# 常州工学院

# 计算机信息工程学院

## 《数据结构》课程设计报告

| | |
|---|---|
| 题　　目 | 神经网络模型 |
| 年　　级 | 2022 级软件工程 |
| 班　　级 | 22 软件三 |
| 组长姓名学号 | 杨熙承 (22030531) |
| 组员姓名学号 | 王语桐 (22030527) |
| 组员姓名学号 | 林　越 (22030519) |
| 组员姓名学号 | 徐姜旸 (22030528) |
| 指导教师 | 叶　鸿 |

2024 年 1 月 12 日

常州工学院计算机信息工程学院

# 《数据结构》课程设计

# 任 务 书

设计名称：神经网络模型

指导教师：叶　鸿　　下达时间：2023.12.25

学生姓名：杨熙承　　学　　号：22030531

学生姓名：王语桐　　学　　号：22030527

学生姓名：林　越　　学　　号：22030519

学生姓名：徐姜旸　　学　　号：22030528

专　　业：软件工程

## 一、课程设计的基本要求

编写一个应用程序，实现一个简单的神经网络模型。收集一定数量的诗歌数据集，对收集到的诗歌数据进行预处理，使用 Python 构建循环神经网络模型，使用训练数据对循环神经网络模型进行训练，利用测试数据对训练好的模型进行评估。使用训练好的模型，输入一个给定的藏头诗，通过模型生成一首完整的诗歌。

## 二、课程设计的主要内容

**1、主要内容：**通过算法实现 RNN 的神经网络模型并且运用在藏头诗当中。

**2、具体分工：**

(1) 杨熙承（组长）：负责主要代码编写；搭建 RNN 模型搭建和前后端；并且详细分工，使用 git 与组员协作开发。

(2) 王语桐：对代码进行重构和优化；搜集循环神经网络的资料和公式；对大报告进行文案撰写和附录的完成；生成公式和数据的可视化图像；。

(3) 徐姜旸：解释、翻译题目并进行目的分析和需求分析；对代码进行测试、调试；搜集古诗数据并且做预处理；使用对大报告公式的主要撰写以及排版和插图。

(4) 林越：代码反应公式整理；对代码进行 RNN 与 LSTM 的神经网络模型可视化图的设计和流程图的设计；并做大报告的主要文字撰写，文案设计。

## 三、课程设计的进程安排

| 时间 | 安排 |
| --- | --- |
| 2024 年 1 月 1 日之前 | 确定选题和组建团队 |
| 2024 年 1 月 1 日–1 月 2 日 | 实施课程设计：完成功能需求设计和开发环境 |
| 2024 年 1 月 3 日–1 月 8 日 | 编码实现 |
| 2024 年 1 月 9 日 | 答辩验收 |

# 《数据结构》课程设计报告

## 一、 目的

### 1、选题背景

中华优秀传统文化是中华文明的智慧结晶和精华所在，习近平总书记高度重视计算机数字技术对于中华优秀文化的传承和创新作用，“十四五”规划和 2035 远景目标纲要明确指出要用现代科技手段对其进行综合利用，以满足人民日益增长的美好文化需求。

过去一年，以大数据、人工智能为代表的新科技为中华优秀传统文化的传承性和创新性发展加速赋能。本课程设计利用了神经网络 RNN 模型，基于对传统古诗的批量训练，实现了自动生成藏头诗的效果，达到了对中华优秀传统古诗文化创造性转化、创新性发展的目的。

### 2、编码实现

使用 Python 和神经网络来处理和生成文本，通过分析中国古诗文本来进行学习和生成新的诗句，实现一个简单的递归神经网络（RNN）用于处理和生成文本数据。

## 二、需求分析

### 1、课设要求

（1）编写程序以实现一个简单 RNN 神经网络模型。

（2）能接受数据输入，计算模型输出。

（3）能定制激活函数和权重分布函数。

（4）自带反向传播算法实现以具备学习功能。

针对本次课程设计，经过逐一分析要实现的功能点后，本组最终选择了基于纯 python 语言实现神经网络 RNN 和基于 pytorch 的 LSTM 模型，对古诗批量训练以实现自动生成藏头诗的课题。

2、**选题原因**

（1）自动生成藏头诗不仅展示人工智能在文学创作领域的应用，也能为创新传统文化提供新的途径，适应了国家文化战略需求。

（2）在藏头诗生成的背景下，实现一个神经网络模型是理解和模拟古诗结构的基础。模型能够学习古诗的语言规律和风格，有利于自动生成具有艺术感和文化价值的藏头诗。

（3）神经网络能够处理大量的古诗文本数据。输入提供了必要的训练材料，输出则可以观察和评估网络是否能够满足输入材料的要求。

（4）可以通过不同的激活函数和权重初始化方法对模型学习古诗的风格和结构产生显著影响，进而影响生成藏头诗的质量。

（5）可以通过反向传播算法，网络根据生成的诗歌与实际古诗之间的差异来不断调整其参数，从而提高藏头诗的生成质量，从而模型不仅能够复制古诗的风格，而且还能够在给定的藏头约束下创造性地组合词汇和意象。

3、**步骤分析**

（1）收集大量的诗歌样本并且清理诗歌数据，从而进行训练。

（2）生成诗歌数据的词汇集，并建立字符到索引的映射关系，同时建立字词到索引的映射关系，与字符映射做比较分析。

（3）将诗歌文本转换为神经网络的输入和输出数据。

（4）通过以上算法步骤设计模型。

（5）通过迭代，在给定训练数据上更新神经网络参数，进而训练神经网络。

（6）调整神经网络的超参数，例如学习率、隐藏层大小等，已达到最好的效果。

（7）将模型参数导出为文件，以便后续应用。

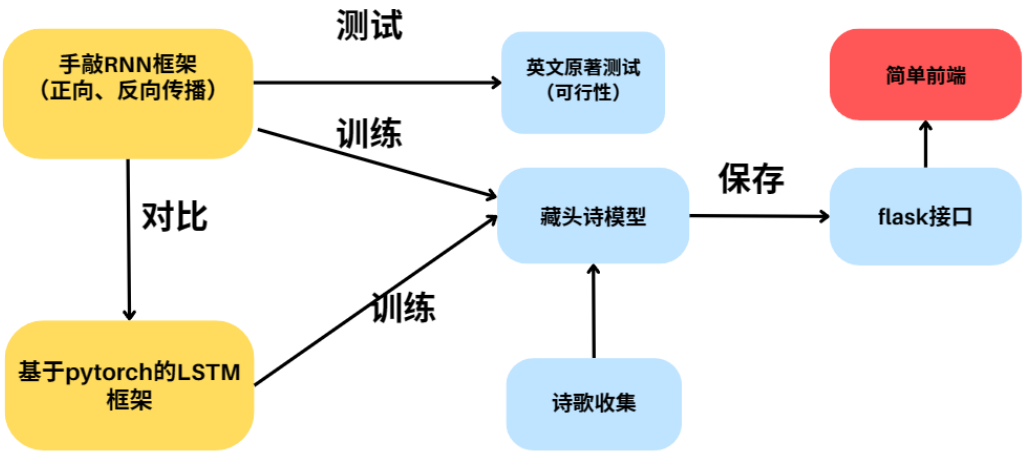
（8）调用保存的模型参数进行藏头诗生成。

（9）设计简单的前端界面，生成可交互性的交互平台。

# 三、概要设计

## 1、流程简介

该课程设计展示了从搭建基本的 RNN 框架开始，到实现 LSTM 模型，再到环境测试、持续集成和模型迭代，最后部署到 Flask 接口的完整机器学习项目开发流程。涉及到了模型开发、测试、集成和部署等关键步骤，体现了机器学习项目的典型开发周期。




## 2、模块简介

**（1）手敲 RNN 框架（正向、反向传播）：**流程图的起始点，表示创建一个循环神经网络模型的基础。从零开始编写代码，实现 RNN 的正向传播算法（用于计算输出和损失）和反向传播算法（用于计算梯度并更新模型权重）。

**（2）基于 Pytorch 的 LSTM 框架：**从手写的 RNN 框架转向使用 Pytorch 高级框架来实现 LSTM 模型。LSTM 是一种特殊的 RNN，用于解决 RNN 在处理长序列时的梯度消失问题。Pytorch 则提供了有效的模块，使实现 LSTM 变得更容易和高效。

