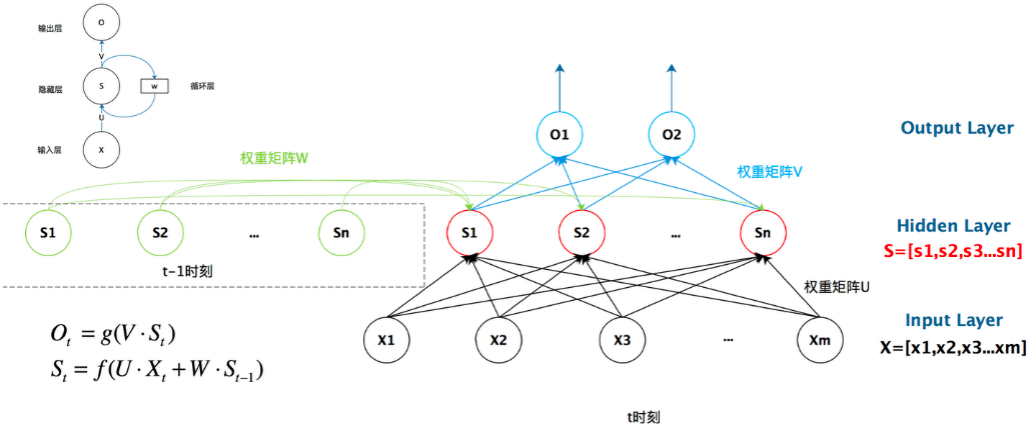
实验报告

模型解释

1.RNN



RNN 是一种用于处理序列数据的神经网络结构，适用于时间序列、语音识别、自然语言处理等任务。其核心思想是**引入循环结构，使得当前时刻的输出可以依赖于前一时刻的隐藏状态**，从而能够捕捉时间序列中的动态信息。

RNN 的基本结构

- 设 x_t 为时间步 t 的输入，h_t 为隐藏状态，y_t 为输出，则 RNN 的计算公式如下：

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h)$$

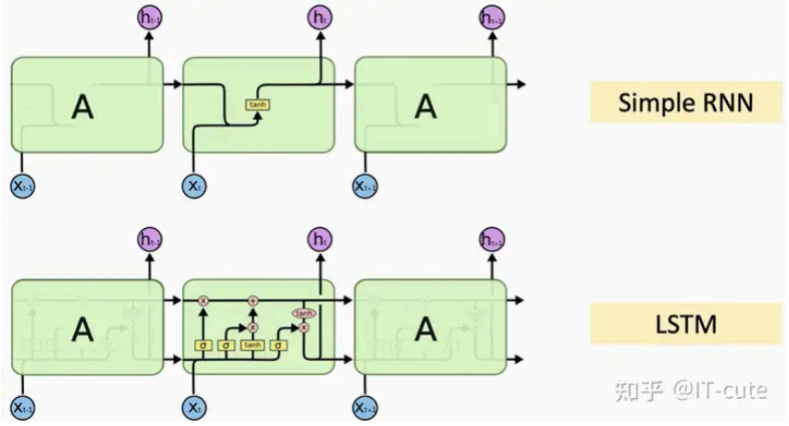
$$y_t = W_y h_t + b_y$$

- 其中，W_h, W_x, W_y 为权重矩阵，b_h, b_y 为偏置项。

RNN 的问题

- **梯度消失/梯度爆炸**：随着时间步的增加，梯度可能会指数级衰减或增长，导致长时依赖难以学习。
- **无法有效记忆长序列信息**：标准 RNN 在长时间依赖任务上的表现较差。

2.LSTM



LSTM 是 RNN 的改进版本，**通过引入“门”机制来解决梯度消失问题，从而能够学习长时间依赖关系。**

LSTM 结构 LSTM 主要由 **输入门、遗忘门、输出门** 组成，核心是**细胞状态（Cell State）**，它允许梯度在较长序列中有效传播。

计算公式：

- **遗忘门**（决定遗忘多少旧信息）：

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

- **输入门**（决定更新多少新信息）：

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

- **细胞状态更新**：

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **输出门**（决定输出多少信息）：

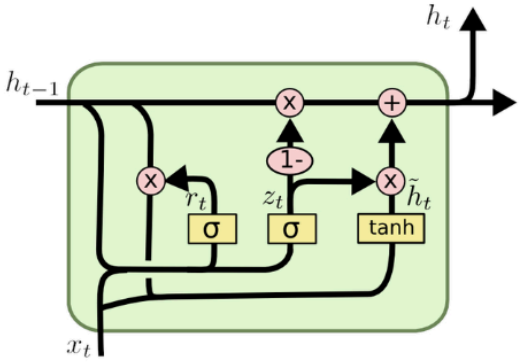
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

LSTM 的优势

- 通过 **遗忘门** 控制信息丢失，使模型更容易学习长时间依赖关系。
- 通过 **输入门和输出门** 控制信息流动，避免梯度消失问题。

3.GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU 是 LSTM 的简化版本，它将 **输入门和遗忘门合并为“更新门”**，减少了计算复杂度，同时仍然可以解决梯度消失问题。

