RNN实验报告

解释RNN ， LSTM，GRU模型

RNN

RNN 是一种用于处理序列数据的神经网络，它的特点是每个时间步的输出不仅依赖于当前输入，还依赖于前一时刻的输出。通过这种方式，RNN 能够在一定程度上捕捉序列中的时间依赖关系，适合处理如语音、文本等时序数据。需要注意的是，标准 RNN 在训练长序列时会面临梯度消失或梯度爆炸的问题，使得它难以有效学习长程依赖。

LSTM

LSTM 是一种特别设计的 RNN，旨在解决标准 RNN 中的梯度消失问题。LSTM 引入了一个称为“记忆单元”的结构，并通过三个“门”来控制信息的流动：遗忘门（决定丢弃多少先前的记忆）、输入门（控制多少新的信息进入记忆单元）和输出门（决定记忆单元中哪些信息被输出）。这些门使得 LSTM 能够在较长时间内保持信息，从而有效地捕捉长程依赖关系，广泛应用于机器翻译、语音识别等任务。

GRU

GRU 是 LSTM 的简化版，结合了 LSTM 中的遗忘门和输入门为一个单一的更新门，同时没有独立的记忆单元。GRU 通过控制信息更新的比例和重置前一时刻的记忆状态，达到类似于 LSTM 的效果。由于 GRU 的结构更简单，参数更少，其在计算上比 LSTM 更高效，训练速度更快，并且在许多任务中能够获得与 LSTM 相似甚至更好的表现。GRU 适用于资源受限或需要更快训练的场景。

诗歌的生成过程

对于训练好的模型，诗歌的生成过程如下：

首先需要给定一个起始字并转换为对应的数字索引，并作为输入传递给模型。模型中的word_embedding层会将这个数字索引映射为一个稠密的词向量，其中包含了字的语义信息。

```
class word_embedding(nn.Module):
    def __init__(self,vocab_length , embedding_dim):
        super(word_embedding, self).__init__()
        w_embeding_random_intial = np.random.uniform(-1,1,size=(vocab_length
,embedding_dim))
        self.word_embedding = nn.Embedding(vocab_length,embedding_dim)

 self.word_embedding.weight.data.copy_(torch.from_numpy(w_embeding_random_intial)
)
    def forward(self,input_sentence):
        """
        :param input_sentence:  a tensor ,contain several word index.
        :return: a tensor ,contain word embedding tensor
        """
        sen_embed = self.word_embedding(input_sentence)
        return sen_embed
```

接下来，这个词向量会被传入LSTM网络。在这一过程中，LSTM根据当前输入（即当前字的词向量）和之前的隐藏状态（包括记忆单元的状态）来计算新的隐藏状态和记忆单元。这个新的隐藏状态包含了关于已经生成的诗句的所有信息，它决定了接下来要生成的字。

```
    def forward(self,sentence,is_test = False):
        batch_input =
self.word_embedding_lookup(sentence).view(1,-1,self.word_embedding_dim)
        # print(batch_input.size()) # print the size of the input
        ################################################
        # here you need to put the "batch_input"  input the self.lstm which is
defined before.
        # the hidden output should be named as output, the initial hidden state
and cell state set to zero.
        # ???
        h0=torch.zeros(2, batch_input.size(0), self.lstm_dim)
        c0=torch.zeros(2, batch_input.size(0), self.lstm_dim)
        output,_=self.rnn_lstm(batch_input,(h0,c0))
        ################################################
```

LSTM的输出是一个新的隐藏状态，代表了当前输入字及其上下文信息。这个输出会被传递到全连接层，并与一组权重矩阵相乘，生成一个大小为词汇表大小的向量，这个向量的每个元素代表生成下一个字的概率。然后，应用softmax函数对其进行归一化，得到一个概率分布，表示每个字作为下一个字的可能性。接着，通过to_word选择可能性最高的字为输出的下一个字，并通过词汇表将其转回汉字编码。生成过程不断迭代生成下一个字，直到生成了结束符或者达到了长度上限。

```
        out = output.contiguous().view(-1,self.lstm_dim)

        out =  F.relu(self.fc(out))

        out = self.softmax(out)

        if is_test:
            prediction = out[ -1, : ].view(1,-1)
            output = prediction
        else:
           output = out
        # print('tt',out)
        return output
```