**（3）测试环境：**通过损失函数计算每次迭代的损失度，以评估模型的可行性。模型训练过程中，利用多次迭代逐渐调整其参数以最小化损失函数，直到达到某个终止条件的目的。

**（4）flask 接口：**使用一个轻量级的 Web 应用框架将训练好的模型通过 Flask 框架封装成 API 接口，使得模型可以通过网络被访问和使用。

**（5）前端界面：**通过 flask 封装的接口搭建前端界面，进行用户输入、输出可视化。

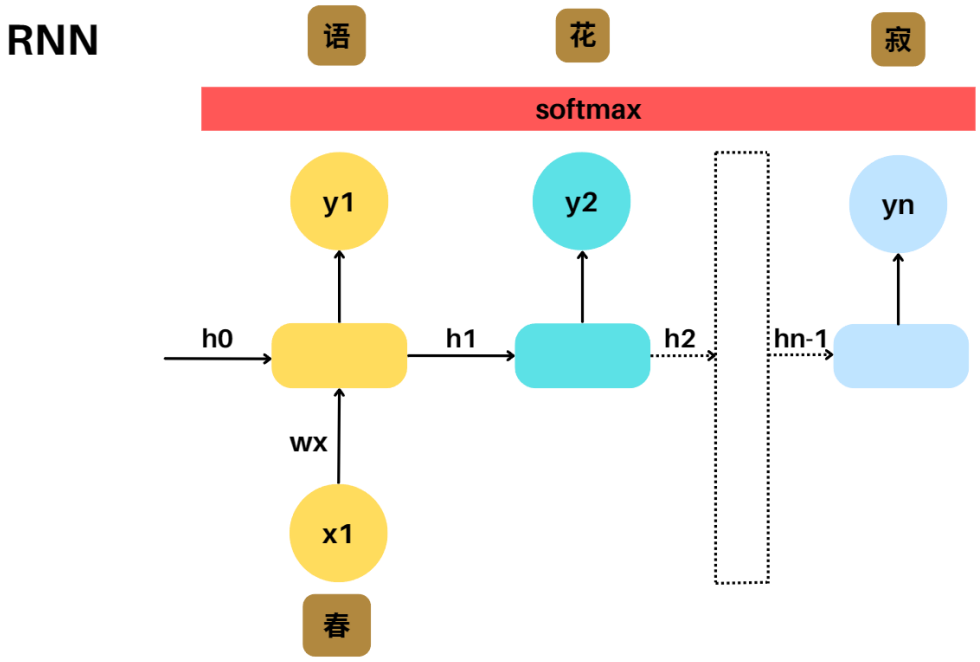
# 四、详细设计

## 1、数据结构与算法设计

### （1）手敲 RNN 模型

#### ① 外部模型

RNN 模型的数据结构设计图如下，展示了一个循环神经网络（RNN）模型的结构。




这张图以下是它的主要组成部分：

**I.输入层（x）：** 该层接收序列中的第一个输入。

**II.隐藏层（h0, h1, ..., hn-1）：** 这些是模型中的状态单元，它们能够将来自前一个状态的信息传递到下一个状态。在图中，黄色和蓝色的矩形分别代表不同时间步的隐藏状态。每个隐藏层都接收来自前一个隐藏状态的信息以及当前时间步的输入。

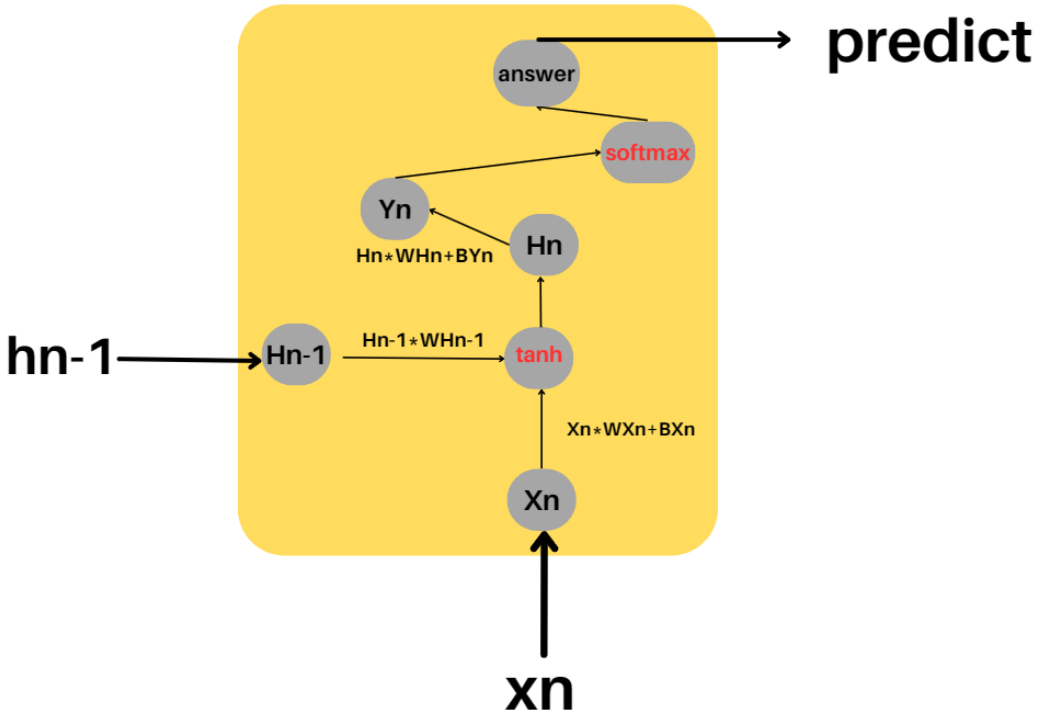
**III.权重（w）：** 这表示输入和隐藏状态之间的权重。

**IV.输出层（y1, y2, ..., yn）：** 每一个时间步的隐藏状态都会导致一个输出，这通常是序列中的下一个预测值。在这个模型中，输出层使用 softmax 函数来处理，softmax 层通常用于多分类问题，它可以将输出转换为概率分布。

图中的汉字代表了序列中的不同阶段或部分，这在处理如文本数据或时间序列数据时尤其有用。例如，在语言模型中，每个字或词都会按照序列被模型处理，这种方式可以帮助模型学习序列中的长期依赖性。

② **内部模型**

RNN 输出层 y 是内部隐藏层 h 与 x 共同决定的，并且通过激活函数进行分类预测，内部具体结构如下：




③ **正向传播步骤**

内部模型的实现采用正向传播算法，具体如下：

**I.设置输入 x:**

**相关代码：**

```
xs[t] = np.zeros((input_size, 1))  # 初始化向量
xs[t][inputs[t]] = 1          # 将输入向量的对应索引设 1
```

**数学公式：**表示为 $x_t[i] = 1$ 其中 $i$ 是在时间步 $t$ 的输入索引。

**II.计算时间步 $t$ 的隐藏状态 $h_t$:**

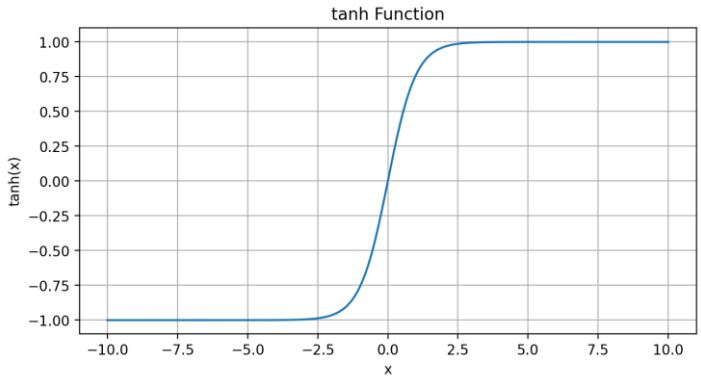
**相关代码：**

```
hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh)
```

