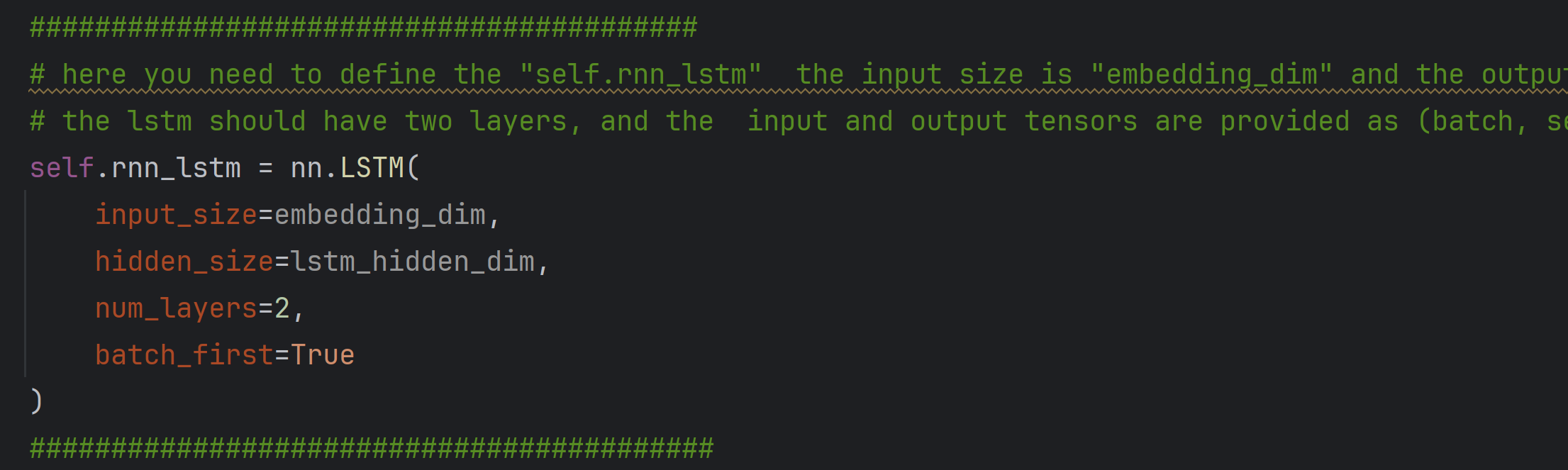
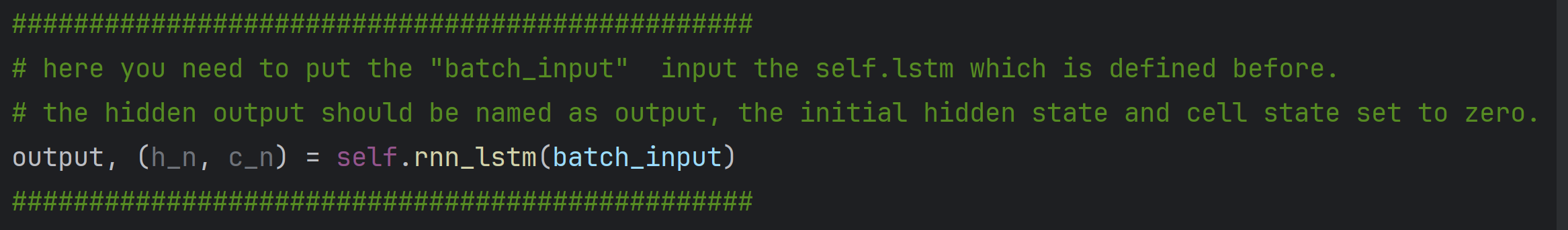
**RNN生成唐诗（pytorch版）**

2251499 桂欣远

# 代码补全：

LSTM层定义：  


前向传播逻辑：



# RNN、LSTM、GRU 模型解释

## RNN（循环神经网络）

 处理序列数据的经典模型，其核心是通过循环结构将前一时刻的隐藏状态传递到当前时刻，从而捕捉序列中的时序信息。例如，在生成诗歌时，RNN会根据已生成的字符预测下一个字符。然而，传统RNN存在梯度消失和梯度爆炸的问题，当序列较长时，模型难以记住远距离的依赖关系，像是诗歌中前后句的押韵或主题的一致，容易导致生成内容逻辑断裂或重复。