代码中主要生成过程在gen_poem函数中：

```
def gen_poem(begin_word):
    #...模型初始化等...
    while word != end_token:
        input = np.array([word_int_map[w] for w in poem],dtype= np.int64)
        input = Variable(torch.from_numpy(input))
        output = rnn_model(input, is_test=True)
        word = to_word(output.data.tolist()[-1], vocabularies)
        poem += word
        if len(poem) > 80:
            break
    return poem
```

生成完毕后，最终的诗歌会由起始字和模型生成的字组成。为了让输出更具可读性，通过pretty_print_poem函数对诗歌进行了格式化（这里我改动了一些把连续逗号也删掉了），使其去除起始和结束符号，并按句号分行显示。

生成诗歌

训练过程

```
******************************
epoch   0 batch number 162 loss is:  6.227686882019043
prediction [33, 5, 4, 6, 45, 0, 13, 6, 4, 4, 24, 1, 4, 6, 9,
b_y        [62, 7, 589, 4579, 954, 0, 674, 1632, 516, 268, 16
******************************
epoch   0 batch number 163 loss is:  6.22873592376709
prediction [25, 23, 4, 6, 42, 0, 25, 112, 4, 4, 1, 1, 36, 7,
b_y        [11, 22, 546, 182, 552, 0, 1342, 926, 130, 102, 16
******************************
epoch   0 batch number 164 loss is:  6.163106918334961
prediction [84, 8, 4, 18, 44, 0, 46, 7, 4, 4, 25, 1, 7, 27, 4
b_y        [906, 66, 313, 290, 57, 0, 1391, 163, 221, 423, 25
******************************
```

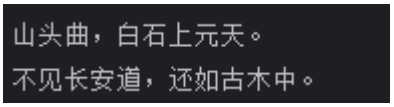
日

```
日色已成，曲江云雨。
四时有风，万国万家。
日光有，金玉莲。
玉山有，一居。
```

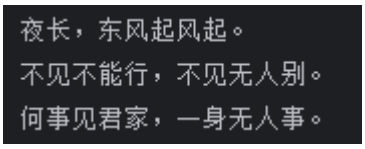
红

```
红树中花。
不见天香，天心不举。
我有三年，神昭万里。
神光中，见有仙，金玉玉，金玉虚。
人不见，天上有，万古风。
```

山

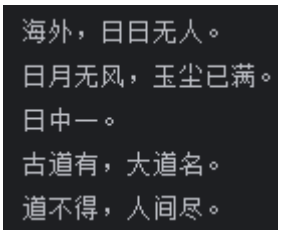


夜

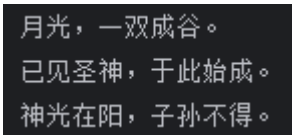


湖

海



月



总结

本次实验中，我在理论学习的基础上，尝试实践了RNN模型的实现与训练。通过观察实验结果，我发现其实模型生成的句子中往往单看词语还是对的，甚至简单的语法在输出中也有体现出来。但是，从整个句子乃至整首诗来看，输出的句子大多数还是并不通顺，句意也表达不清，更不用说诗意了。经过思考，我认为主要还是因为本次实验中搭建的RNN模型结构较为简单，体量也较小，不足以支撑其很好地理解诗歌中的句意和语法以及诗意。此外，由于数据集中的诗歌形式多样，模型也没有很好地理解并理解这些不同的格律，甚至这些多样性可能还对模型的学习造成了干扰。不过，对于本次实验中较为简单的模型来说，能够初步理解简单的词语和语法已经是差强人意了，也足以体现出RNN相对于以往学过的机器学习模型的特点以及优势。

总而言之，通过本次实验，我很好地加深了对RNN的理解，为我未来的科研或者工作提供了宝贵的实践经验。此外，在本次实验中我为了加快训练速度重装了支持CUDA的torch版本，对我而言也是一次珍贵的经历。综上，我认为本次实验对于我的学习以及发展来说还是很有益处的。