**数学公式:**

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

tanh 是常见的处理 0-1 状态的激活函数，图像如下：




其中， $W_{xh}$ 是输入到隐藏状态的权重矩阵， $W_{hh}$ 是前一个隐藏状态到当前隐藏状态的权重矩阵， $b_h$ 是隐藏状态的偏置项。

III.**计算时间步 $t$ 的输出 $y_t$（在应用 softmax 之前的隐藏状态 h）：**

**代码：**

```
ys[t] = np.dot(Why, hs[t]) + by
```

**数学公式:**

$$y_t = W_{hy}h_t + b_y$$

其中 $W_{hy}$ 是隐藏状态到输出的权重矩阵， $b_y$ 是输出的偏置项。

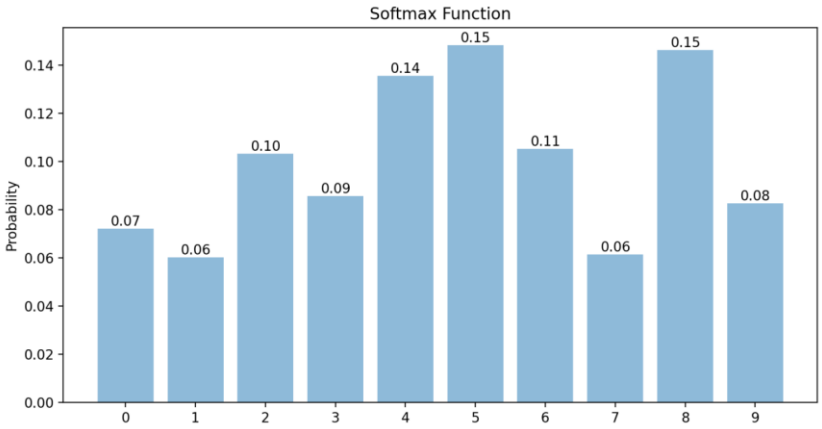
IV.**应用 softmax 函数得到每个时间步的预测概率分布 ：**

**代码：**

```
ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
```

**数学公式:**

$$p_t = \frac{\exp(y_t)}{\sum\exp(y_t)}$$




这个公式将原始输出$y_t$转换成一个概率分布，所有可能输出的概率和为 1。

V.**累计计算并记录交叉熵损失：**

**代码：**

```
loss += -np.log(ps[t][targets[t], 0])
```

**数学公式:**

$$L = -\sum \log\left(p_{t,target}\right)$$

其中 $p_{t,target}$ 是模型在时间步 $t$ 预测正确目标的概率， $target$ 是目标输出的索引。

④ **反向传播**

反向传播是神经网络训练过程中用于计算梯度的一种方法，以便对模型的参数进行更新。迭代训练来降低损失度的反向传播算法，具体如下：

**I.计算输出层梯度 $dy$：**

**代码:**

```
dy = np.copy(ps[t])
dy[targets[t]] -= 1
```

**数学公式：**

$$dy = p_t - 1\{target\}$$

其中 $p_t$ 是前向传播中计算的预测概率，$1\{target\}$ 是正确类别的指示函数。

**II.计算与 $W_{hy}$ 和 $b_y$ 相关的梯度：**

**代码:**

```
dWhy += np.dot(dy, hs[t].T)
dby += dy
```

**数学公式：**

$$\frac{\partial L}{\partial W_{hy}} += dy \cdot h_t^T$$

$$\frac{\partial L}{\partial b_y} += dy$$

**III.计算下一个时间步的隐藏层梯度 $dh$：**

**代码：**

```
dh = np.dot(Why.T, dy) + dhnext
```

**数学公式：**

$$dh = W_{hy}^T \cdot dy + dh_{next}$$

**IV.通过应用导数链式法则计算激活函数的原始梯度 $dhraw$：**

**代码：**

```
dhraw = (1 - hs[t] * hs[t]) * dh
```

**数学公式：**

$$dhraw = (1 - h_t^2) \cdot dh$$

这一步是应用了 $tanh$ 函数的导数 $1 - tanh^2(x)$。

**V.计算与 $W_{xh}$，$W_{hh}$，和 $b_h$ 相关的梯度：**

**代码：**

```
dbh += dhraw
dWxh += np.dot(dhraw, xs[t].T)
dWhh += np.dot(dhraw, hs[t-1].T)
```

**数学公式：**

$$\frac{\partial L}{\partial b_h} += dhraw$$

$$\frac{\partial L}{\partial W_{xh}} += dhraw \cdot x_t^T$$

$$\frac{\partial L}{\partial W_{hh}} += dhraw \cdot h_{t-1}^T$$

**VI.更新下一个时间步的 $dhnext$：**

**代码：**

```
dhnext = np.dot(Whh.T, dhraw)
```

**数学公式：**

$$dh_{next} = W_{hh}^T \cdot dhraw$$

这些步骤中的梯度计算反映了网络中损失对于各个参数的敏感度，即对这些参数的微小改变将如何影响整体的损失。这些梯度随后会被用于参数的更新，通常通过梯度下降或其他优化算法来减小网络的损失，并提高模型的性能。

求得各个参数的梯度后，使用梯度下降法更新权重，参数的更新可以写作：

$$\theta := \theta - \alpha \cdot \nabla_\theta J(\theta)$$

$\theta$ 代表模型参数（Wxh, Whh, Why, bh, by）

$\alpha$ 代表学习率（learning_rate）

$\nabla_\theta J(\theta)$ 代表损失函数 J 关于参数θ的梯度（dWxh, dWhh, dWhy, dbh, dby）

因此，对于每个参数和梯度对 (param, dparam)，参数更新的数学表示将是：

$$\text{param} := \text{param} - \alpha \cdot \text{dparam}$$

具体的参数更新为：

$$W_{xh} := W_{xh} - \alpha \cdot dW_{xh}$$

$$W_{hh} := W_{hh} - \alpha \cdot dW_{hh}$$

$$W_{hy} := W_{hy} - \alpha \cdot dW_{hy}$$

$$b_h := b_h - \alpha \cdot db_h$$

$$b_y := b_y - \alpha \cdot db_y$$

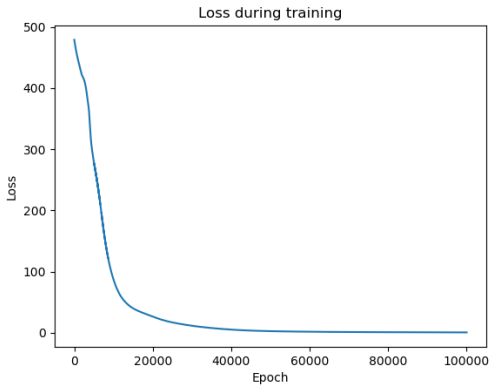
这里的“:=”表示更新操作。简单来说，每个参数都会减去其梯度乘以学习率，这是一个基本的梯度下降步骤，用于最小化损失函数，改善模型的预测性能。

⑤ **英文测试**

本模型因手敲实现，需要初步测试模型的可行性再去实现诗歌集的训练。

这是使用较为简单的英文文章进行初步测试，数据为《哈默雷特》中的片段。

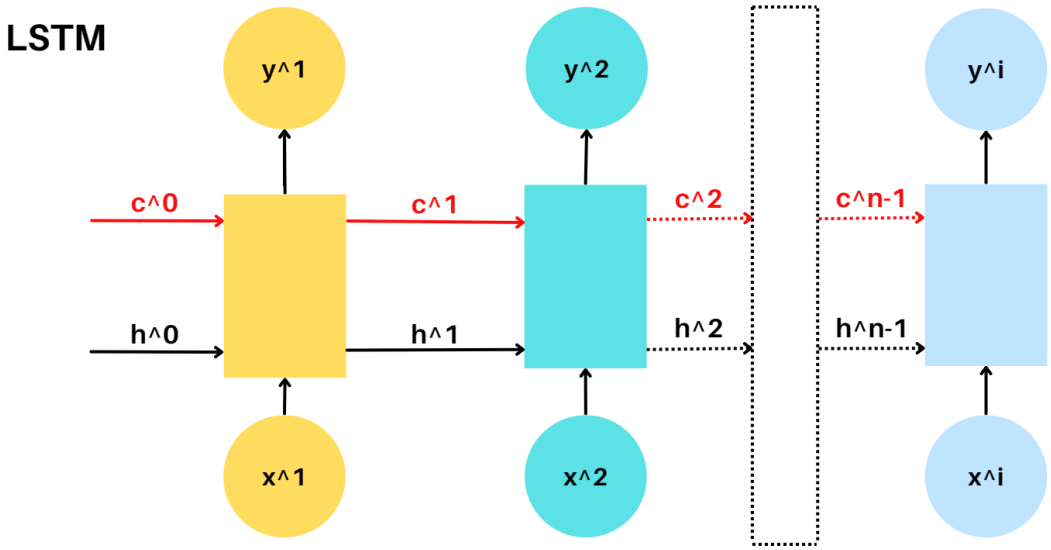
迭代十万次后的 loss 曲线如下：




此曲线是较为平滑的收敛曲线，因此可以初步判断模型的可行性。

**（2）LSTM 模型**

LSTM 从被设计之初就被用于解决一般递归神经网络中普遍存在的**长期依赖问题**，使用 LSTM 可以有效的传递和表达长时间序列中的信息并且不会导致长时间前的有用信息被忽略（遗忘）。与此同时，LSTM 还可以解决 RNN 中的梯度消失/爆炸问题。LSTM 的结构图如下所示：




本课题之所以使用基于 pytorch 的 LSTM 模型与之对比，是因为简单 RNN 处理诗歌时会有一定的局限性。

LSTM 与传统 RNN 的不同点在于$C_t$：$C_t$在时间步 $t$ 的单元状态，这是 LSTM 记忆部分的核心，负责在网络中携带长期信息。

公式的详细如下：

① 输入门：$\sigma(U_i h_{t-1} + W_i x_t + b_i)$ 决定有多少新输入信息允许影响单元状态。

② 遗忘门：$\sigma(U_f h_{t-1} + W_f x_t + b_f)$ 决定有多少旧的单元状态信息应该保留或忘记。

③ 输出门：$\sigma(U_o h_{t-1} + W_o x_t + b_o)$ 决定有多少单元状态的信息应该被输出到隐藏状态 $h_t$。

④ 单元状态更新：$C_t =$ 输入门的输出与新的候选值 $\tanh(U_x h_{t-1} + W_x x_t + b_x)$ 的哈达玛积，加上遗忘门的输出与先前单元状态 $C_{t-1}$ 的哈达玛积。

