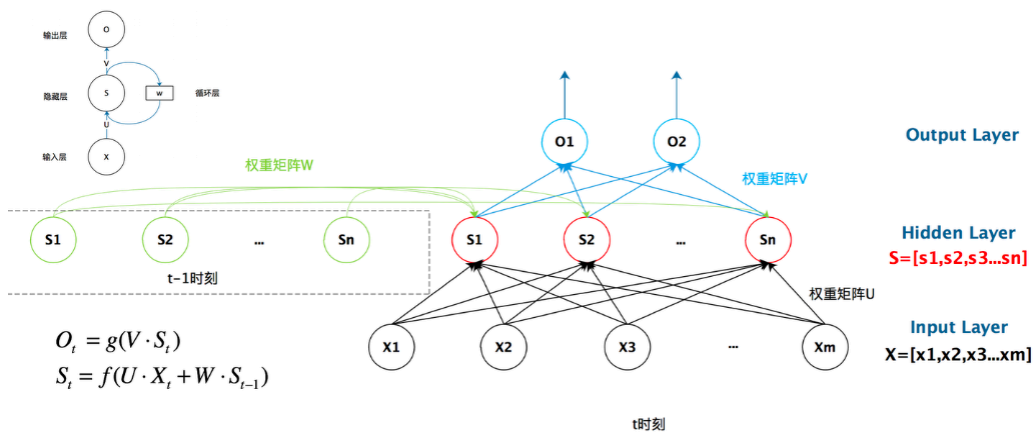


# 实验报告

## 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：

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

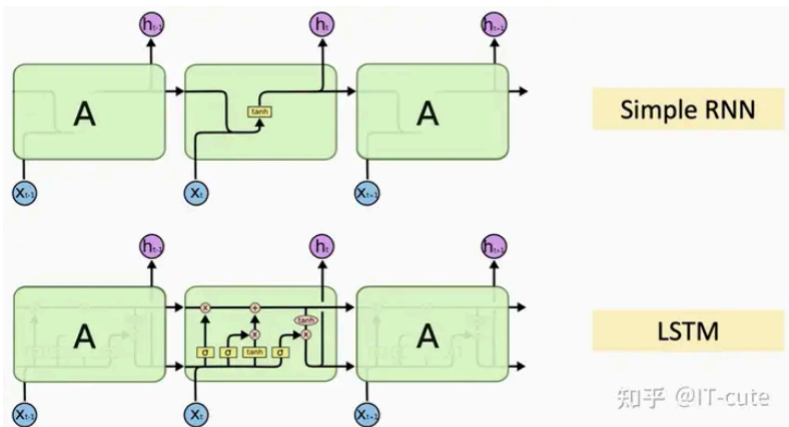
$$y_t = W_y h_t + b_y$$

- 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)：

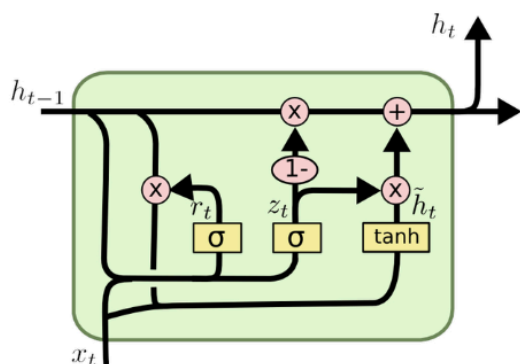
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

