中国科学院大学
University of Chinese Academy of Sciences

深度学习课程实验报告

实验三：自动写诗

姓名 肖一笑

学号 2024E8015082070

院所 中国科学院软件研究所

2025 年 5 月 18 日

目录

1 实验概述 **3**
1.1 实验目的 . . . 3
1.2 实验要求 . . . 3
1.3 数据集介绍 . . . 3

2 解决方案 **3**
2.1 功能模块介绍 . . . 4
2.1.1 基础模块 . . . 4
2.1.2 数据加载模块（poetryData） . . . 4
2.1.3 模型训练模块（train 函数） . . . 5
2.1.4 普通诗歌生成模块 . . . 6
2.1.5 藏头诗生成模块 . . . 8
2.1.6 主程序模块 . . . 9
2.2 模型各层功能详解 . . . 10
2.2.1 Embedding 层：词向量映射 . . . 10
2.2.2 LSTM 层：建模词序列上下文依赖 . . . 11
2.2.3 Linear 层：预测下一个词 . . . 12
2.3 模型前向传播流程 . . . 12
2.4 训练与生成中的使用方式 . . . 13

3 实验流程 **14**
3.1 搭建环境 . . . 14
3.2 编写实验程序 . . . 14

4 实验结果与分析 **14**
4.1 普通诗歌生成 . . . 14
4.2 藏头诗生成 . . . 16
4.3 失败实例 . . . 16

5 总结 **17**

1 实验概述

1.1 实验目的

- 理解和掌握循环神经网络概念及在深度学习框架中的实现。
- 掌握使用深度学习框架进行文本生成任务的基本流程: 如数据读取、构造网络、训练和预测等。

1.2 实验要求

- 基于 Python 语言和任意一种深度学习框架（实验指导书中使用 Pytorch 框架进行介绍），完成数据读取、网络设计、网络构建、模型训练和模型测试等过程，最终实现一个可以自动写诗的程序。网络结构设计要有自己的方案，不能与实验指导书完全相同。
- 随意给出首句，如给定“湖光秋月两相和”，输出模型续写的诗句。也可以根据自己的兴趣，进一步实现写藏头诗（不做要求）。要求输出的诗句尽可能地满足汉语语法和表达习惯。
- 按照规定时间在课程网站上提交实验报告，代码和 PPT。

1.3 数据集介绍

实验提供预处理过的数据集，含有 57580 首唐诗，每首诗限定在 125 词，不足 125 词的以 </s> 填充。数据集以 npz 文件形式保存，包含三个部分:

- data: 诗词数据，将诗词中的字转化为其在字典中的序号表示。
- ix2word: 序号到字的映射
- word2ix: 字到序号的映射

2 解决方案

实现自动写诗的核心在于构建一个能够学习中文诗歌语言规律的生成模型。本实验基于循环神经网络（RNN）中的 LSTM 结构，设计了“Embedding → 多层 LSTM → 全连接层”的神经网络架构，通过对大规模唐诗数据的训练，使模型能够根据输入的起始字或藏头词，逐字预测并生成符合汉语语法与诗歌韵律的句子，从而实现自动化的诗歌创作。该方案不仅具备良好的上下文建模能力，还能通过采样策略提升生成内容的多样性与自然度。

2.1 功能模块介绍

2.1.1 基础模块

- 功能说明：BasicModule 是对 torch.nn.Module 的封装，主要解决 PyTorch 原生模型保存/加载操作繁琐、不直观的问题。它为你的模型提供统一的 save() 和 load() 接口，便于复用和维护。

- 代码：

```
    class BasicModule(nn.Module):
        """
        封装nn.Module，提供模型加载和保存接口
        """
        def __init__(self):
            super(BasicModule, self).__init__()
            self.modelName = str(type(self))  # 模型名称

        def load(self, path):
            """加载指定路径的模型"""
            self.load_state_dict(torch.load(path))

        def save(self, name=None):
            """保存模型到指定路径"""
            if name is None:
                # 默认路径：models/模型名称_月日_时分.pth
                prepath = 'models/' + self.modelName + '_'
                name = time.strftime(prepath + '%m%d_%H_%M.pth')
            torch.save(self.state_dict(), name)
            print("模型保存路径：", name)
            return name
```

2.1.2 数据加载模块（poetryData）

- 功能说明：这个函数用于读取.npz 格式的诗歌数据文件，并将其打包为 PyTorch 的 DataLoader 以便训练时批量加载。它还返回词表相关的两个映射字典（ix2word 和 word2ix）。