⑤ 隐藏状态更新：$h_t = \tanh(C_t) \odot$ 输出门的输出，这决定了最终的输出。

⑥ 最终输出：$y_t = f(V_y h_t + b_y)$，其中 $f$ 可以是另一个激活函数，例

如 softmax，如果 LSTM 用于分类任务。

LSTM 的设计可以帮助解决传统 RNN 面临的长期依赖问题，因为它可以在很长的时间序列中保持信息，而不会遇到梯度消失或梯度爆炸的问题。

在本课设中，LSTM 使用 pytorch 框架构建，目的是与上述的手敲 RNN 进行比较分析。

核心框架为：

I.定义了一个 LSTM 模型类，包括词嵌入层、LSTM 层和全连接层，用于生成诗歌的下一个词。

II.实例化模型，并设置词汇表大小、嵌入维度和 LSTM 隐藏层维度。

III.定义交叉熵损失函数和 Adam 优化器，用于模型训练。

IV.设置训练的周期数和将数据转换为 PyTorch 张量。

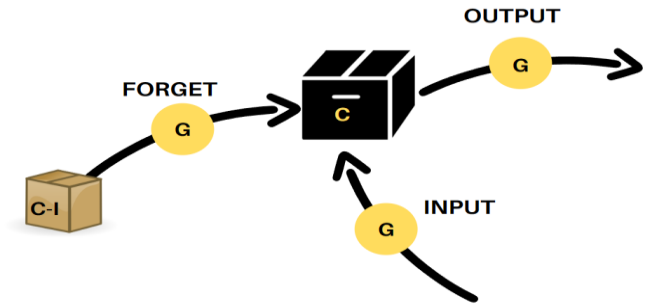
V.创建数据集和数据加载器，用于批量处理训练和测试数据。

核心代码如下：

```
class LSTMModel(nn.Module):
    # 构造函数，定义模型的基本结构
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        # 调用父类 nn.Module 的构造函数
        super(LSTMModel, self).__init__()
        # 定义嵌入层，将词汇映射为嵌入向量
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        # 定义 LSTM 层，embedding_dim 是输入尺寸，
      hidden_dim 是 LSTM 单元数
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_firs
      t=True)
            # 定义一个全连接层，将 LSTM 的输出映射到词汇空间
            self.fc = nn.Linear(hidden_dim, vocab_size)
        # 前向传播函数，定义模型如何处理输入数据
        def forward(self, x):
            # 将输入数据 x 通过嵌入层，转换为嵌入向量
            x = self.embedding(x)
            # 将嵌入向量传递给 LSTM 层进行处理
            lstm_out, _ = self.lstm(x)
            # 取 LSTM 输出的最后一个时间步的数据，传递给全连接层
            out = self.fc(lstm_out[:, -1, :])
            return out
```

**（3）循环神经网络总结**

循环神经网络（Recurrent Neural Networks, RNNs）是一类用于处理序列数据的神经网络。与传统的神经网络不同，它们可以处理变长的输入序列，并且具有内部状态的概念，使得它们能够记住并利用历史信息。简单来说，他会依赖于之前的历史数据。框架图如下：




## 2、数据处理设计

对于数据集中的汉字，可以分为两种处理方式，即**按字和按词处理**。初步分析下，按字拆分会有一定的局限性，例如一些具有实例的名词被拆。因此本课设的 RNN 针对按词和按字进行效果比较。

**（1）数据集按字处理**

**① 读取文件：**打开并且读取名为 data.csv 的文件，内部是古诗集。

**② 文本清洗:**

```
[re.sub(r'[^\w\s]', '', poem)]
[re.sub(r'[\s\n]', '', poem)]
```

使用正则表达式移除所有非单词字符和空白字符以外的字符，移除所有空白字符，包括空格和换行符。

**③ 建立词汇表：**所有数据去重并且计算大小作为神经网络层数。

**④ 创建字符索引映射：**

加密映射：

```
{'扑': 0,
 '荻': 1,
 '烛': 2,
 '壁': 3,
 '镂': 4,
 '登': 5,
 '窗': 6,
 '乱': 7,
 '溯': 8,
```

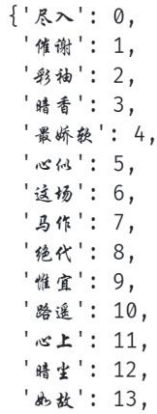
解密映射：

```
{0: '扑',
 1: '荻',
 2: '烛',
 3: '壁',
 4: '镂',
 5: '登',
 6: '窗',
 7: '乱',
 8: '溯',
 9: '姮',
 10: '沙',
```

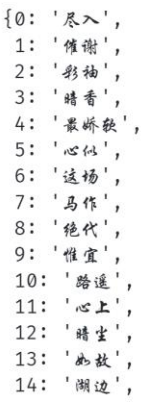
（2）**数据集按词处理**

使用 python 的 jieba 库进行分词，其余方法与上述一致。

加密映射：




解密映射：




## 3、工程文件设计

**（1）保存模型**

**① 手敲 RNN 模型的保存**

**I.保存权重和偏置**

使用 numpy.save 函数保存了 RNN 的权重和偏置参数到 NumPy 的 .npy 文件格式中。这种格式方便于之后的加载和使用。其中，Wxh, Whh, Why 是网络的权重矩阵，bh 和 by 是偏置项。这些参数是 RNN 模型的核心，控制着网络如何处理输入和记忆信息。

**II.保存字符到索引和索引到字符的映射**

pickle 用于序列化和反序列化一个 Python 对象结构。

将 char_to_index 字典保存到 char_to_index.pickle 里。

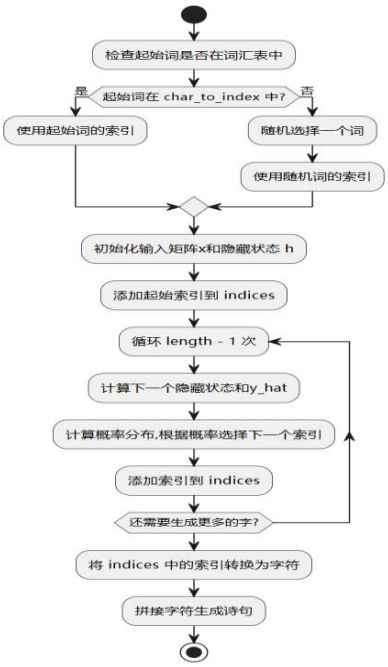
将 index_to_char 字典保存到 index_to_char.pickle 里。

**② LSTM 的保存**

使用了 PyTorch 库中的 torch.save 函数来保存一个模型的状态字典（state dictionary）。具体来说，它是用于保存一个使用 PyTorch 框架构建的循环神经网络。

（2）**藏头诗函数**

generate_line 函数是用于生成藏头诗的一行。这个函数接受一个起始字符和一个长度参数，然后生成一个以该起始字符开头的文本序列。内部设计流程图如下图所示：




**具体步骤如下：**

**① 检查起始字符：**

首先检查提供的起始字符是否在您的字符到索引的映射（char_to_index）中。如果不在，函数会从词汇表（vocab）中随机选择一个字符作为起始字符。这确保了无论输入字符是否在词汇表中，函数都能正常运行。

**② 生成循环：**

函数进入一个循环，因为起始字符已经确定。在每次循环中，它执行以下步骤：

**I.更新隐藏状态：**使用当前的输入 x 和前一时刻的隐藏状态 h 来更新当前时刻的隐藏状态。这是通过计算 Wxh 与 x 的点积、Whh 与 h 的点积，加上偏置 bh，然后应用 tanh 激活函数来完成的。

**II.生成下一个字符：**计算下一个字符的输出分布 y_hat，这是通过计算 Why 与 h 的点积加上偏置 by 来实现的。应用 softmax 函数（这里使用 exp 和求和实现）来得到下一个字符的概率分布。根据概率分布 prob，随机选择下一个字符的索引。将选择的下一个字符的索引转换为 one-hot 编码，作为下一次循环的输入。

**③ 构建生成的文本：**

函数完成循环后，它使用 index_to_char 映射将字符索引转换回字符，并将它们拼接成一个字符串。然后，它将生成的文本序列的第一个字符替换为原始的起始字符，确保生成的文本以给定的起始字符开头。

（3）**前后端设计**

① **后端：**使用 python 加载模型参数和加密解密映射，并且使用 flask 库来建立与前端页面的接口。接口的端口号为：127.0.0.5000 和 127.0.0.8080。

② **前端：**使用简单 html,css,js 设计前端界面，完成模型的可视化处理，用于生成古诗。前端的设计图如下：

# 五、调试分析

因实现的是基于 RNN 的文本创作，对准确率不做要求，但需要考虑模型的损失率，本模型使用的是交叉熵函数计算损失度。

模型的 loss 函数若收敛，则可以称模型有效。

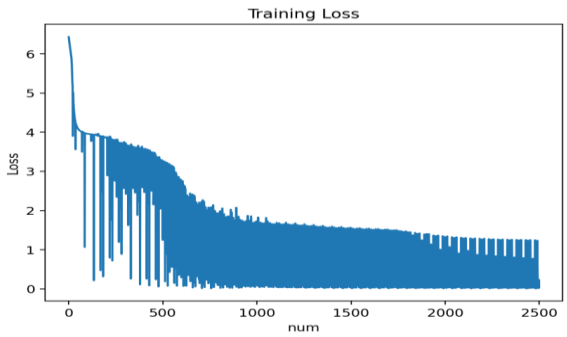
## 1、 RNN 的分词与分字损失度对比

### （1）分字测试结果

通过一系列测试分析，当模型在学习率为 0.05，隐藏层数为 25 时效果较好。

纯手写 RNN 缘故，没有较好的优化器适配，需要大量的迭代次数，本次测试总共迭代 50000 次。

**以下是分字的 loss 收敛结果:**

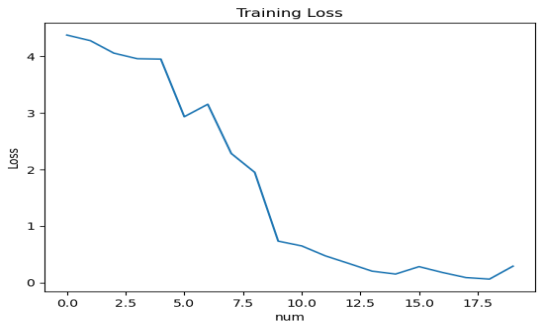


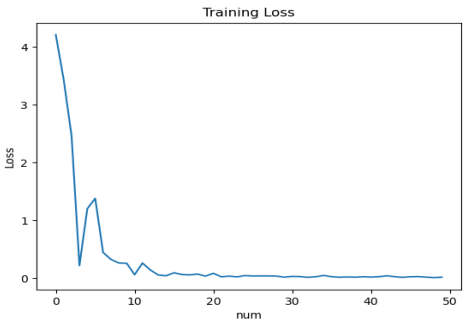

由图可知，损失存在很大波动，但随着时间推移逐渐减少。这可能是因为批量训练过程中固有的变异性，不同子集的数据可能有略微不同的特性。在训练过程的末尾，损失似乎趋于稳定，表明模型从新数据中学习的内容不多。这可能表示模型已接近其能力上限，或者表明需要降低学习率以获得进一步改进。

