# 《数据结构》课程设计报告

## 1. 目的

### 1.1 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。

### 1.2 实现

你的数据结构课程设计大作业涉及到使用Python和神经网络来处理和生成文本，具体是通过分析中国古诗文本来进行学习和生成新的诗句。你的代码实现了一个简单的递归神经网络（**RNN**），用于处理和生成文本数据。

## 2.需求分析：

### 2.1 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。

### 2.2 选择必要性：

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。

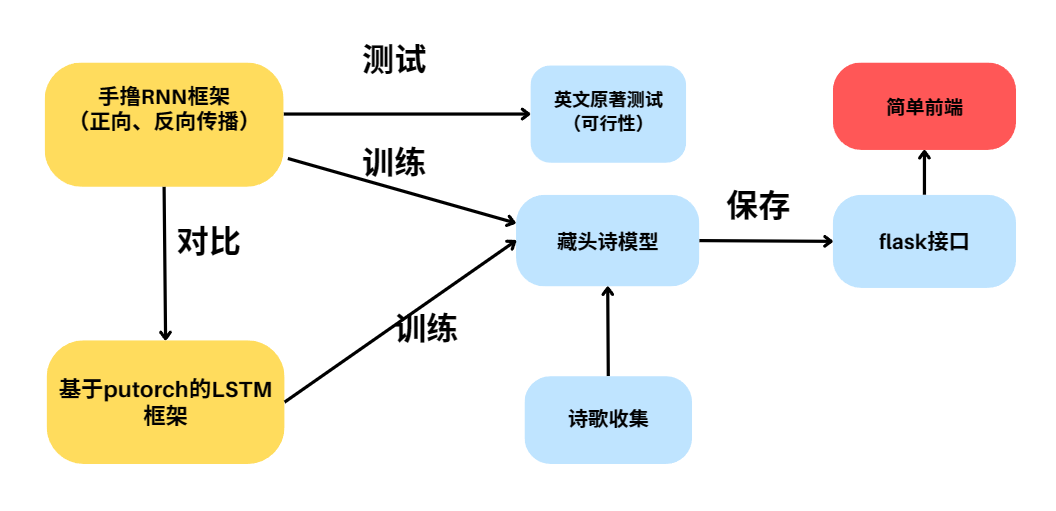
### 2.3 步骤分析：

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. 测试结果导出：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。

## 3.概要分析：

### 3.1 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。



### 3.2 模块简介

1. **手撸RNN框架（正向、反向传播）**：  
   这是流程图的起始点，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。