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

**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



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

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

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

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

<https://blog.csdn.net/jerr...>

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

```
epoch 7 batch number 270 loss is: 5.682552337646484
prediction [10, 30, 10, 357, 91, 87, 8, 0, 10, 261, 228, 429, 124, 82, 71, 1, 7, 45, 7, 135, 9, 17, 45, 0, 10, 98, 83, 53, 36, 20, 20,
1, 7, 671, 7, 117, 12, 9, 42, 0, 7, 85, 7, 7, 243, 285, 121, 1, 128, 124, 10, 21, 9, 612, 309, 0, 4, 30, 7, 53, 10, 33, 162, 1, 3, 3]
b_y [359, 37, 1190, 2184, 196, 4323, 640, 0, 347, 175, 277, 407, 77, 445, 5, 1, 566, 2148, 122, 118, 80, 586, 1372, 0, 276, 79,
76, 865, 275, 23, 13, 1, 11, 1250, 182, 252, 105, 38, 45, 0, 166, 177, 84, 7, 1504, 285, 121, 1, 470, 438, 100, 199, 626, 413, 73, 0, 5
1, 181, 7, 16, 120, 22, 714, 1, 3, 3]
*****
epoch 7 batch number 271 loss is: 5.955153465270996
prediction [10, 162, 71, 64, 80, 284, 119, 0, 83, 140, 10, 35, 80, 34, 5, 1, 7, 78, 7, 56, 25, 82, 17, 0, 83, 5, 4, 530, 4, 83, 329, 1,
83, 56, 7, 7, 52, 6, 0, 6, 230, 56, 8, 17, 75, 59, 1, 4, 21, 12, 165, 12, 7, 177, 0, 4, 685, 15, 7, 17, 34, 5, 1, 3, 3]
b_y [1028, 576, 1497, 281, 46, 2236, 2348, 0, 84, 168, 282, 35, 34, 317, 965, 1, 34, 300, 17, 180, 688, 186, 642, 0, 734, 5, 243
, 947, 351, 308, 748, 1, 17, 412, 84, 292, 130, 82, 733, 0, 54, 1752, 65, 454, 383, 59, 132, 1, 204, 1105, 12, 57, 102, 7, 177, 0, 1749
, 1842, 259, 50, 34, 1604, 5, 1, 3, 3]
*****
epoch 7 batch number 272 loss is: 5.797921657562256
prediction [10, 8, 10, 17, 25, 59, 76, 0, 10, 74, 46, 87, 4, 18, 20, 1, 10, 48, 82, 116, 9, 17, 61, 0, 17, 119, 34, 137, 77, 62, 62, 1,
6, 52, 4, 42, 9, 58, 61, 0, 7, 48, 9, 14, 4, 72, 50, 1, 128, 95, 84, 5, 9, 188, 170, 0, 4, 14, 10, 74, 43, 5, 5, 1, 3, 3]
b_y [1039, 1039, 254, 750, 634, 59, 603, 0, 227, 1473, 87, 22, 100, 323, 1066, 1, 17, 247, 904, 199, 50, 92, 61, 0, 7, 243, 133,
663, 729, 89, 500, 1, 315, 46, 473, 43, 83, 791, 182, 0, 1347, 48, 24, 462, 82, 454, 102, 1, 128, 288, 124, 5, 55, 101, 91, 0, 297, 6,
414, 284, 1164, 150, 648, 1, 3, 3]
*****
epoch 7 batch number 273 loss is: 5.807739734649658
```

### 结果输出

```
cat $(cat /dev/urandom | fold -n 100 | tr -dc 'a-z' | fold -n 100 | tr -dc 'a-z' | xargs echo) | xargs echo
日月生红罗。玉皇不用天文字，玉女何曾在玉台。不知何必无为人，不知何必不相顾。
inital linear weight
红罗玉阶开，玉山红妆枝。不知天下意，不见金城中。不知天下人，不知何以为。
inital linear weight
山阴生有时，风光不见白须留。不知何处无人见，不见君家不可期。不得长安无一事，不知何必不相顾。
inital linear weight
夜东风入帝王台。不知今日意，不见白头人。
inital linear weight
湖华发生降，天地不得安。不知天地士，不见玉梁儿。不及神仙曲，不知天下生。不知天下意，不与白头人。
inital linear weight
海龙龙，龙车不可说。不知天地士，不得生其生。不知天下人，不知我皇德。
inital linear weight
月长生红叶尽，玉楼风月不相见。不知何处不得眠，不知何必不相见。
```

## 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。