总之，loss 曲线最终接近收敛，但是波动幅度较大，与原先的猜想一致，按字分数据并非最优解。

### （2）分词测试结果

由于每次 jieba 对文本的分词不同，因此 loss 曲线也会相应不同，本报告中存放两种 jieba 取词时的 loss 曲线，结果如下：



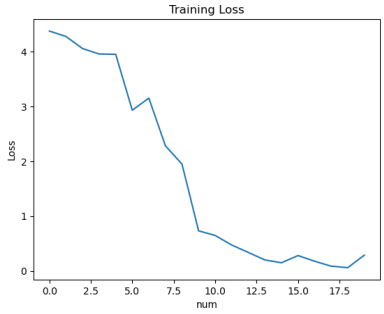




结果可知，分词处理的 loss 曲线波动明显降低，由此得出结论：分词处理藏头诗模型方面优于分字处理。此结论也可以进一步说明：处理对中文的自然语言模型时，**以词划分的效果更佳。**

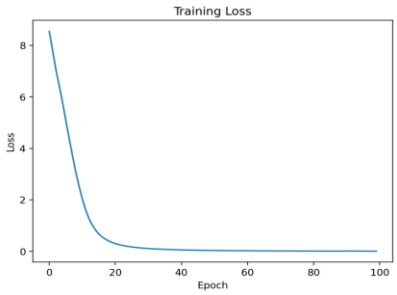
## 2、 RNN 与 LSTM 损失度对比

当然，手敲的 RNN 虽然使用了藏头诗功能并且 loss 函数收敛，但是曲线并不平滑，甚至也存在一些波动形况，显然传统 RNN 在处理诗集时会出现一些问题。

**RNN：**




**LSTM：**

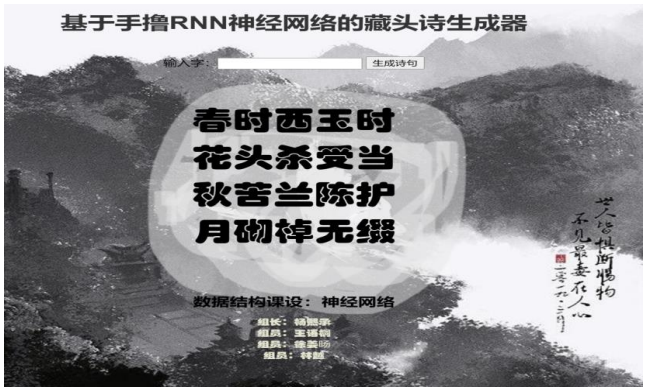



因此考虑基于 pytorch 实现 LSTM 模型来进行比较分析。由下图可知，因 pytorch 的库模块强大并且支持优化器和学习率自适应，在 loss 曲线上表现出完美的平滑收敛，同时也基于 LSTM 更复杂的数学逻辑结构，这种长短期记忆模型在处理诗歌集方面更为优秀。

# 六、测试结果

## 1、RNN

下图为 RNN 输入“春花秋月”时的结果。




## 2、LSTM

下图为 LSTM 输入“春花秋月”时的结果。


## 3、遇到的困难

在生成诗歌时，遇到生成的诗歌有标点，空格，没有完整一句就换行等等的问题，经过调试代码查看词汇表发现，误将空格，换行符这些没有具体意义的字符存放进去导致的，解决方案为使用正则表达式对诗句进行进一步的清洗。

在生成诗歌时，总是出现“注”这个字，使得诗歌整体质量不佳，后来得知是数据集里面的诗歌后面有注解导致的。解决方案还是用正则表达式将小括号中的注解删除。

# 七、用户使用说明

**1、简要使用说明：**

(1) app1.py 文件为 RNN 模型的后端接口文件,运行后浏览器后输入 127.0.0.5000 即可打开前端界面，输入藏头诗的开头，生成一副完整的诗歌。

(2) app2.py 文件为 LSTM 模型的后端接口文件,运行后浏览器后输入 127.0.0.8080 即可打开前端界面，输入藏头诗的开头，生成一副完整的诗歌。

2、**具体使用说明：**

这是基于神经网络的藏头诗生成器。顾名思义，就是用户输入相关汉字，系统生成以相关汉字开头的诗句，主要满足于人们的娱乐目的。

当用户想要输入单个字生成藏头诗时，输入后按回车即可；若是多个字，要求每两个字中间加入一个空格，再按回车。若用户感觉不满意，则再按一下回车，即可生成新的藏头诗。

**3、 代码主要文件树即功能(简化版)：**

**Peo**

**——RNNci.py　按词划分的手敲 RNN**

**——RNNzi.py　按字划分的手敲 RNN**

**——Demo1.ipynb　RNN 的可视化，藏头诗生成，以及保存函数**

**——Demo2.ipynb　LSTM 的可视化，藏头诗生成，以及保存函数**

**——Peo1　　基于 RNN 的前端界面**

**———index1.html**

**———app1.py　flask 接口**

**———model1　模型参数**

**——Peo2　　基于 LSTM 的前端界面**

**———index2.html**

**———app2.py　flask 接口**

**———model2　模型参数**

# 八、总结

在开始阶段，我们明确了藏头诗创作的任务目标，并对其背景进行了深入的研究。藏头诗是一种具有特定格式和要求的文学形式，要求每行诗的第一个字或词合起来，能形成一句话或者传达某种特定的信息。

数据收集与预处理中，我们重点进行了藏头诗数据的收集，并进行了必要的预处理工作，如数据清洗、格式统一等，为后续的模型训练做准备。

基于对任务需求和数据特点的分析，我们选择了 RNN 作为基础架构。利用预处理后的数据，我们对模型进行了训练。在训练过程中，我们不断调整超参数、优化损失函数，并采用有效的学习策略，以使模型性能达到最优。

在模型训练完成后，我们对生成的结果进行了评估。通过与真实数据集进行对比，我们评估了模型的准确率、召回率等指标，并根据评估结果对模型进行了进一步的优化。

然后，我们将训练好的模型集成到一个应用程序中，用户可以通过输入主题，自动生成藏头诗。这个应用程序不仅是一个学术研究的展示，也可以作为娱乐工具，供人们欣赏和创作藏头诗。

回顾整个课程设计，是我们小组四人十多天来共同努力和老师认真指导的结果。在学术上，做到了数据结构的实践应用和神经网络的深度学习；在协作上，做到了小组成员的团结一致、共同努力，增强了我们同学间的分工合作能力。我相信这次的课程设计对我们四人未来的学习和发展都具有重要意义。