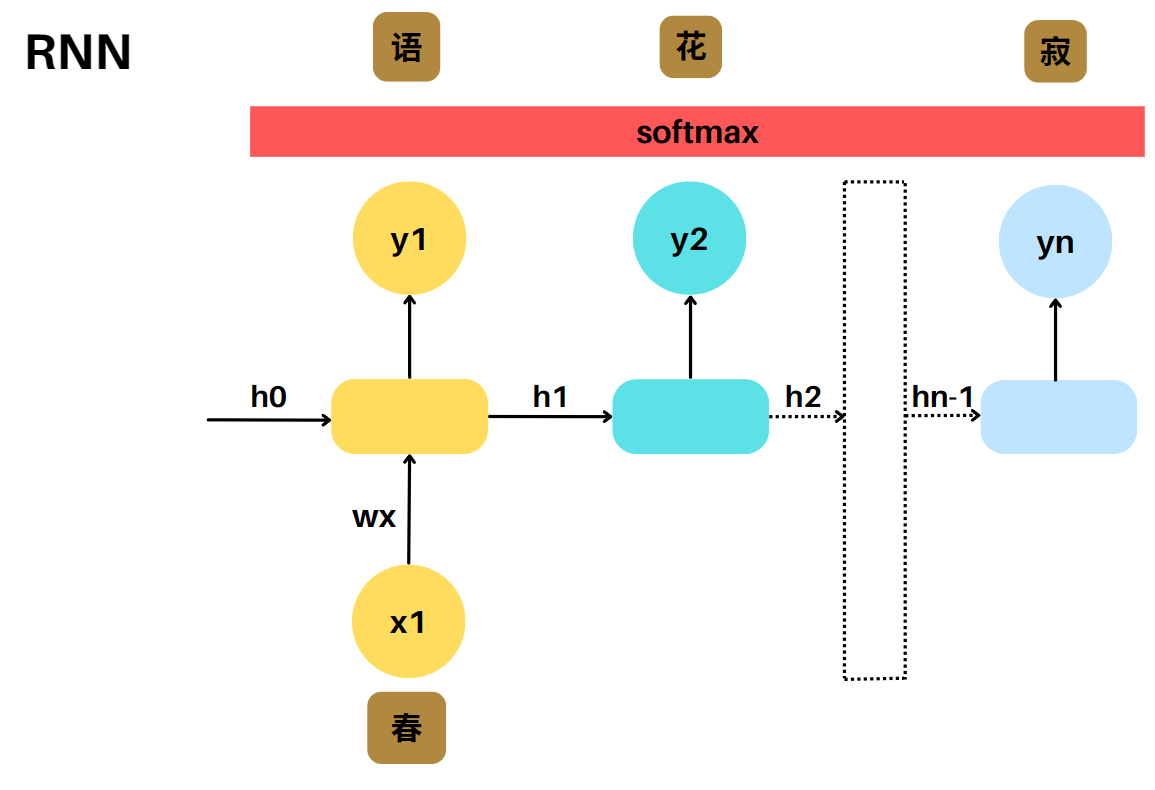
## 4. 详细设计

### 4.1 数据结构与算法设计

#### 4.1.1 手撸RNN模型

##### 外部模型

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



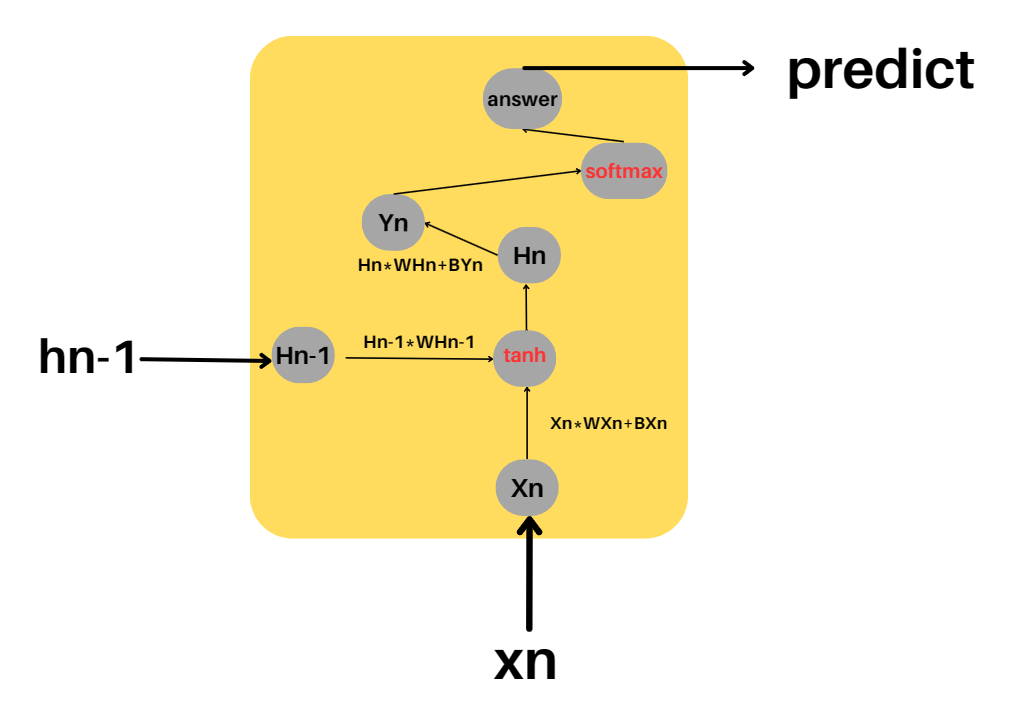
这张图以下是它的主要组成部分：

1. **输入层（x）**：该层接收序列中的第一个输入。
2. **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。
3. **权重（w）**：这表示输入和隐藏状态之间的权重。
4. **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。

##### 内部模型

该**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：



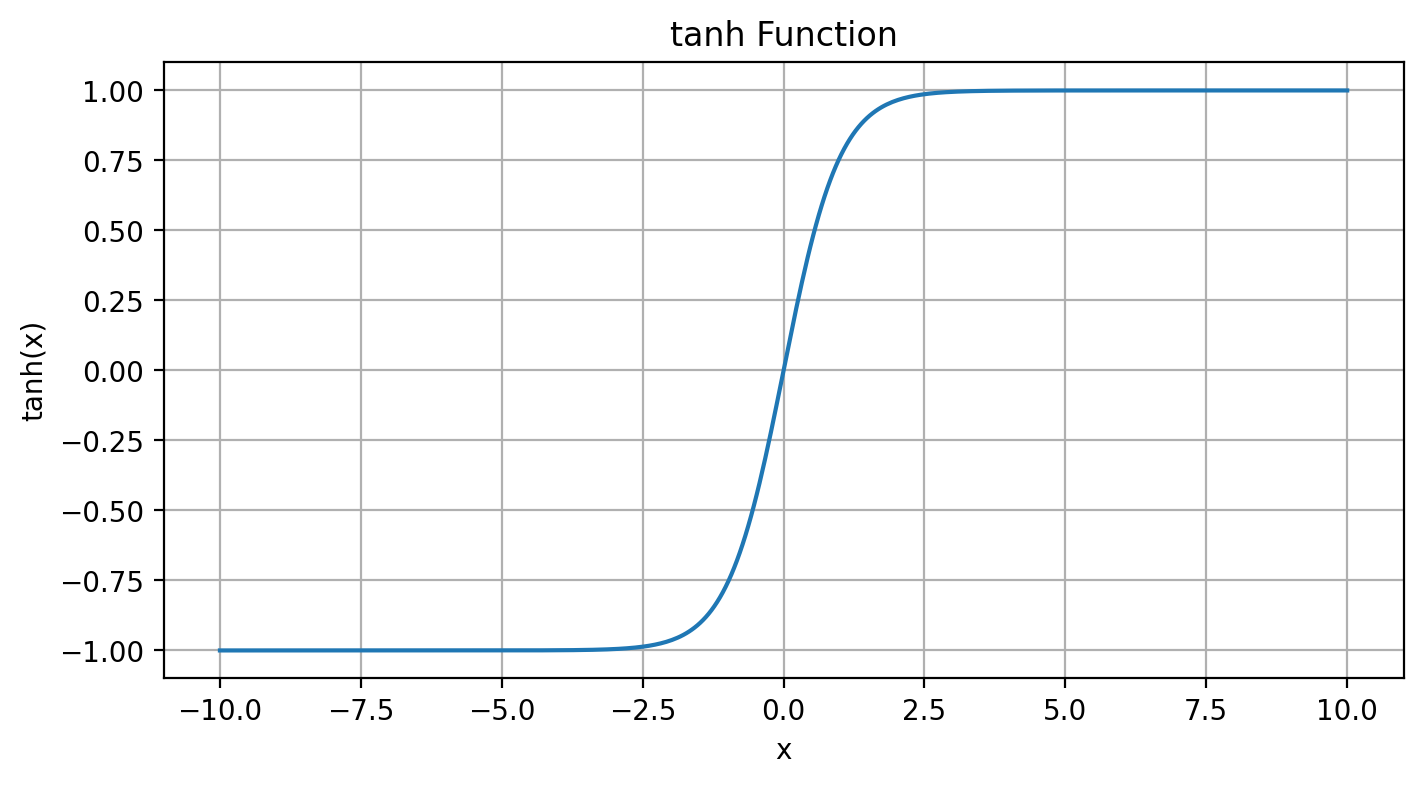
###### 正向传播步骤

内部模型的实现采用正向传播算法，具体如下：

1. 设置输入

* xs[t] = np.zeros((input\_size, 1)) # 初始化一个全零的输入向量  
  xs[t][inputs[t]] = 1 # 将输入向量的对应索引设为1
* 数学表示为 其中 是在时间步 的输入索引。

