# 深度学习第 4 次作业

2152180 文溪

## 1. 补全程序，主要是前面的 3 个空和 生成诗歌的一段代码。(tensorflow)

程序已经补全，完成的是 tensorflow 版本的。

```
    @tf.function
    def call(self, inp_ids):
        '''
        此处完成建模过程，可以参考Learn2Carry
        '''
        inp_emb = self.embed_layer(inp_ids)
        rnn_out = self.rnn_layer(inp_emb)
        logits = self.dense(rnn_out)
        return logits
```

首先在此处完成了建模过程，使用内部函数来实现通过输入的词汇索引来返回每个时间步预测得到的每个词汇的几率。

```
@tf.function
def train_one_step(model, optimizer, x, y, seqlen):
    '''
    完成一步优化过程，可以参考之前做过的模型
    '''
    with tf.GradientTape() as tape:
        logits = model(x)
        loss = compute_loss(logits, y, seqlen)
    # 计算梯度
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
    return loss
```

之后在一步训练中使用 tensorflow 的上下文管理器来辅助计算梯度，并使用 tf 自带的优化器来优化梯度。

```python
import random
def gen_sentence(start_words=['日', '红', '山', '夜', '湖', '海', '月']):
    # 随机选择一个起始词
    start_word = random.choice(start_words)
    # 检查起始词是否在词汇表中
    if start_word not in word2id:
        raise ValueError(f"起始词 '{start_word}' 不在词汇表中。")

    # 将起始词转换为模型可接受的格式
    cur_token = tf.constant([word2id[start_word]], dtype=tf.int32)
    # 初始化隐藏状态
    state = tf.zeros([1, 128])
    collect = [start_word]  # 收集生成的词汇

    for _ in range(50):  # 假设诗歌最多50个词
        cur_token, state = model.get_next_token(cur_token, state)
        next_word_id = cur_token.numpy()[0]
        # 如果生成结束标记，则结束
        if id2word[next_word_id] == end_token:
            break
        collect.append(id2word[next_word_id])
        # 更新当前词为下一次迭代的输入
        cur_token = tf.constant([next_word_id], dtype=tf.int32)

    return ''.join(collect)

# 打印生成的诗歌
for i in range(10):
    generated_poem = gen_sentence(start_words=['日', '红', '山', '夜', '湖', '海', '月'])
    print(generated_poem)
```
✓ 0.1s

最后编写符合题目要求的生成诗歌代码，开头词汇是 “ 日 、 红 、 山 、 夜 、 湖、 海 、 月 ”，结果在第四部分展示。

## 2. 解释一下 RNN ，LSTM，GRU 模型

循环神经网络（RNN）

基本概念：RNN 是一种用于处理序列数据的神经网络。与传统的前馈神经网络不同，RNN 在处理序列的每个元素时，都会将一定量的前一步的信息传递到当前步骤。这通过在网络中引入循环来实现，使得网络能够保持某种状态，从而捕获序列中时间上的动态特性。

问题：虽然理论上 RNN 能够捕获长距离依赖，但在实践中，由于梯度消失和梯度爆炸问题，RNN 很难学习到序列中时间上距离较远的依赖关系。

长短期记忆网络（LSTM）

基本概念：LSTM 是一种特殊类型的 RNN，旨在解决标准 RNN 模型难以学习长期依赖的问题。LSTM 通过引入三个门（遗忘门、输入门和输出门）和一个细胞状态（Cell State），可以在长序列中更有效地传递信息。

特点：

遗忘门：决定哪些信息应该从细胞状态中丢弃。

输入门：决定哪些新的信息应该被添加到细胞状态中。

输出门：基于当前的细胞状态和输入，决定最终的输出。

这些门结构使得 LSTM 能够在序列中保持长期的依赖关系，同时避免了梯度消失和爆炸的问题。

门控循环单元（GRU）

基本概念：GRU 是 LSTM 的一种变体，同样旨在解决标准 RNN 无法有效捕获长期依赖的问题。GRU 通过两个门（更新门和重置门）简化了 LSTM 的结构。

特点：

更新门：决定细胞状态中的哪些信息需要被更新。

重置门：决定如何结合新的输入信息与过去的信息。

GRU 在某些情况下可以提供与 LSTM 类似的性能，但是参数更少，计算效率更高。

总的来说，RNN 能够处理序列数据，但在实践中难以捕捉长期依赖。LSTM 通过引入门控机制和细胞状态，有效解决了梯度消失的问题，能够捕获长期依赖关系。GRU 是 LSTM 的简化版本，通过减少门的数量来减少模型的复杂度，同时仍然保持了对长期依赖的捕捉能力。

### 3. 叙述一下 这个诗歌生成的过程。

- 数据准备与预处理

首先需要收集一个包含大量诗歌的数据集。这个数据集通常包含不同风格和时期的诗歌，以确保模型学习到丰富的语言模式。在这里就是 poems.txt，里面准备了特别多的诗歌，支持 RNN 学习。

同时将诗歌转换成一系列数字（即单词或字符的索引），这样才能被模型处理。同时，加入特殊标记，如开始（bos）和结束（eos）标记，帮助模型识别序列的起始和结束。

- 使用 RNN 模型

构建使用 RNN 模型时需要定义多个层，包括嵌入层（将单词或字符的索引转换成密集向量），一个或多个循环层（处理序列数据，学习语言模式），以及输出层（根据学到的模式预测下一个单词或字符）