最后，再次感谢小组成员的共同努力和老师的辛勤指导，谢谢！

# 附录

我的仓库链接：https://github.com/Naasi-LF/poetry

代码仅放主要代码，具体代码详见 github 仓库

**RNNzi.py**

```
import re
import numpy as np
import csv
import matplotlib.pyplot as plt
import random
poems = []
with open('data.csv', 'r', encoding='utf-8') as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
            poem_content = row['content']
            poems.append(poem_content)

poems = [re.sub(r'[^\w\s]', '', poem) for poem in poems]
poems = [re.sub(r'[\s\n]', '', poem) for poem in poems]

# 去重
vocab = set("".join(poems))

# 长度就是模型接受的大小
vocab_size = len(vocab)
# 加密解密
char_to_index = {char: i for i, char in enumerate(vocab)}
index_to_char = {i: char for i, char in enumerate(vocab)}

# 模型参数初始化
input_size = vocab_size
hidden_size = 5 # 可以调整这个值以优化模型
output_size = vocab_size
rate = 0.01

Wxh = np.random.randn(hidden_size, input_size) * 0.01
Whh = np.random.randn(hidden_size, hidden_size) * 0.01
Why = np.random.randn(output_size, hidden_size) * 0.01
bh = np.zeros((hidden_size, 1))
by = np.zeros((output_size, 1))
```

```
# 前向和后向传播
def forward_backward(inputs, targets, hprev):
    xs, hs, ys, ps = {}, {}, {}, {}
    # RNN 通过上一次的 h 与 x 共同作用
    hs[-1] = np.copy(hprev)
    loss = 0

    for t in range(len(inputs)):
        xs[t] = np.zeros((input_size, 1))
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + b
h)
        ys[t] = np.dot(Why, hs[t]) + by
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
        loss += -np.log(ps[t][targets[t], 0])

    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zer
os_like(Why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])

    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1
        dWhy += np.dot(dy, hs[t].T)
        dby += dy
        dh = np.dot(Why.T, dy) + dhnext
        dhraw = (1 - hs[t] * hs[t]) * dh
        dbh += dhraw
        dWxh += np.dot(dhraw, xs[t].T)
        dWhh += np.dot(dhraw, hs[t-1].T)
        dhnext = np.dot(Whh.T, dhraw)

    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

# 每五个为一个序列
length = 5
# 训练参数
num = 500000  # 迭代次数
patience = 800   # 耐心值

def train(data, num, patience=500):
    global Wxh, Whh, Why, bh, by  # 全局
```

```
    lowest_loss = np.inf # 迭代找最小
    best = {}
    counter = 0

    n, p = 0, 0
    hprev = np.zeros((hidden_size, 1))
    losses = [] # 记录loss
    lossess = []
    for i in range(num):
        if p + length + 1 >= len(data) or n == 0:
            hprev = np.zeros((hidden_size, 1))
            p = 0

        inputs = [char_to_index[data[p]]]
        targets = [char_to_index[ch] for ch in data[p+1:p+length+1]]

        loss, dWxh, dWhh, dWhy, dbh, dby, hprev = forward_backward(inp
uts, targets, hprev)

        # 梯度下降
        for param, dparam in zip([Wxh, Whh, Why, bh, by], [dWxh, dWhh,
 dWhy, dbh, dby]):
            param += -rate * dparam

        p += length
        n += 1
        if n%10 ==0:
            lossess.append(loss)
        if n % 200 == 0:
            print(f"Epoch {n}, Loss: {loss}")
            losses.append(loss)

        if loss < lowest_loss:
            lowest_loss = loss
            best = {
                'Wxh': Wxh.copy(),
                'Whh': Whh.copy(),
                'Why': Why.copy(),
                'bh': bh.copy(),
                'by': by.copy()
            }
            counter = 0
        else:
```

```
            counter += 1

        # if counter > patience:
        #     print(f"Early stopping at iteration {n}, Lowest loss: {l
owest_loss}")
        #     break

    Wxh, Whh, Why, bh, by = best.values()
    plt.plot(range(len(losses)),losses)
    plt.xlabel('num')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.show()
    plt.scatter(range(len(lossess)), lossess, color='red', marker='o
')
    plt.xlabel('num')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.show()
    return best, lowest_loss

data = ''.join(poems)
best_params, lowest_loss = train(data, num, patience)

print(f"lowest loss: {lowest_loss}")

def generate_line(start_char, length=5):

    if start_char not in char_to_index:
        random_char = np.random.choice(list(vocab))
        start_index = char_to_index[random_char]
    else:
        start_index = char_to_index[start_char]

    x = np.zeros((input_size, 1))
    x[start_index] = 1
    h = np.zeros((hidden_size, 1))
    indices = [start_index]

    for _ in range(length - 1):
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
        y_hat = np.dot(Why, h) + by
        prob = np.exp(y_hat) / np.sum(np.exp(y_hat))
        next_char = np.random.choice(range(vocab_size), p=prob.ravel
```

```
())
        indices.append(next_char)
        x = np.zeros((input_size, 1))
        x[next_char] = 1

    generated_poem = "".join(index_to_char[i] for i in indices)
    return start_char + generated_poem[1:]


chars = input("请输入藏头诗的头: ").split(' ')

for char in chars:
    generated_line = generate_line(char, length=5)
    print(generated_line)
```

RNNci.py

```
 import re
import numpy as np
import csv
import matplotlib.pyplot as plt
import random
import jieba
poems = []

with open('data.csv', 'r', encoding='utf-8') as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
            poem_content = row['content']
            poems.append(poem_content)

# 使用jieba分词
# 使用jieba分词，并跳过标点符号和换行符
poems = [list(filter(lambda x: x not in ['\n', '，', '。', '！', '？',
'、'], jieba.cut(poem))) for poem in poems]


# 将列表的列表展平，得到词列表
words = [word for poem in poems for word in poem]

# 去重以构建词汇表
vocab = set(words)
# 去重
# vocab = set("".join(poems))
```

```
# 长度就是模型接受的大小
vocab_size = len(vocab)
# 加密解密
char_to_index = {char: i for i, char in enumerate(vocab)}
index_to_char = {i: char for i, char in enumerate(vocab)}

# 模型参数初始化
input_size = vocab_size
hidden_size = 5 # 可以调整这个值以优化模型
output_size = vocab_size
rate = 0.1

Wxh = np.random.randn(hidden_size, input_size) * 0.01
Whh = np.random.randn(hidden_size, hidden_size) * 0.01
Why = np.random.randn(output_size, hidden_size) * 0.01
bh = np.zeros((hidden_size, 1))
by = np.zeros((output_size, 1))

# 前向和后向传播
def forward_backward(inputs, targets, hprev):
    xs, hs, ys, ps = {}, {}, {}, {}
    # RNN 通过上一次的 h 与 x 共同作用
    hs[-1] = np.copy(hprev)
    loss = 0

    for t in range(len(inputs)):
        xs[t] = np.zeros((input_size, 1))
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + b
h)
        ys[t] = np.dot(Why, hs[t]) + by
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
        loss += -np.log(ps[t][targets[t], 0])

    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zer
os_like(Why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])

    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1
        dWhy += np.dot(dy, hs[t].T)
        dby += dy
```

```
        dh = np.dot(Why.T, dy) + dhnext
        dhraw = (1 - hs[t] * hs[t]) * dh
        dbh += dhraw
        dWxh += np.dot(dhraw, xs[t].T)
        dWhh += np.dot(dhraw, hs[t-1].T)
        dhnext = np.dot(Whh.T, dhraw)

    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

# 每五个为一个序列
length = 5
# 训练参数
num = 100000  # 迭代次数
patience = 800   # 耐心值

def train(data, num, patience=500):
    global Wxh, Whh, Why, bh, by  # 全局

    lowest_loss = np.inf # 迭代找最小
    best = {}
    counter = 0

    n, p = 0, 0
    hprev = np.zeros((hidden_size, 1))
    losses = [] # 记录 loss
    lossess = []
    for i in range(num):
        if p + length + 1 >= len(data) or n == 0:
            hprev = np.zeros((hidden_size, 1))
            p = 0

        inputs = [char_to_index[data[p]]]
        targets = [char_to_index[ch] for ch in data[p+1:p+length+1]]

        loss, dWxh, dWhh, dWhy, dbh, dby, hprev = forward_backward(inp
uts, targets, hprev)

        # 梯度下降
        for param, dparam in zip([Wxh, Whh, Why, bh, by], [dWxh, dWhh,
 dWhy, dbh, dby]):
            param += -rate * dparam

        p += length
        n += 1
```

```
        if n%10 ==0:
            lossess.append(loss)
        if n % 2000 == 0:
            print(f"Epoch {n}, Loss: {loss}")
            losses.append(loss)

        if loss < lowest_loss:
            lowest_loss = loss
            best = {
                'Wxh': Wxh.copy(),
                'Whh': Whh.copy(),
                'Why': Why.copy(),
                'bh': bh.copy(),
                'by': by.copy()
            }
            counter = 0
        else:
            counter += 1

        # if counter > patience:
        #     print(f"Early stopping at iteration {n}, Lowest loss: {l
owest_loss}")
        #     break

    Wxh, Whh, Why, bh, by = best.values()
    plt.plot(range(len(losses)),losses)
    plt.xlabel('num')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.show()
    plt.scatter(range(len(lossess)), lossess, color='red', marker='o
')
    plt.xlabel('num')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.show()
    return best, lowest_loss


best_params, lowest_loss = train(words, num, patience)

print(f"Lowest loss: {lowest_loss}")
def generate_line(start_word, length=5):
```

```
    if start_word not in char_to_index:
        # 如果起始词不在词汇表中，随机选择一个词
        random_word = np.random.choice(list(vocab))
        start_index = char_to_index[random_word]
    else:
        start_index = char_to_index[start_word]

    x = np.zeros((input_size, 1))
    x[start_index] = 1
    h = np.zeros((hidden_size, 1))
    indices = [start_index]

    for _ in range(length - 1):
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
        y_hat = np.dot(Why, h) + by
        prob = np.exp(y_hat) / np.sum(np.exp(y_hat))
        next_index = np.random.choice(range(vocab_size), p=prob.ravel
())
        indices.append(next_index)
        x = np.zeros((input_size, 1))
        x[next_index] = 1

    generated_poem = "".join(index_to_char[i] for i in indices)
    return generated_poem

# 输入藏头词列表
acrostic_input = input("请输入藏头诗的头: ").split(' ')

for word in acrostic_input:
    generated_line = generate_line(word, length=5)
    print(generated_line)
```