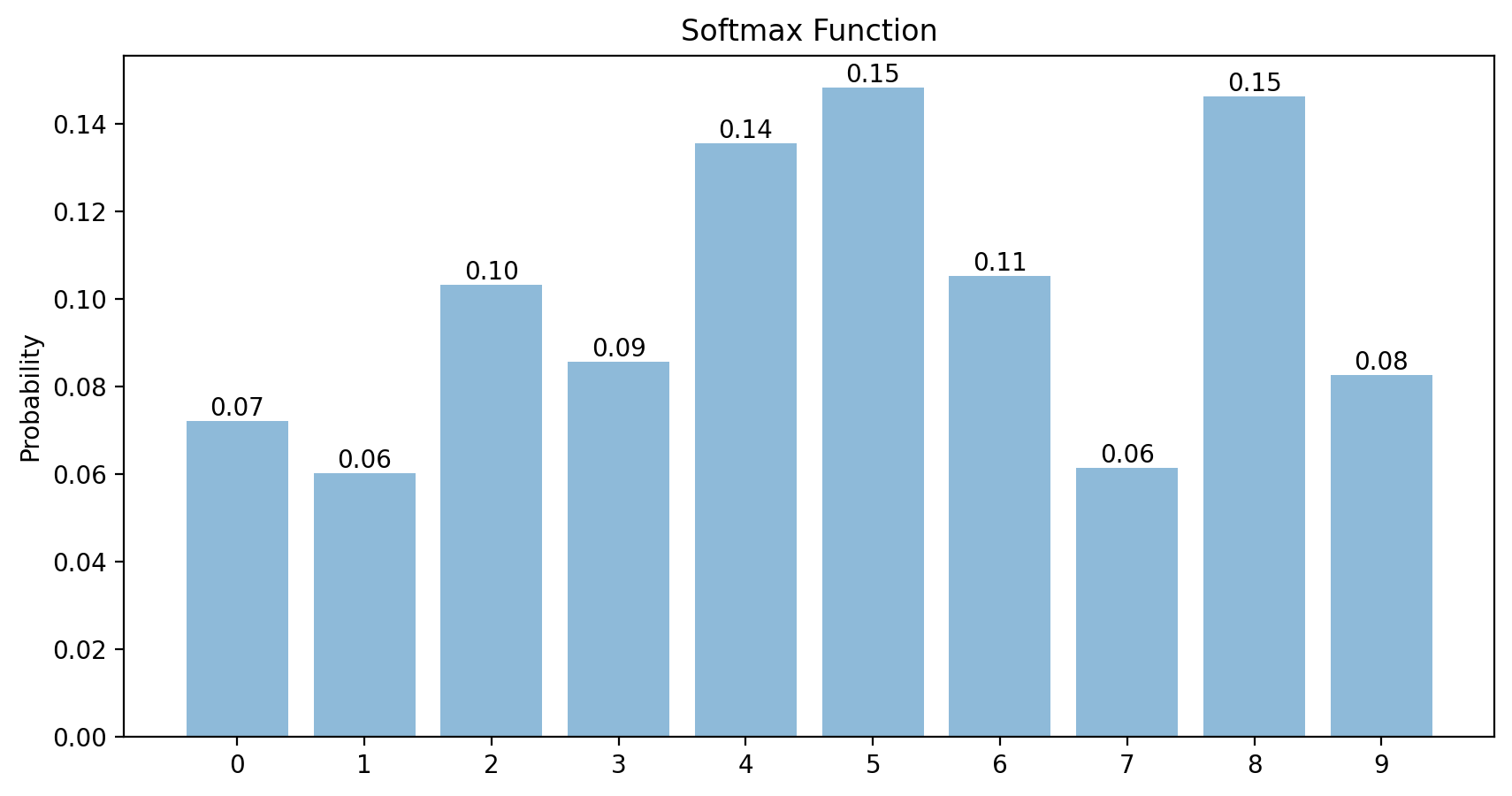
1. 计算时间步 的隐藏状态

* hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh)
* 对应的数学公式为:
* tanh是常见的处理0-1状态的激活函数，图像如下：
* 其中， 是输入到隐藏状态的权重矩阵， 是前一个隐藏状态到当前隐藏状态的权重矩阵， 是隐藏状态的偏置项。

1. 计算时间步 的输出 （在应用softmax之前的隐藏状态h）：

* ys[t] = np.dot(Why, hs[t]) + by
* 对应的数学公式为:  
  其中 是隐藏状态到输出的权重矩阵， 是输出的偏置项。

1. 应用softmax函数得到每个时间步的预测概率分布 ：

* ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
* 对应的数学公式为:  
    
  这个公式将原始输出 转换成一个概率分布，所有可能输出的概率和为1。

1. 累计计算并记录交叉熵损失：

* loss += -np.log(ps[t][targets[t], 0])
* 对应的数学公式为:  
  其中 是模型在时间步 预测正确目标的概率，这里 是目标输出的索引。

###### 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：

1. 计算输出层梯度 ：

* dy = np.copy(ps[t])  
  dy[targets[t]] -= 1
* 对应的数学公式为：  
  其中 是前向传播中计算的预测概率， 是正确类别的指示函数。