- 训练模型

准备训练数据：从预处理过的数据集中准备训练数据，通常包括输入序列和目标序列（输入序列的下一个单词或字符）。

设置损失函数和优化器：使用适合序列预测任务的损失函数（如交叉熵损失）和优化器（如 Adam）。

训练过程：通过多次迭代训练数据，不断更新模型的权重，以最小化损失函数值，从而使模型能够更好地预测序列中的下一个单词或字符。

- 生成诗歌

选择起始词：从预定义的起始词集合中随机选择一个作为生成序列的开始，或指定一个起始词。

迭代生成文本：基于当前的序列（起始词或已生成的序列的一部分），模型预测下一个最可能的单词或字符，然后将这个预测加入到序列中。这个过程重复进行，直到遇到结束标记或达到预定的序列长度。

## 4. 生成诗歌 开头词汇是“ 日 、 红 、 山 、 夜 、 湖、 海 、 月 ”，等词汇作为 begin word

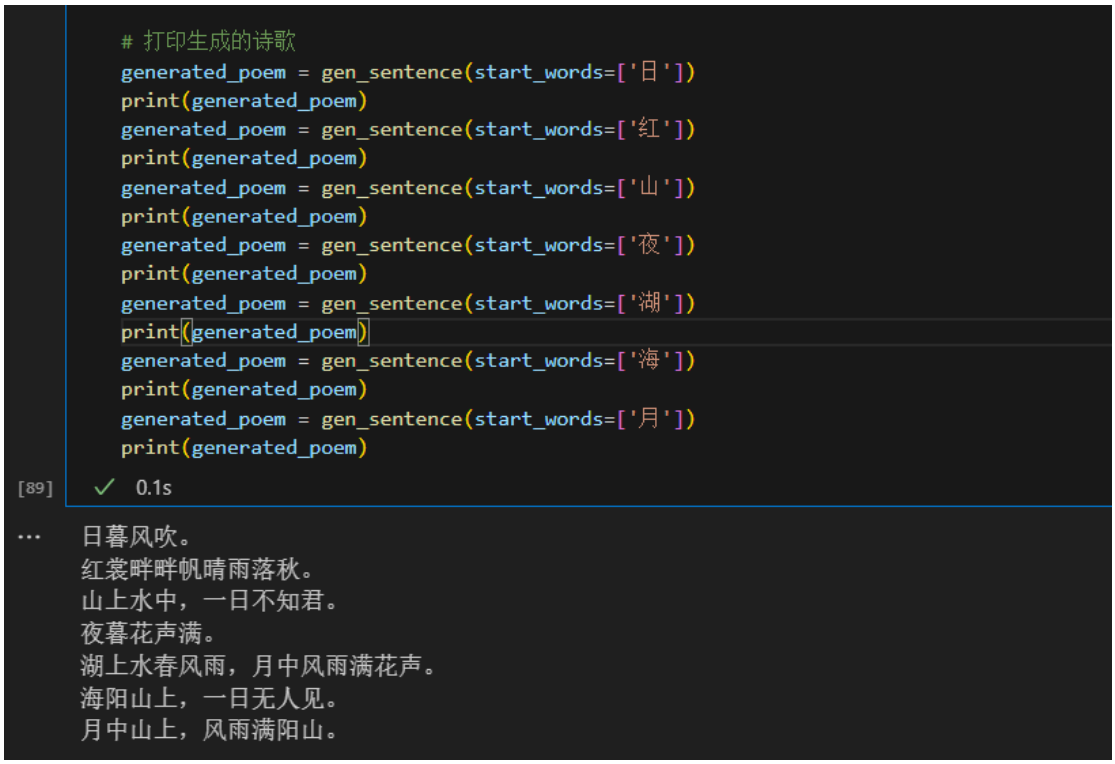
```python
def gen_sentence(start_words=[]):
    # 随机选择一个起始词
    start_word = start_words[0]
    # 检查起始词是否在词汇表中
    if start_word not in word2id:
        raise ValueError(f"起始词 '{start_word}' 不在词汇表中。")

    # 将起始词转换为模型可接受的格式
    cur_token = tf.constant([word2id[start_word]], dtype=tf.int32)
    # 初始化隐藏状态
    state = tf.zeros([1, 128])
    collect = [start_word]  # 收集生成的词汇

    for _ in range(50):  # 假设诗歌最多50个词
        cur_token, state = model.get_next_token(cur_token, state)
        next_word_id = cur_token.numpy()[0]
        # 如果生成结束标记，则结束
        if id2word[next_word_id] == end_token:
            break
        collect.append(id2word[next_word_id])
        # 更新当前词为下一次迭代的输入
        cur_token = tf.constant([next_word_id], dtype=tf.int32)

    return ''.join(collect)

# 打印生成的诗歌
generated_poem = gen_sentence(start_words=['日'])
print(generated_poem)
generated_poem = gen_sentence(start_words=['红'])
print(generated_poem)
generated_poem = gen_sentence(start_words=['山'])
print(generated_poem)
generated_poem = gen_sentence(start_words=['夜'])
print(generated_poem)
generated_poem = gen_sentence(start_words=['湖'])
print(generated_poem)
generated_poem = gen_sentence(start_words=['海'])
print(generated_poem)
generated_poem = gen_sentence(start_words=['月'])
```




生成的诗歌如上：（训练一次模型一个多小时，确实好久）

日暮风吹。

红裳畔畔帆晴雨落秋。

山上水中，一日不知君。

夜暮花声满。

湖上水春风雨，月中风雨满花声。

海阳山上，一日无人见。

月中山上，风雨满阳山。