## LSTM（长短期记忆网络）

个人感觉更像RNN的改进版本，通过引入门控机制和细胞状态解决长期依赖问题。LSTM包含三个关键门：

遗忘门：决定哪些历史信息需要丢弃，当诗歌主题变化时，遗忘无关的上下文。

输入门：控制哪些新信息需要存储到细胞状态，记住新诗句的关键意象。

输出门：生成押韵的字符时基于当前输入和细胞状态，决定最终的隐藏状态输出。

LSTM的细胞，允许梯度在长距离传播时保持稳定，从而有效建模诗歌中的长程结构

## GRU（门控循环单元）

是LSTM的简化版本，将输入门和遗忘门合并为更新门，并引入重置门，减少了参数量，训练速度更快。重置门决定如何将过去信息与当前输入结合，更新门控制隐藏状态的更新程度。GRU在短序列任务中表现接近LSTM，但在复杂的长序列生成（如长篇诗歌）中，LSTM因更精细的门控设计通常更具优势。

# 诗歌生成过程解释

# （以静夜思为例子分析）

## 数据预处理与词表构建

代码文件：main.py 中的 process\_poems1 函数。

步骤：

1. 读取与清洗：

输入文件 poems.txt 的每一行格式为 标题:内容，例如：

**静夜思:床前明月光，疑是地上霜**

清洗时会移除空格、特殊符号，并过滤长度不符或格式错误的行。

2. 添加起止标记：

每首诗被包裹为 G内容E，例如：

**G床前明月光，疑是地上霜E**

3. 构建词表映射：

统计所有字符的频率，生成 字符 → 索引 的字典。

**word\_int\_map = {'G': 0, '床': 1, '前': 2, '明': 3, '月': 4, ...}**

4. 转换为数字序列：

诗歌被转换为索引序列，例如：

**[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**

## 模型架构

代码文件：rnn.py 中的 RNN\_model 类。

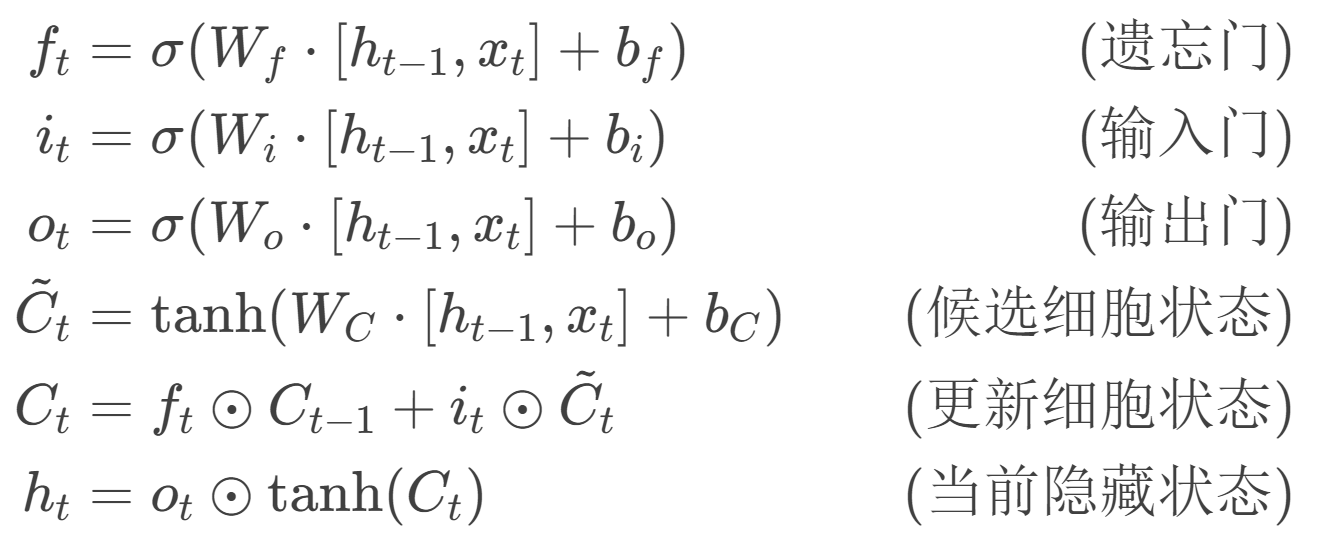
模型组件：

词嵌入层：将索引映射为稠密向量，例如：