1. 计算与 和 相关的梯度：

* dWhy += np.dot(dy, hs[t].T)  
  dby += dy
* 对应的数学公式为：

1. 计算下一个时间步的隐藏层梯度 ：

* dh = np.dot(Why.T, dy) + dhnext
* 公式为：

1. 通过应用导数链式法则计算激活函数的原始梯度 ：

* dhraw = (1 - hs[t] \* hs[t]) \* dh
* 公式为：  
  这一步是应用了 函数的导数 。

1. 计算与 ，，和 相关的梯度：

* dbh += dhraw  
  dWxh += np.dot(dhraw, xs[t].T)  
  dWhh += np.dot(dhraw, hs[t-1].T)
* 对应的数学公式为：

1. 更新下一个时间步的 ：

* dhnext = np.dot(Whh.T, dhraw)
* 对应的数学公式为：

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。

1. 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：

* 其中：
* 代表模型参数（Wxh, Whh, Why, bh, by）。
* 代表学习率（learning\_rate）。
* 代表损失函数J关于参数的梯度（dWxh, dWhh, dWhy, dbh, dby）。
* 在代码中的 zip([Wxh, Whh, Why, bh, by], [dWxh, dWhh, dWhy, dbh, dby]) 用于同时遍历参数和对应的梯度。
* 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：
* 或者展开为具体的参数：
* 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。

#### 4.1.2 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：

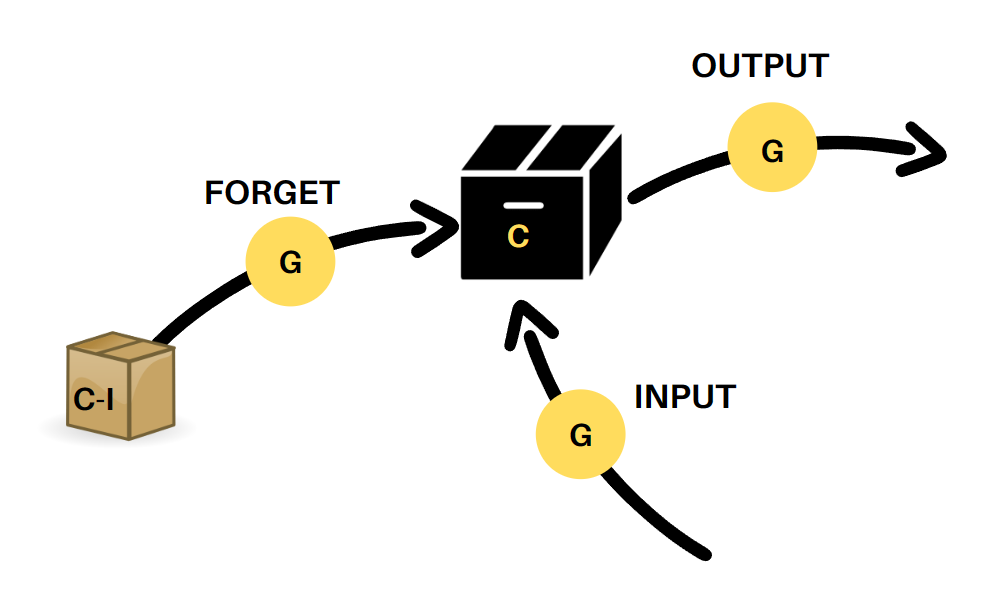


本课题之所以使用基于pytorch的LSTM模型与之对比，是因为简单RNN处理诗歌时会有一定的局限性。

为了解决梯度消失问题，提出了长短期记忆网络（LSTMs）和门控循环单元（GRUs）。这些结构引入了门控机制，使得网络能够更好地学习长期依赖。

#### 4.1.3 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：



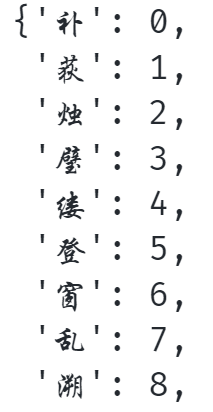
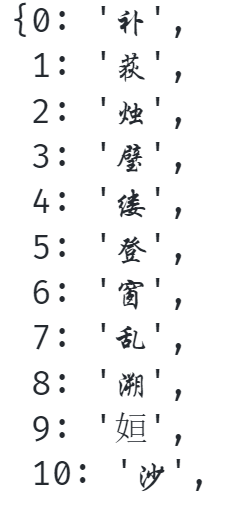
### 4.2 数据处理设计

##### 4.2.1 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**:

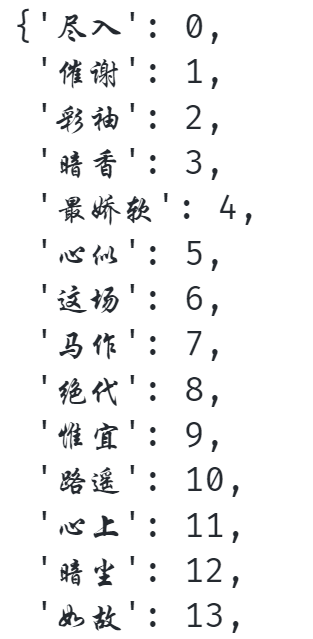
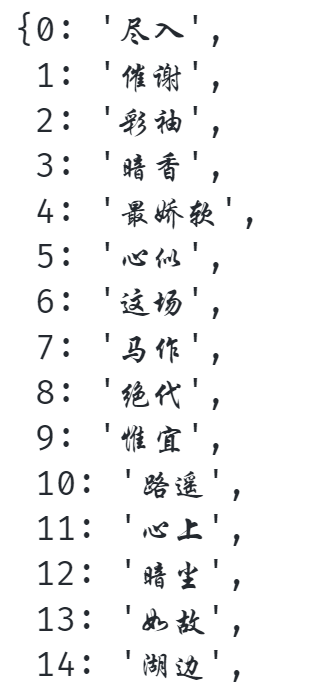
* [re.sub(r'[^\w\s]', '', poem)]
* [re.sub(r'[\s\n]', '', poem)]
* 使用正则表达式移除所有非单词字符和空白字符以外的字符，移除所有空白字符，包括空格和换行符。