demo1.ipynb

```
 import re
import numpy as np
import csv
import matplotlib.pyplot as plt
import random
import jieba
poems = []
with open('data.csv', 'r', encoding='utf-8') as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
```

```
        if random.random() < 0.1:
            poem_content = row['content']
            poems.append(poem_content)
# 使用jieba分词，并跳过标点符号和换行符
poems = [list(filter(lambda x: x not in ['\n', ', ', '。', '! ', '? ',
'、'], jieba.cut(poem))) for poem in poems]


# 将列表的列表展平，得到词列表
words = [word for poem in poems for word in poem]

# 去重以构建词汇表
vocab = set(words)

# 长度就是模型接受的大小
vocab_size = len(vocab)
# 加密解密
char_to_index = {char: i for i, char in enumerate(vocab)}
index_to_char = {i: char for i, char in enumerate(vocab)}

# 模型参数初始化
input_size = vocab_size
hidden_size = 5 # 可以调整这个值以优化模型
output_size = vocab_size
rate = 0.05

Wxh = np.random.randn(hidden_size, input_size) * 0.01
Whh = np.random.randn(hidden_size, hidden_size) * 0.01
Why = np.random.randn(output_size, hidden_size) * 0.01
bh = np.zeros((hidden_size, 1))
by = np.zeros((output_size, 1))

# 前向和后向传播
def forward_backward(inputs, targets, hprev):
    xs, hs, ys, ps = {}, {}, {}, {}
    # RNN通过上一次的h与x共同作用
    hs[-1] = np.copy(hprev)
    loss = 0

    for t in range(len(inputs)):
        xs[t] = np.zeros((input_size, 1))
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + b
h)
```

```
        ys[t] = np.dot(Why, hs[t]) + by
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
        loss += -np.log(ps[t][targets[t], 0])

    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zer
os_like(Why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])

    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1
        dWhy += np.dot(dy, hs[t].T)
        dby += dy
        dh = np.dot(Why.T, dy) + dhnext
        dhraw = (1 - hs[t] * hs[t]) * dh
        dbh += dhraw
        dWxh += np.dot(dhraw, xs[t].T)
        dWhh += np.dot(dhraw, hs[t-1].T)
        dhnext = np.dot(Whh.T, dhraw)

    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

# 每五个为一个序列
length = 25
# 训练参数
num = 100000  # 迭代次数
patience = 800   # 耐心值

def train(data, num, patience=500):
    global Wxh, Whh, Why, bh, by  # 全局

    lowest_loss = np.inf # 迭代找最小
    best = {}
    counter = 0

    n, p = 0, 0
    hprev = np.zeros((hidden_size, 1))
    losses = [] # 记录 loss
    lossess = []
    for i in range(num):
        if p + length + 1 >= len(data) or n == 0:
            hprev = np.zeros((hidden_size, 1))
            p = 0
```

```
        inputs = [char_to_index[data[p]]]
        targets = [char_to_index[ch] for ch in data[p+1:p+length+1]]

        loss, dWxh, dWhh, dWhy, dbh, dby, hprev = forward_backward(inp
uts, targets, hprev)

        # 梯度下降
        for param, dparam in zip([Wxh, Whh, Why, bh, by], [dWxh, dWhh,
 dWhy, dbh, dby]):
            param += -rate * dparam

        p += length
        n += 1
        if n%10 ==0:
            lossess.append(loss)
        if n % 5000 == 0:
            print(f"Epoch {n}, Loss: {loss}")
            losses.append(loss)

        if loss < lowest_loss:
            lowest_loss = loss
            best = {
                'Wxh': Wxh.copy(),
                'Whh': Whh.copy(),
                'Why': Why.copy(),
                'bh': bh.copy(),
                'by': by.copy()
            }
            counter = 0
        else:
            counter += 1

        # if counter > patience:
        #     print(f"Early stopping at iteration {n}, Lowest loss: {l
owest_loss}")
        #     break

    Wxh, Whh, Why, bh, by = best.values()
    plt.plot(range(len(losses)),losses)
    plt.xlabel('num')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.show()
```

```
    plt.scatter(range(len(lossess)), lossess, color='red', marker='o
')
    plt.xlabel('num')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.show()
    return best, lowest_loss


best_params, lowest_loss = train(words, num, patience)

print(f"lowest loss: {lowest_loss}")
def generate_line(start_word, length=5):

    if start_word not in char_to_index:
        # 如果起始词不在词汇表中，随机选择一个词
        random_word = np.random.choice(list(vocab))
        start_index = char_to_index[random_word]
    else:
        start_index = char_to_index[start_word]

    x = np.zeros((input_size, 1))
    x[start_index] = 1
    h = np.zeros((hidden_size, 1))
    indices = [start_index]

    for _ in range(length - 1):
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
        y_hat = np.dot(Why, h) + by
        prob = np.exp(y_hat) / np.sum(np.exp(y_hat))
        next_index = np.random.choice(range(vocab_size), p=prob.ravel
())
        indices.append(next_index)
        x = np.zeros((input_size, 1))
        x[next_index] = 1

    generated_poem = "".join(index_to_char[i] for i in indices)
    return generated_poem

import numpy as np
import pickle

np.save('Wxh.npy', Wxh)
np.save('Whh.npy', Whh)
```

```
np.save('Why.npy', Why)
np.save('bh.npy', bh)
np.save('by.npy', by)

with open('char_to_index.pickle', 'wb') as handle:
    pickle.dump(char_to_index, handle, protocol=pickle.HIGHEST_PROTOC
OL)

with open('index_to_char.pickle', 'wb') as handle:
    pickle.dump(index_to_char, handle, protocol=pickle.HIGHEST_PROTOC
OL)
```

**demo2.ipynb**

```
import pandas as pd

file_path = 'data.csv'
data = pd.read_csv(file_path)

data.head()

import jieba

def jieba_tokenizer(text):
    # 使用 jieba 进行分词
    words = jieba.cut(text)
    # 过滤掉一些无意义的字符（可根据需要调整）
    filtered_words = [word for word in words if len(word) > 1 and word
 != '\r\n']
    return filtered_words

# 应用分词到数据集
data['tokenized_content'] = data['content'].apply(jieba_tokenizer)

# 显示分词结果
print(data['tokenized_content'].head())

all_words = [word for tokens in data['tokenized_content'] for word in
 tokens]

unique_words = set(all_words)

word_to_num = {word: i for i, word in enumerate(unique_words)}
num_to_word = {i: word for word, i in word_to_num.items()}

data['numerical_sequences'] = data['tokenized_content'].apply(lambda
```

```
 tokens: [word_to_num[word] for word in tokens])

len(unique_words), data['numerical_sequences'].head()

import numpy as np

# 参数初始化
vocab_size = len(word_to_num)  # 词汇表大小
hidden_size = 100  # 隐藏层神经元数量
learning_rate = 1e-1

# 权重初始化
Wxh = np.random.randn(hidden_size, vocab_size)*0.01  # 输入到隐藏层
Whh = np.random.randn(hidden_size, hidden_size)*0.01  # 隐藏层到隐藏层

Why = np.random.randn(vocab_size, hidden_size)*0.01  # 隐藏层到输出层
bh = np.zeros((hidden_size, 1))  # 隐藏层偏置
by = np.zeros((vocab_size, 1))   # 输出层偏置

import numpy as np
sequence_length = 20  # 选择序列长度
features = []
labels = []

for poem in data['numerical_sequences']:
    for i in range(len(poem) - sequence_length):
        # 提取长度为sequence_length的序列和下一个词作为标签
        seq = poem[i:i + sequence_length]
        label = poem[i + sequence_length]
        features.append(seq)
        labels.append(label)

features = np.array(features)
labels = np.array(labels)

# 数据分割
train_size = int(len(features) * 0.8)
train_features = features[:train_size]
train_labels = labels[:train_size]
test_features = features[train_size:]
test_labels = labels[train_size:]

print(features)
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
# 定义模型
class LSTMModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(LSTMModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=Tr
ue)
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        lstm_out, _ = self.lstm(x)
        out = self.fc(lstm_out[:, -1, :])
        return out

# 实例化模型
vocab_size = len(word_to_num)  # 词汇表大小
embedding_dim = 100            # 嵌入维度
hidden_dim = 128               # LSTM 隐藏层维度
model = LSTMModel(vocab_size, embedding_dim, hidden_dim)

# 损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练轮数
num_epochs = 100
# 训练模型
train_features_tensor = torch.tensor(train_features, dtype=torch.lon
g)
train_labels_tensor = torch.tensor(train_labels, dtype=torch.long)

test_features_tensor = torch.tensor(test_features, dtype=torch.long)
test_labels_tensor = torch.tensor(test_labels, dtype=torch.long)

train_dataset = TensorDataset(train_features_tensor, train_labels_te
nsor)
test_dataset = TensorDataset(test_features_tensor, test_labels_tenso
r)
```

```
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuff
le=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle
=True)

# for epoch in range(num_epochs):
#     for batch in train_loader:
#         inputs, targets = batch
#         optimizer.zero_grad()
#         outputs = model(inputs)
#         loss = criterion(outputs, targets)
#         loss.backward()
#         optimizer.step()
#     if(epoch%10==0): print(f'Epoch {epoch}, Loss: {loss.item()}')
losses = []  # 用于存储每个周期的损失值

for epoch in range(num_epochs):
    total_loss = 0
    for batch in train_loader:
        inputs, targets = batch
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    average_loss = total_loss / len(train_loader)
    losses.append(average_loss)

    if(epoch % 10 == 0):
        print(f'Epoch {epoch}, Loss: {average_loss}')

plt.plot(losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()

model.eval()  # 将模型设置为评估模式
test_loss, correct = 0, 0

with torch.no_grad():  # 在评估期间不计算梯度
    for inputs, targets in test_loader:
```

```
        outputs = model(inputs)
        test_loss += criterion(outputs, targets).item()  # 累加损失
        correct += (outputs.argmax(1) == targets).type(torch.float).s
um().item()  # 计算正确预测数

test_loss /= len(test_loader.dataset)
print(f'Test Loss: {test_loss}')

# import random
# def generate_line(start_word, length=5):
#     model.eval()  # 设置为评估模式
#     text = start_word
#     for _ in range(length - 1):  # 减1是因为已经有了起始字
#          input_sequence = [word_to_num.get(word, random.choice(list
(word_to_num.values()))) for word in text][-sequence_length:]
#         input_tensor = torch.tensor(input_sequence, dtype=torch.lon
g).unsqueeze(0)
#         with torch.no_grad():
#             output = model(input_tensor)
#         next_word = num_to_word[output.argmax(1).item()]
#         text += next_word
#     return text

# def generate_acrostic(start_words):
#     poem = []
#     for word in start_words:
#         word = find_starting_word(word, word_to_num)
#         line = generate_line(word)
#         poem.append(line)
#     return '\n'.join(poem)

# def find_starting_word(char, word_to_num):
#     # 在词汇表中查找以特定字符开头的词语
#     starting_words = [word for word in word_to_num if word.startswi
th(char)]
#     if starting_words:
#         return random.choice(starting_words)  # 如果找到，随机选择一个

#     else:
#         return char
# 生成藏头诗
import random

def generate_line(model, start_word, sequence_length, word_to_num, nu
```

```
m_to_word, length=3):
    model.eval()  # 设置为评估模式
    text = start_word
    for _ in range(length - 1):  # 减1是因为已经有了起始字
        input_sequence = [word_to_num.get(word, random.choice(list(wo
rd_to_num.values()))) for word in text][-sequence_length:]
        input_tensor = torch.tensor(input_sequence, dtype=torch.long).
unsqueeze(0)
        with torch.no_grad():
            output = model(input_tensor)
        next_word = num_to_word[output.argmax(1).item()]
        text += next_word
    return text

def generate_acrostic(model, sequence_length, start_words, word_to_nu
m, num_to_word):
    poem = []
    for word in start_words:
        word = find_starting_word(word, word_to_num)
        line = generate_line(model, word, sequence_length, word_to_nu
m, num_to_word)
        poem.append(line)
    return '\n'.join(poem)

def find_starting_word(char, word_to_num):
    # 在词汇表中查找以特定字符开头的词语
    starting_words = [word for word in word_to_num if word.startswith
(char)]
    if starting_words:
        return random.choice(starting_words)  # 如果找到，随机选择一个
    else:
        return char

# print(generate_acrostic('新年快乐'))


# 使用示例
print(generate_acrostic(model, sequence_length, '新年快乐', word_to_n
um, num_to_word))

# 保存模型
torch.save(model.state_dict(), 'peo2/lstm_model.pth')

import pickle
```

```
with open('peo2/word_to_num.pkl', 'wb') as f:
    pickle.dump(word_to_num, f)
with open('peo2/num_to_word.pkl', 'wb') as f:
    pickle.dump(num_to_word, f)
```