**embedding = word\_embedding(vocab\_length=5000, embedding\_dim=100)**

字符“床”（索引1）被映射为 [0.2, -0.5, ..., 0.7]（100维向量）。

LSTM层：



全连接层与Softmax：

**self.fc = nn.Linear(128, 5000) 映射到词表空间**

**self.softmax = nn.LogSoftmax(dim=1)**

## 训练过程

代码逻辑：main.py 中的 run\_training 函数。

步骤：

1. 生成批次数据：

输入序列为 [G, 床, 前, 明]，标签序列为 [床, 前, 明, 月]。

2. 前向传播：

输入序列经过词嵌入层，转换为形状为 (batch\_size, seq\_len, 100) 的张量。

LSTM处理序列，输出形状为 (batch\_size, seq\_len, 128) 的隐藏状态。

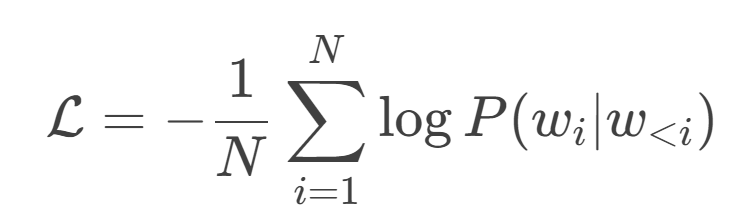
全连接层将隐藏状态映射到词表空间，得到形状为 (batch\_size \* seq\_len, 5000) 的 logits。

使用 LogSoftmax 计算概率分布：



3. 损失计算：

使用负对数似然损失（NLLLoss）：



4. 反向传播与优化：

通过RMSprop优化器更新参数，梯度裁剪防止爆炸。

## 诗歌生成（自回归预测）

代码逻辑：main.py 中的 gen\_poem 函数。

步骤示例（以生成“日”开头的诗为例）：

1. 初始化：

输入起始字符“日”，转换为索引序列 [word\_int\_map["日"]] = [10]。

2. 逐步生成：

第1步：

输入 [10] → 模型预测下一个字符的概率分布。

假设概率最高的字符是“落”（索引15），追加到序列得到 [10, 15]。

第2步：

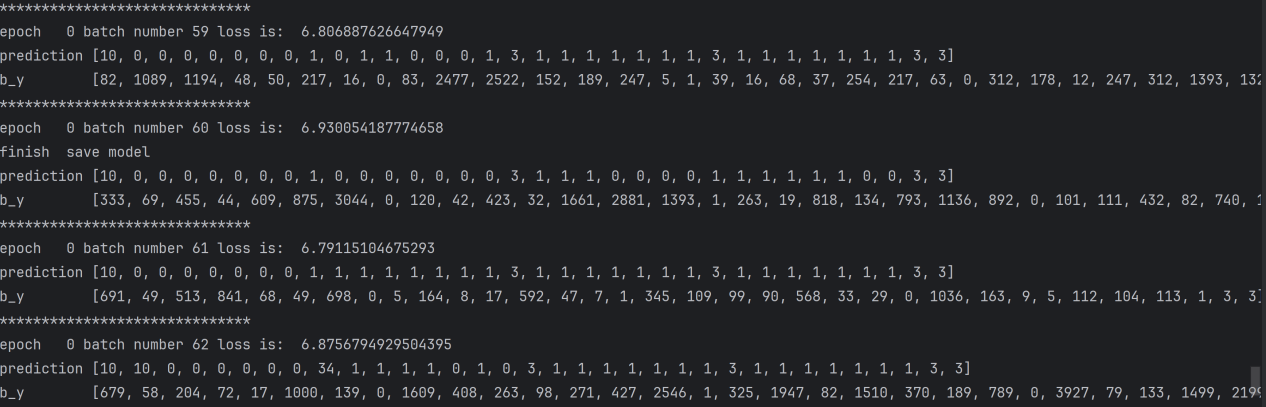
输入 [10, 15] → 模型预测下一个字符为“长”（索引20），序列变为 [10, 15, 20]。