1. **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。
2. **创建字符索引映射**:

* 加密映射：解密映射：

##### 4.2.2 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：

### 4.3 工程文件设计

##### 4.3.1 保存模型

1. **手撸RNN模型的保存**

* **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。
* **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 字典保存到一个名为 char\_to\_index.pickle
* 将 index\_to\_char 字典保存到一个名index\_to\_char.pickle

1. **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。
* 核心框架：
  1. 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。
  2. 实例化模型，并设置词汇表大小、嵌入维度和LSTM隐藏层维度。
  3. 定义交叉熵损失函数和Adam优化器，用于模型训练。
  4. 设置训练的周期数和将数据转换为PyTorch张量。
  5. 创建数据集和数据加载器，用于批量处理训练和测试数据。

##### 4.3.2 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。

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**：接下来，函数进入一个循环，循环的次数为 length - 1，因为起始字符已经确定。在每次循环中，它执行以下步骤：
   * **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。
   * **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。

##### 4.3.3 前后端设计

**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127.0.0.5000

**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。前端的设计界面如下：

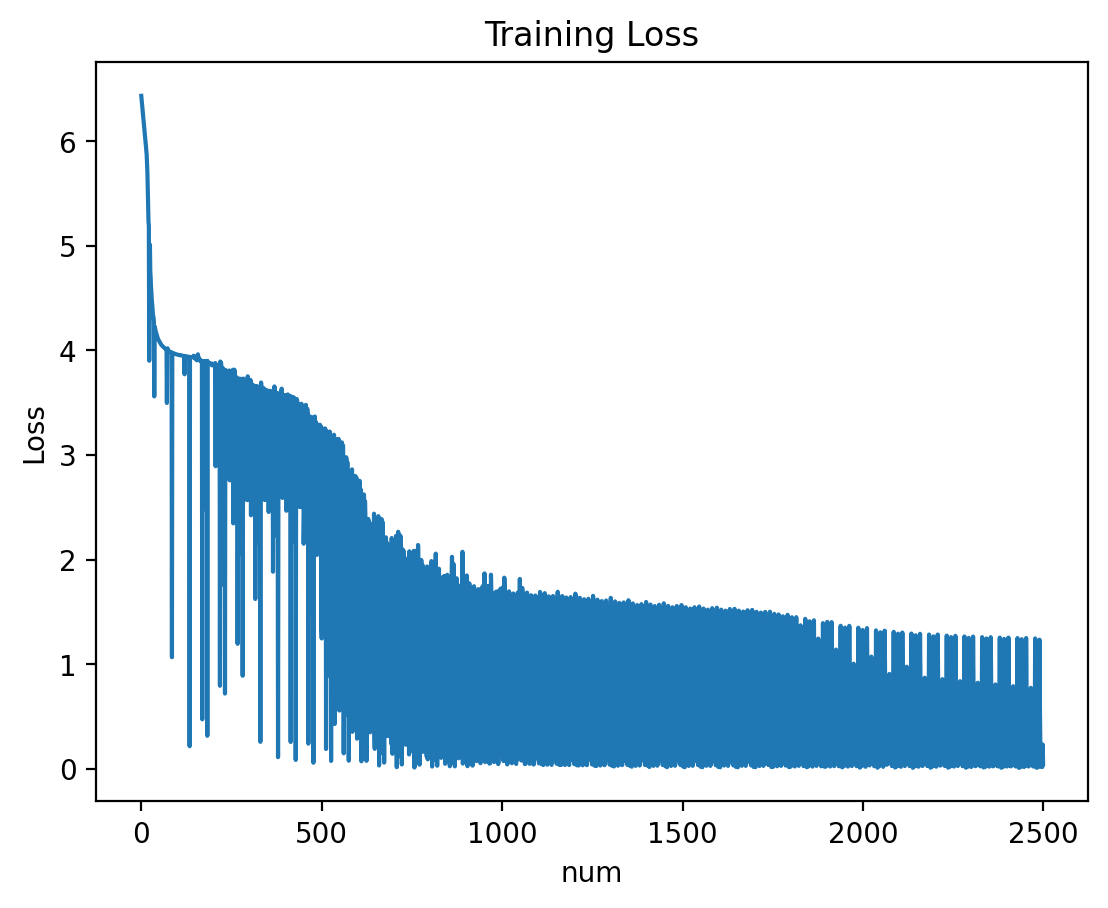


## 5.调试分析

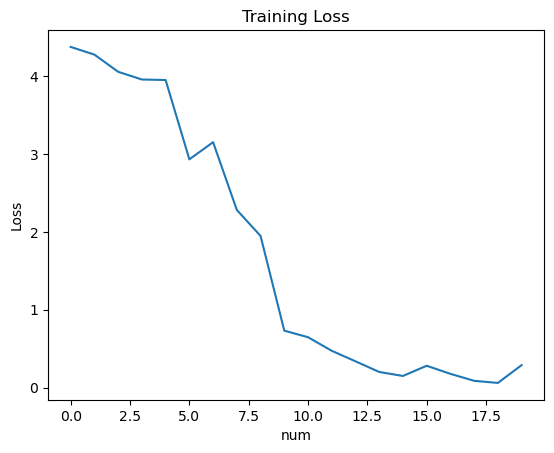
因为实现的是基于RNN的文本创作，对准确率不做要求，因为需要考虑模型的损失率，本模型使用的是交叉熵函数计算损失度。