app1.py

```
from flask import Flask, request, render_template
import numpy as np
import pickle

def load_model():
    Wxh = np.load('Wxh.npy')
    Whh = np.load('Whh.npy')
    Why = np.load('Why.npy')
    bh = np.load('bh.npy')
    by = np.load('by.npy')

    with open('char_to_index.pickle', 'rb') as handle:
        char_to_index = pickle.load(handle)

    with open('index_to_char.pickle', 'rb') as handle:
        index_to_char = pickle.load(handle)

    return Wxh, Whh, Why, bh, by, char_to_index, index_to_char

def generate_line(Wxh, Whh, Why, bh, by, char_to_index, index_to_cha
r, start_char, length=5):
    input_size, hidden_size = Wxh.shape[1], Wxh.shape[0]

    if start_char not in char_to_index:
        random_char = np.random.choice(list(char_to_index.keys()))
        start_index = char_to_index[random_char]
    else:
        start_index = char_to_index[start_char]

    x = np.zeros((input_size, 1))
    x[start_index] = 1
    h = np.zeros((hidden_size, 1))
    indices = [start_index]

    for _ in range(length - 1):
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
        y_hat = np.dot(Why, h) + by
```

```
        prob = np.exp(y_hat) / np.sum(np.exp(y_hat))
        next_char_index = np.random.choice(range(len(char_to_index)),
 p=prob.ravel())
        indices.append(next_char_index)
        x = np.zeros((input_size, 1))
        x[next_char_index] = 1

    generated_poem = "".join(index_to_char[i] for i in indices)
    return generated_poem
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    poem = ""
    if request.method == 'POST':
        input_chars = request.form['input_chars'].split(' ')
        Wxh, Whh, Why, bh, by, char_to_index, index_to_char = load_mod
el()
        for char in input_chars:
            generated_line = generate_line(Wxh, Whh, Why, bh, by, char
_to_index, index_to_char, char, length=5)
            poem += generated_line + "\n"
    return render_template('index.html', poem=poem)

if __name__ == '__main__':
    app.run(debug=True)
```

**index1.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>藏头诗生成器</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/s
tyle.css') }}">
</head>
<body>
    <h1>基于手撸 RNN 神经网络的藏头诗生成器</h1>
    <form method="post">
        输入字：<input type="text" name="input_chars">
        <input type="submit" value="生成诗句">
    </form>
    <pre>{{ poem }}</pre>
    <script src="{{ url_for('static', filename='js/script.js') }}"></
script>
```

```
    <div class="project-details">
        <h2>数据结构课设：神经网络</h2>
        <div class="team">
            <ul>
                <li>组长：杨熙承</li>
                <li>组员：王语桐</li>
                <li>组员：徐姜旸</li>
                <li>组员：林越</li>
            </ul>
        </div>
    </div>

</body>
</html>
```

app2.py

```
 import torch
import pickle
import random
from flask import Flask, request, render_template
# 定义 LSTM 模型类
class LSTMModel(torch.nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(LSTMModel, self).__init__()
        self.embedding = torch.nn.Embedding(vocab_size, embedding_dim)
        self.lstm = torch.nn.LSTM(embedding_dim, hidden_dim, batch_fi
rst=True)
        self.fc = torch.nn.Linear(hidden_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        lstm_out, _ = self.lstm(x)
        out = self.fc(lstm_out[:, -1, :])
        return out

# 载入模型和字典
def load_model_and_dictionaries(model_path, word_to_num_path, num_to
_word_path):
    with open(word_to_num_path, 'rb') as f:
        word_to_num = pickle.load(f)
    with open(num_to_word_path, 'rb') as f:
        num_to_word = pickle.load(f)

    model = LSTMModel(len(word_to_num), 100, 128)
    model.load_state_dict(torch.load(model_path))
```

```
    model.eval()

    return model, word_to_num, num_to_word

# 生成藏头诗的函数
def generate_acrostic(model, sequence_length, start_words, word_to_nu
m, num_to_word):
    poem = []
    for word in start_words:
        word = find_starting_word(word, word_to_num)
        line = generate_line(model, word, sequence_length, word_to_nu
m, num_to_word)
        poem.append(line)
    return '\n'.join(poem)

def generate_line(model, start_word, sequence_length, word_to_num, nu
m_to_word, length=3):
    model.eval()
    text = start_word
    for _ in range(length - 1):
        input_sequence = [word_to_num.get(word, random.choice(list(wo
rd_to_num.values()))) for word in text][-sequence_length:]
        input_tensor = torch.tensor(input_sequence, dtype=torch.long).
unsqueeze(0)
        with torch.no_grad():
            output = model(input_tensor)
        next_word = num_to_word[output.argmax(1).item()]
        text += next_word
    return text

def find_starting_word(char, word_to_num):
    starting_words = [word for word in word_to_num if word.startswith
(char)]
    if starting_words:
        return random.choice(starting_words)
    else:
        return char

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    poem = ""
    if request.method == "POST":
```

```
        start_words = request.form.get("start_words")
        if start_words:
            # 加载模型和字典
            model, word_to_num, num_to_word = load_model_and_dictionar
ies('peo2/lstm_model.pth', 'peo2/word_to_num.pkl', 'peo2/num_to_word.
pkl')
            # 生成藏头诗
            poem = generate_acrostic(model, 20, start_words, word_to_n
um, num_to_word)
    return render_template("index.html", poem=poem)

if __name__ == "__main__":
    app.run(debug=True,port=8080)
```

**index2.html**

```
<!DOCTYPE html>
<html>
<head>
    <title>基于LSTM神经网络的藏头诗生成器</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/s
tyle.css') }}">

</head>
<body>
    <h1>基于LSTM神经网络的藏头诗生成器</h1>
    <form method="post">
        开始词：<input type="text" name="start_words" required />
        <input type="submit" value="生成" />
    </form>
    {% if poem %}
        <pre>{{ poem }}</pre>
    {% endif %}
    <div class="team-info">
        <h2>数据结构课设：神经网络</h2>
        <ul>
            <li>组长：杨熙承</li>
            <li>组员：王语桐</li>
            <li>组员：徐姜旸</li>
            <li>组员：林越</li>
        </ul>
    </div>
    <script src="{{ url_for('static', filename='js/script.js') }}"></
script>
</body>
</html>
```