是构建 `wordTable` 的时候需要加入文本开始、文本结束以及占位符。具体的代码参见 `data_preprocess.py` 和 `utils.py`

```
word2id_dict = {
    ", ":0,
    "<PAD>":1,
    "<BOS>":2,
    "<EOS>":3,
}
```

预处理得到 `corpus_onehot.txt` 以及两个词表相关的json文件。

3.2. 构建数据集

语料库的One-hot表示结果为

```
4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,0,25,26,26,27,0,28,29
```

而Seq2Seq需要的数据格式可以描述如下：

输入第i句文本的One-hot，期望的输出为第i+1句文本的One-hot

因而数据集构建代码如下：

```
class BS_Seq2Seq_Data(Dataset):
    def __init__(self,corpus_onehotL):
        self.corpus_onehot_LL = []
        self.max_len = 131+2
        for corpus_onehot in corpus_onehotL:
            if corpus_onehot.strip() == "":
                continue
            data_floatL = [int(i) for i in corpus_onehot.strip().split(",")]
            self.corpus_onehot_LL.append(data_floatL)



    def __getitem__(self,index):
        inputs = self.corpus_onehot_LL[index]
        targets = self.corpus_onehot_LL[index+1]
        return (inputs, targets)

    def __len__(self):
        return len(self.corpus_onehot_LL)-1

    def collate_fn(self, examples):
        # "1": "<PAD>",
        # "2": "<BOS>",
        # "3": "<EOS>",
        inputsL = []
        targetsL = []
        for inputs,targets in examples:
            inputsL.append( inputs + [1] * (self.max_len-len(inputs)))
            targetsL.append([2]+ targets + [3] + [1]*(self.max_len-2-
len(targets)))

        inputs = torch.tensor(inputsL)
        targets = torch.tensor(targetsL)
        return inputs, targets
```

3.3. 编码器和解码器实现

编码器和解码器的代码实现如下：

```
class Encoder(nn.Module):
    def __init__(self,encoder_embedding_num,encoder_hidden_num,corpus_len):
        super().__init__()
        self.embedding = nn.Embedding(corpus_len,encoder_embedding_num)
        self.lstm =
nn.LSTM(encoder_embedding_num,encoder_hidden_num,batch_first=True)

    def forward(self,en_index):
        en_embedding = self.embedding(en_index)
        _,encoder_hidden =self.lstm(en_embedding)
        return encoder_hidden

class Decoder(nn.Module):
    def __init__(self,decoder_embedding_num,decoder_hidden_num,corpus_len):
        super().__init__()
```

```
        self.embedding = nn.Embedding(corpus_len,decoder_embedding_num)
        self.lstm =
nn.LSTM(decoder_embedding_num,decoder_hidden_num,batch_first=True)

    def forward(self,decoder_input,hidden):
        embedding = self.embedding(decoder_input)
        decoder_output,decoder_hidden = self.lstm(embedding,hidden)
        return decoder_output,decoder_hidden
```

3.4. 训练

定义`Seq2Seq`模型

```
Class Seq2Seq(nn.Module):
    def
__init__(self,encoder_embedding_num,encoder_hidden_num,decoder_embedding_num,dec
oder_hidden_num,corpus_len):
        super().__init__()
        self.encoder =
Encoder(encoder_embedding_num,encoder_hidden_num,corpus_len)
        self.decoder =
Decoder(decoder_embedding_num,decoder_hidden_num,corpus_len)
        self.classifier = nn.Linear(decoder_hidden_num,corpus_len)
```

设当前的语句为`inputs_tensor`下一句话是`targets_tensor`

```
inputs_tensor = 30,31,32,0,33,34,35
targets_tensor = 36,37,0,38,39,0,40,41,42
```

则经过数据集的处理后会变为下面的形式

```
# "1": "<PAD>",
# "2": "<BOS>",
# "3": "<EOS>",
inputs_tensor = 30,31,32,0,33,34,35,1,1……,1
targets_tensor = 2,36,37,0,38,39,0,40,41,42,3,1,……, 1
```

`inputs_tensor`输入编码器后得到RNN的输出

```
encoder_hidden = self.encoder(inputs_tensor)
```

`decoder_input`和编码器的RNN的隐层输出`encoder_hidden`输入解码器获得预测

```
decoder_input = targets_tensor[:,:-1]
label = targets_tensor[:,1:]
decoder_output,_ = self.decoder(decoder_input,encoder_hidden)
```

并通过交叉信息熵计算损失函数

```
pre = self.classifier(decoder_output)
loss = self.cross_loss(pre.reshape(-1,pre.shape[-1]),label.reshape(-1))
```

训练部分的代码参见`bs_seq2seq_train.py`

3.5. 预测

预测部分和训练过程类似，也是先输入当前的语句得到编码器的隐层输出

```
encoder_hidden = self.encoder(inputs_tensor)
```

然后将语句起始`<BOS>`占位符和编码器的隐层输出输入解码器获得预测

```
decoder_output,decoder_hidden = self.decoder(decoder_input,decoder_hidden)
pre = self.classifier(decoder_output)
```

持续预测直到输出终止占位符或者达到最大的长度

```
while True:
    decoder_output,decoder_hidden = self.decoder(decoder_input,decoder_hidden)
    pre = self.classifier(decoder_output)

    w_index = int(torch.argmax(pre,dim=-1))

    # "3": "<EOS>",
    if w_index == 3 or len(result) > :
        break

    result.append(w_index)
    decoder_input = torch.tensor([[w_index]])
```

最后再通过词表将One-hot编码转化为文本。

测试的完整代码参见`bs_seq2seq_predict.py`

4. 结果分析

训练的过程好像有点慢。。。渣笔记本用1050ti差不多5min一个epoch。训练了20个epoch的结果如下

随机选取了几个句子发现输出好像都差不多，可能和训练的样本有关，检查训练集后发现，3W句话中，约有十分之一都是以张无忌开头的，所有训练输出的结果大多以张无忌开头。加上算力有限，网络结构比较简单，所有Seq2Seq的结果不够理想，加入注意力机制可能会更好一些。

- 句子1

输入：这少女姓郭，单名一个襄字，乃大侠郭靖和女侠黄蓉的次女，有个外号叫做小东邪

理想输出：她一驴一剑，只身漫游，原想排遣心中愁闷，岂知酒入愁肠固然愁上加愁，而名山独游，一般的也是愁闷徒增

预测输出：张无忌道我是为甚么

- 句子2

输入: 张无忌痴痴呆呆，只想着小姐的声音笑貌，但觉便是她恶狠狠挥鞭打狗神态，也是说不出的娇媚可爱

理想输出: 有心想自行到后院去，远远瞧她一眼也好，听她向别人说一句话也好，但乔福叮嘱了好几次，若非主人呼唤，决不可走进中门以内，否则必为猛犬所噬

预测输出: 张无忌道谢天谢地，幸好我在你身上先堆了些树枝石头

- 句子3

输入: 赵敏道嗯，你怕不怕痛

理想输出: 那人道小人只怕死，不怕痛

预测输出: 张无忌道我正要对你变心，倘若你再不知道，我的心腹你也不知道

5. 参考资料

https://www.bilibili.com/video/BV1z5411f7Bm

https://www.bilibili.com/video/BV1hf4y1u7ez

https://github.com/shouxieai/seq2seq_translation