#### RNN的分词与分字损失度对比

**分字:**

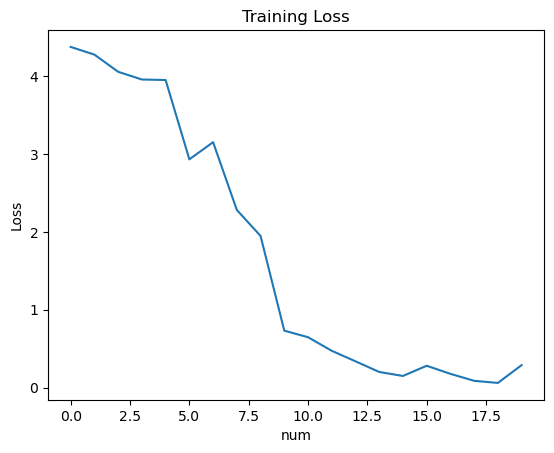


**分词：**

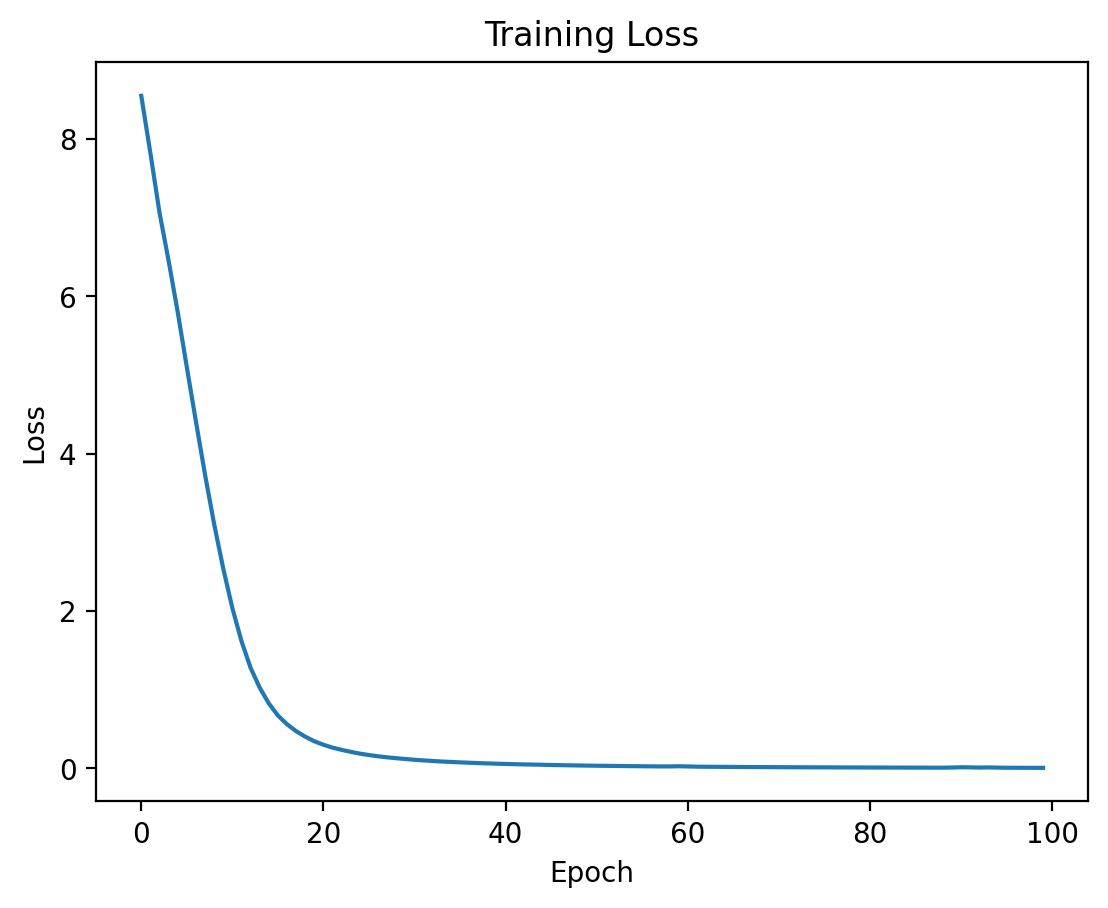


#### RNN与LSTM损失度对比

**RNN**

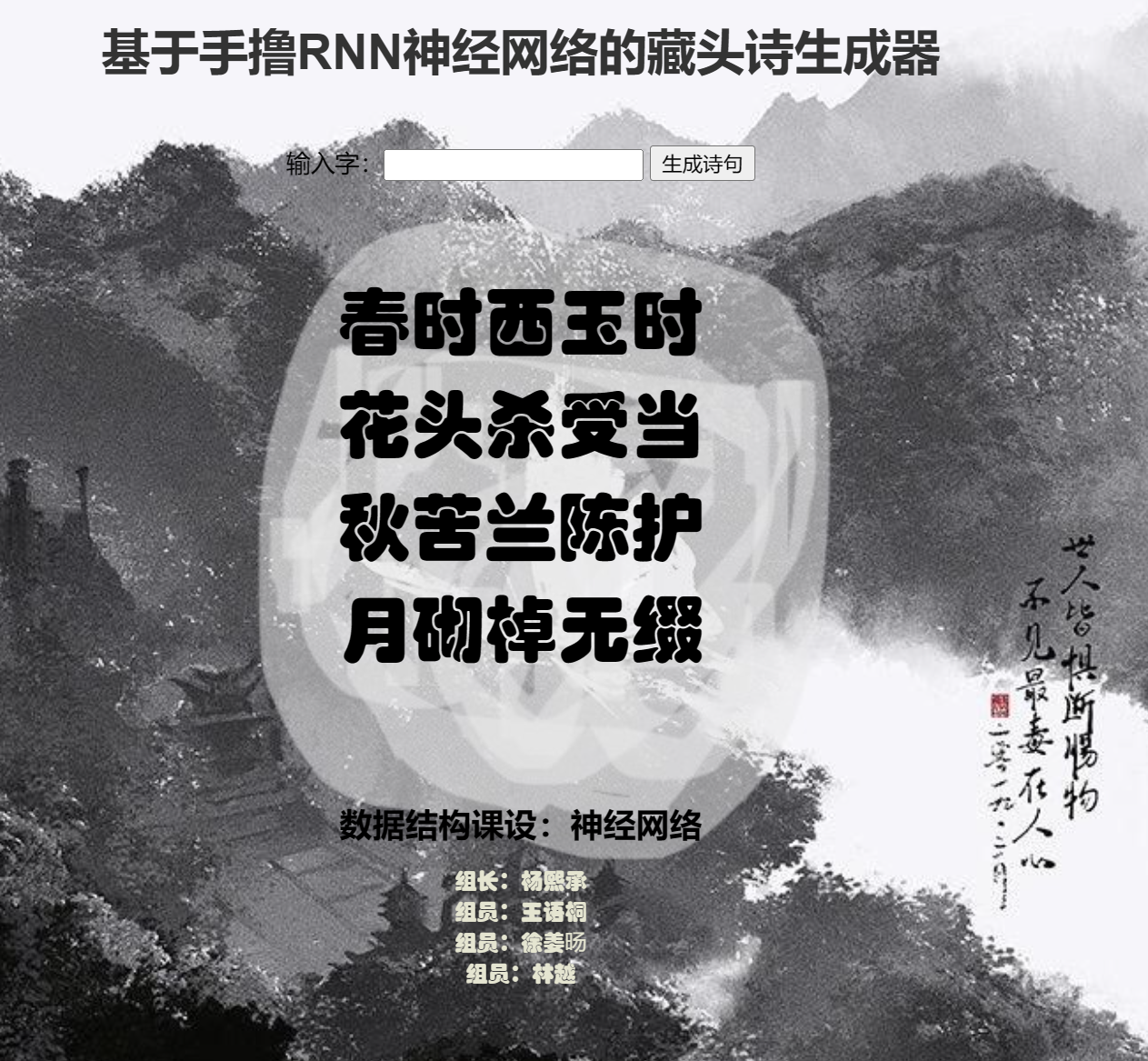


**LSTM**

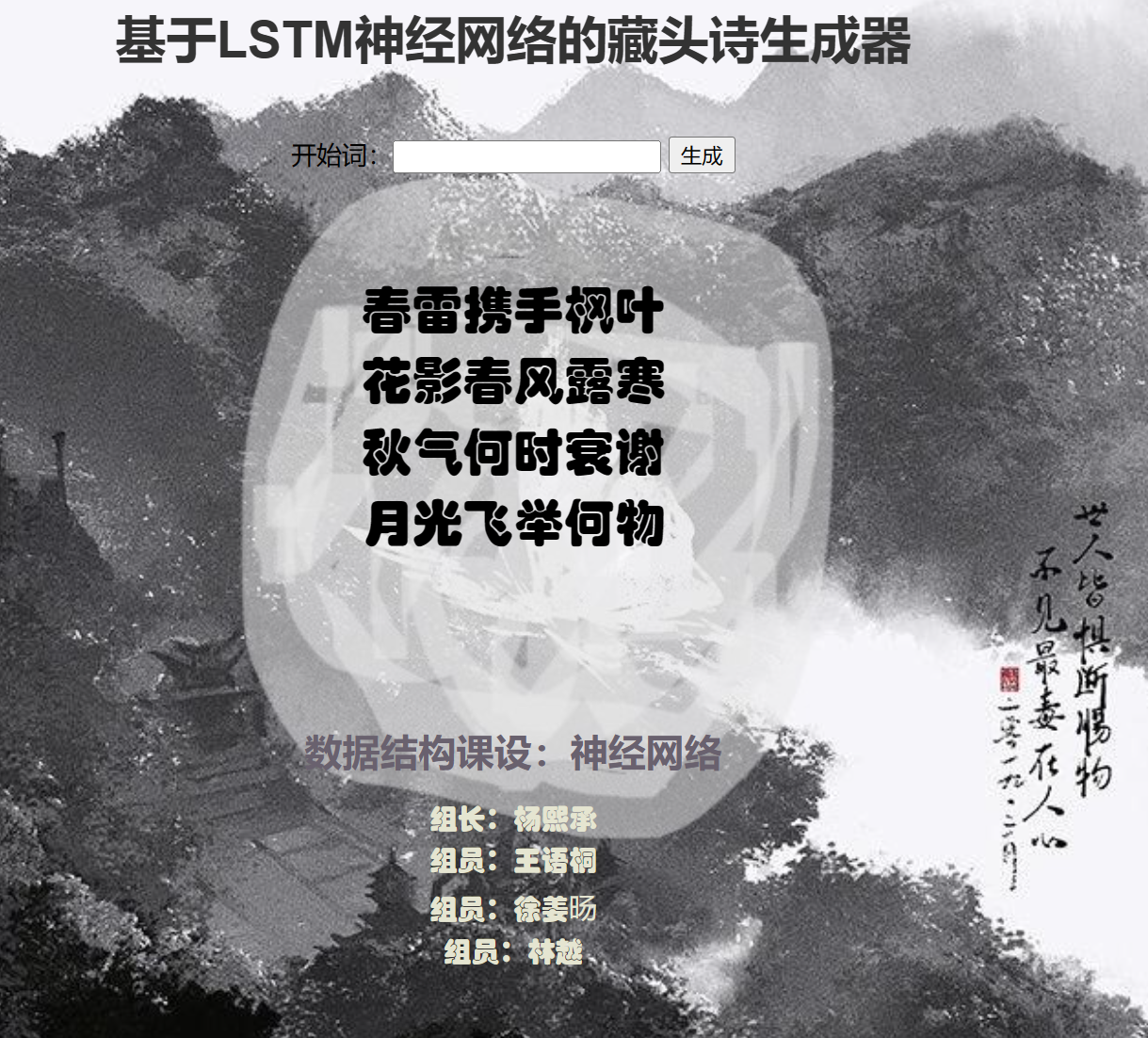


## 6. 测试结果

#### RNN



#### LSTM



## 附录

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

### demo1.py

（RNN）

import re   
import numpy as np   
import csv   
import matplotlib.pyplot as plt   
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   
poems = [re.sub(r'[\s\n]', '', poem) for poem in poems]   
poems   
# 去重   
vocab = set("".join(poems))   
   