重复：直到生成 E 或达到长度限制。

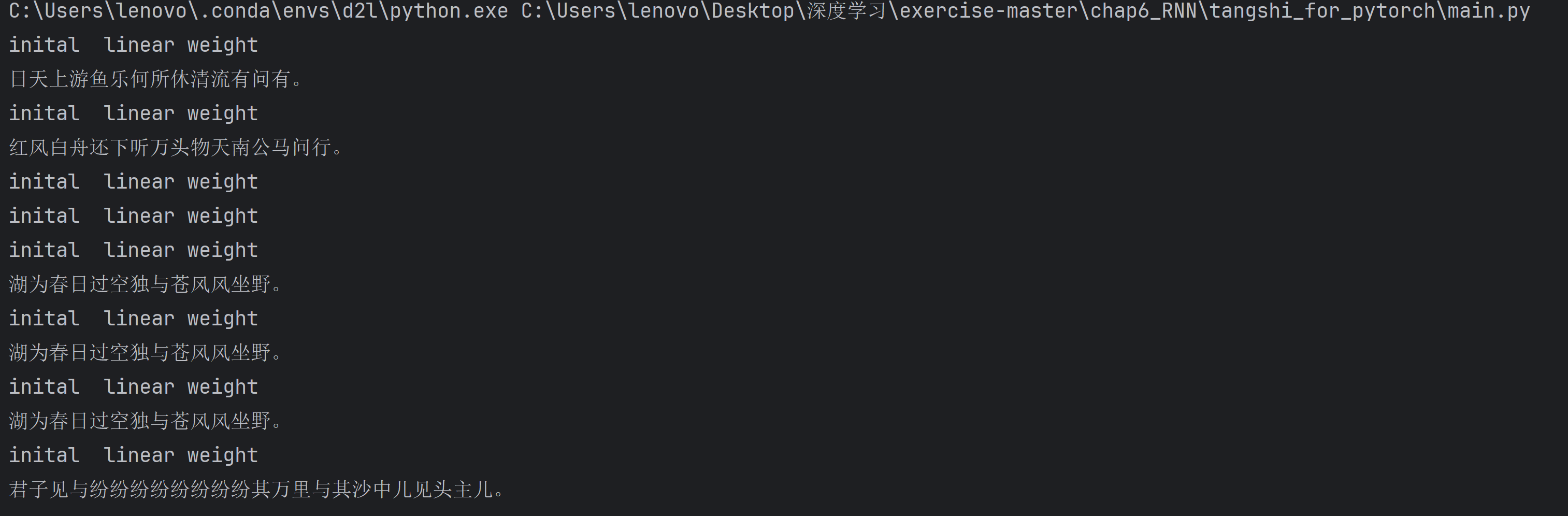
3. 输出结果：

最终序列 [G, 日, 落, 长, 河, E] 转换为诗句：“日落长河”。

# 训练过程和结果 尝试使用poems训练时的过程：



由于使用CPU来训练模型比较慢，就使用了较小的tangshi作为数据集训练，并进行推理得到的结果：（“山”和“夜”没有输出,可能是因为使用的tangshi数据集太小，没有充分学习这些词的上下文关系导致山和夜提前生成了E，提前结束）



# 实验总结

本次实验使用PyTorch实现了基于RNN的唐诗生成模型。运行整个代码结构首先对唐诗数据集进行了预处理，包括清洗数据、添加起止标记、构建词表映射以及将诗歌转换为数字序列等步骤。接着，构建了一个包含词嵌入层、LSTM层、全连接层和Softmax激活函数的RNN模型。在训练过程中，通过生成批次数据、前向传播、损失计算和反向传播等步骤，利用RMSprop优化器更新模型参数，并采用梯度裁剪防止梯度爆炸。最后，在生成诗歌时，使用自回归预测的方法，逐步生成字符直到遇到结束标记或达到长度限制。实验中，由于使用较小的tangshi数据集进行训练，导致在生成以“山”和“夜”开头的诗歌时出现提前终止的问题，这可能是由于数据不足导致模型未能充分学习诗歌的生成规律。