GRU 结构 GRU 主要由 **更新门** 和 **重置门** 组成：

- **更新门**：控制旧信息和新信息的比例

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

- **重置门**：控制丢弃多少旧信息

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

- **候选隐藏状态**：

$$\tilde{h_t} = \tanh(W_h[r_t * h_{t-1}, x_t] + b_h)$$

- **最终隐藏状态更新**：

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$$

GRU 的优势

- 计算复杂度较 LSTM 更低，参数更少，适用于资源受限的场景。
- 由于结构更简单，GRU 在某些任务上的表现与 LSTM 相近甚至更好。

诗歌生成过程

这个诗歌生成项目基于循环神经网络（RNN）实现，主要分为以下几个步骤：

数据预处理

- **读取诗歌文件**：从指定文件中读取诗歌内容。
- **过滤无效诗歌**：去除包含特殊字符和不符合长度要求的诗歌。
- **添加起始和结束标记**：在每首诗歌的开头和结尾分别添加起始标记“G”和结束标记“E”。
- **生成诗歌向量**：将每首诗歌中的每个字转换为对应的整数索引。

模型构建

- **词嵌入层**：将整数索引的词转换为向量表示。
- **LSTM层**：使用两层LSTM网络，捕捉词语的序列关系。
- **全连接层**：将LSTM的输出转换为词汇表大小的向量。
- **初始化权重**：对模型的权重进行初始化。

模型训练

- **生成批次数据**：将诗歌向量按批次大小进行分割。
- **前向传播**：将输入数据通过模型进行前向传播。
- **计算损失**：使用NLLLoss函数计算预测值与真实值之间的损失。
- **反向传播**：根据损失进行反向传播，更新模型参数。

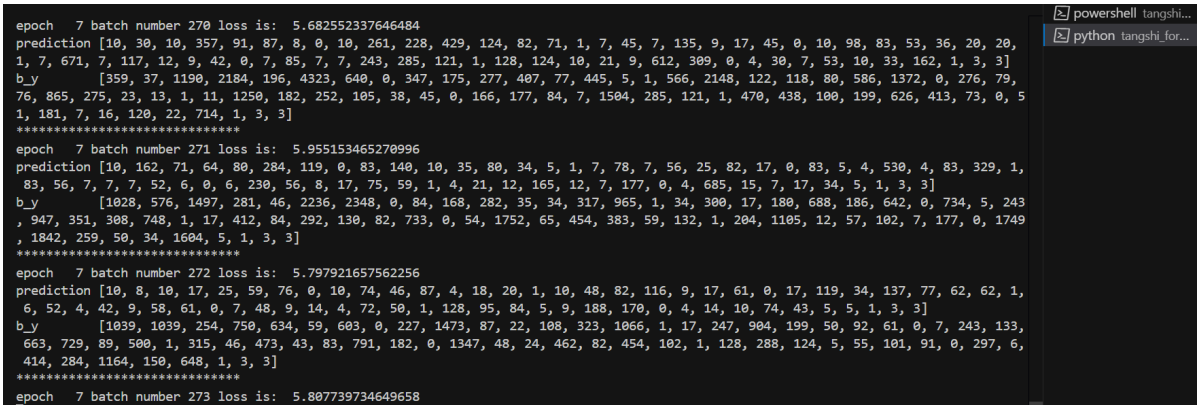
生成诗歌

- **加载训练好的模型**：加载之前训练好的模型参数。
- **指定起始字**：给定一个起始字，开始生成诗歌。
- **循环预测**：根据当前已生成的诗歌内容，不断预测下一个字。

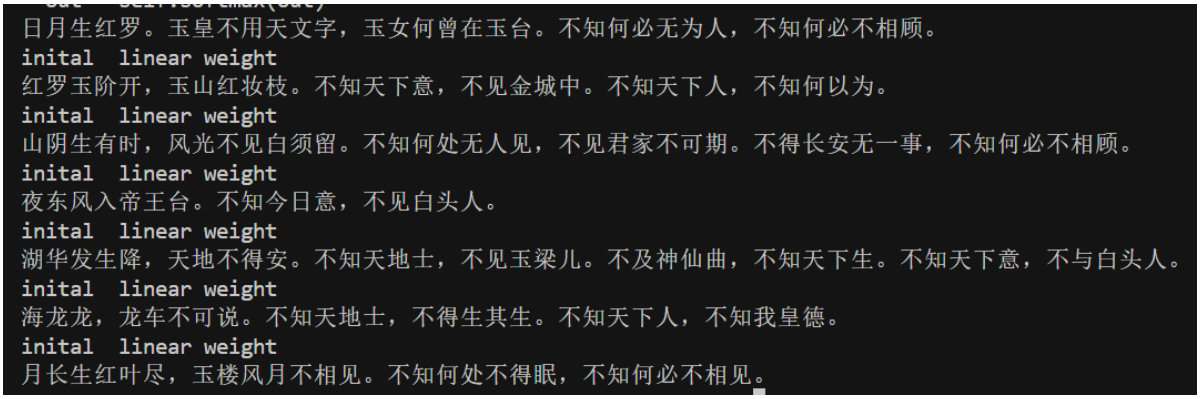
- **结果输出**：将生成的诗歌内容进行格式化输出。

实验结果

训练截图



结果输出



实验总结

在这个诗歌生成实验中，我通过构建和训练RNN模型，成功实现了根据给定起始字生成古诗的功能。模型能够学习古诗的模式和规律，生成具有一定形式和连贯性的诗歌。

在实验过程中，我深刻体会到了RNN及其变体LSTM在处理序列数据方面的强大能力。通过LSTM层的有效捕捉序列信息，使得生成的诗歌不仅在形式上符合古诗的特点，还具有一定的逻辑性。同时，我也认识到了数据质量和数量对模型性能的重要影响，良好的数据预处理和充足的训练数据是提高模型效果的关键。