# 长度就是模型接受的大小   
vocab\_size = len(vocab)   
vocab\_size   
# 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)}   
char\_to\_index   
index\_to\_char   
# 模型参数初始化   
input\_size = vocab\_size   
hidden\_size = 100 # 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]) + bh)   
 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.zeros\_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 = 2000   
patience = 800   
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(inputs, 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   
   
 lossess.append(loss)   
 if n % 100 == 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 n {n}, Lowest loss: {lowest\_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   
%config InlineBackend.figure\_format = 'retina'   
%matplotlib inline   
data = ''.join(poems)   
best\_params, lowest\_loss = train(data, num, patience)   
lowest\_loss   
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\_PROTOCOL)   
   
with open('index\_to\_char.pickle', 'wb') as handle:   
 pickle.dump(index\_to\_char, handle, protocol=pickle.HIGHEST\_PROTOCOL)   
import numpy as np   
import pickle   
   
np.save('./peo1/Wxh.npy', Wxh)   
np.save('./peo1/Whh.npy', Whh)   
np.save('./peo1/Why.npy', Why)   
np.save('./peo1/bh.npy', bh)   
np.save('./peo1/by.npy', by)   
   
with open('char\_to\_index.pickle', 'wb') as handle:   
 pickle.dump(char\_to\_index, handle, protocol=pickle.HIGHEST\_PROTOCOL)   
   
with open('index\_to\_char.pickle', 'wb') as handle:   
 pickle.dump(index\_to\_char, handle, protocol=pickle.HIGHEST\_PROTOCOL)

### demo2.py

(LSTM)

import matplotlib.pyplot as plt  
%config InlineBackend.figure\_format = 'retina'  
%matplotlib inline  
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()  
len(word\_to\_num)  
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)  
print(labels)  
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=True)  
 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.long)  
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\_tensor)  
test\_dataset = TensorDataset(test\_features\_tensor, test\_labels\_tensor)  
  
train\_loader = DataLoader(dataset=train\_dataset, batch\_size=32, shuffle=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).sum().item() # 计算正确预测数  
  
test\_loss /= len(test\_loader.dataset)  
print(f'Test Loss: {test\_loss}')  
word\_to\_num  
import random  
def generate\_quatrain(model, start\_sequence, sequence\_length, num\_lines=4, words\_per\_line=3):  
 model.eval() # 设置为评估模式  
 words = start\_sequence.split()  
 poem = []  
  
 for \_ in range(num\_lines):  
 line = []  
 for \_ in range(words\_per\_line):  
 input\_sequence = [word\_to\_num.get(word, random.choice(list(word\_to\_num.values()))) for word in words[-sequence\_length:]]  
 input\_sequence = input\_sequence[-sequence\_length:] # 确保序列长度正确  
 input\_tensor = torch.LongTensor(input\_sequence).unsqueeze(0)  
 with torch.no\_grad():  
 output = model(input\_tensor)  
 next\_word = num\_to\_word[output.argmax(1).item()]  
 line.append(next\_word)  
 words.append(next\_word)  
 poem.append(''.join(line))  
  
 return '\n'.join(poem)  
  
# 使用模型生成四言绝句  
start\_sequence = "江雪"  
generated\_poem = generate\_quatrain(model, start\_sequence, sequence\_length)  
print(generated\_poem)  
# 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.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(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.startswith(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, num\_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(word\_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\_num, 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\_num, 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\_num, 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)  
 import torch  
import torch.nn as nn  
import torch.optim as optim  
from torch.utils.data import TensorDataset, DataLoader  
# 假设 LSTMModel 是你的模型类  
# 首先重新创建模型架构  
  
model1 = LSTMModel(vocab\_size, embedding\_dim, hidden\_dim) # 使用与原模型相同的参数  
  
# 然后加载保存的模型状态  
model1.load\_state\_dict(torch.load('lstm\_poem\_model.pth'))  
  
# 确保将模型设置为评估模式  
model1.eval()  
# 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  
seq\_length = 25 # 序列长度  
learning\_rate = 1e-1  
# 使用模型生成四言绝句  
  
start\_sequence = input("输入想要的诗歌标题:")  
generated\_poem = generate\_quatrain(model1, start\_sequence, sequence\_length)  
print(start\_sequence)  
print(generated\_poem)  
start\_words = input()  
  
# 使用示例  
print(generate\_acrostic(model1, sequence\_length, start\_words, word\_to\_num, num\_to\_word))

### app1.py

(RNN)

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\_char, 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\_model()  
 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

(RNN)

<!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/style.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

(LSTM)

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\_first=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\_num, 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\_num, num\_to\_word)  
 poem.append(line)  
 return '\n'.join(poem)  
  
def generate\_line(model, start\_word, sequence\_length, word\_to\_num, num\_to\_word, length=3):  
 model.eval()   
 text = start\_word  
 for \_ in range(length - 1):   
 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.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\_dictionaries('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\_num, num\_to\_word)  
 return render\_template("index.html", poem=poem)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=True,port=8080)

### index2.html

(LSTM)

<!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/style.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>