- 输入输出说明：

  - 输入：压缩的 tang.npz 数据文件
  - 输出：DataLoader（训练用）、ix2word（索引 → 词）、word2ix（词 → 索引）

  - Linear：将 LSTM 输出映射为词表概率分布

- 代码：

```
    def poetryData(filename, batch_size):
        """
        加载诗歌数据
        - filename: 数据文件路径 (tang.npz)
        - batch_size: 批大小
        """
        datas = np.load(filename, allow_pickle=True)
        data = datas['data']  # 诗歌数据
        ix2word = datas['ix2word'].item()  # 索引到词的映射
        word2ix = datas['word2ix'].item()  # 词到索引的映射

        # 转换为Tensor并创建DataLoader
        data = torch.from_numpy(data)
        dataloader = DataLoader(data, batch_size=batch_size,
            shuffle=True, num_workers=0)
        return dataloader, ix2word, word2ix
```

2.1.3 模型训练模块（train 函数）

- 功能说明：这是模型训练核心逻辑的封装。它实现了完整的训练循环：数据加载、前向传播、计算损失、反向传播、优化更新、学习率调节、日志可视化等。

- 代码：

```
    def train(model, filename, batch_size, lr, epochs, device,
        trainwriter, pre_model_path=None):

        dataloader, ix2word, word2ix = poetryData(filename,
            batch_size)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=lr)
        scheduler = optim.lr_scheduler.ReduceLROnPlateau(
            optimizer, mode='min', patience=5)

        model.to(device)  # 模型转移到设备

        for epoch in range(epochs):
            model.train()
```

```
            total_loss = 0
            for i, data in enumerate(dataloader):
                # 数据转移到设备
                data = data.long().transpose(1, 0).contiguous().
                    to(device)
                input, target = data[:-1, :], data[1:, :]

                # 前向传播
                output, _ = model(input)
                loss = criterion(output, target.view(-1))

                # 反向传播
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                # 记录损失
                total_loss += loss.item()
                trainwriter.add_scalar('Train Loss', loss.item()
                    , epoch * len(dataloader) + i)

                if i % 100 == 0:
                    print(f'Epoch: {epoch+1}/{epochs}, Batch: {i
                        }, Loss: {loss.item():.4f}')

            # 学习率调整
            avg_loss = total_loss / len(dataloader)
            scheduler.step(avg_loss)

        trainwriter.close()
        model.save()
```

2.1.4 普通诗歌生成模块

- 功能说明：给定一个开头（start_words），自动续写完整诗句，直到生成结束符 <EOP> 或到达最大长度。可选“意境词”（prefix_words）用于生成前的语境引导。
- 代码：

```
    def generate(model, filename, device, start_words,
        max_gen_len, prefix_words=None):
        """
        生成诗歌（给定开头）
        - model: 诗歌模型
        - filename: 数据文件路径
        - device: 设备（CPU/GPU）
        - start_words: 诗歌开头
        - max_gen_len: 最大生成长度
        - prefix_words: 意境词（可选）
        """
        # 加载数据字典
        _, ix2word, word2ix = poetryData(filename, 1)
        model.to(device)
        results = list(start_words)

        # 初始输入为<START>
        input = torch.Tensor([word2ix['<START>']]).view(1, 1).
            long().to(device)
        hidden = None

        # 生成诗歌
        for i in range(max_gen_len):
            output, hidden = model(input, hidden)
            top_index = output.data[0].topk(1)[1][0].item()
            w = ix2word[top_index]

            # 前几个词使用给定的开头
            if i < len(start_words):
                w = results[i]
                input = input.data.new([word2ix[w]]).view(1, 1)
            else:
                results.append(w)
                input = input.data.new([top_index]).view(1, 1)

            # 遇到结束符停止
            if w == '<EOP>':
                del results[-1]
                break

```

```
        return results
```

2.1.5 藏头诗生成模块

- 功能说明：这是模型训练核心逻辑的封装。它实现了完整的训练循环：数据加载、前向传播、计算损失、反向传播、优化更新、学习率调节、日志可视化等。
- 代码：

```
    def generate_acrostic(model, filename, device,
        start_words_acrostic, max_gen_len_acrostic,
        prefix_words_acrostic=None):
        """
        生成藏头诗
        - model: 诗歌模型
        - filename: 数据文件路径
        - device: 设备 (CPU/GPU)
        - start_words_acrostic: 藏头词
        - max_gen_len_acrostic: 最大生成长度
        - prefix_words_acrostic: 意境词 (可选)
        """
        # 加载数据字典
        _, ix2word, word2ix = poetryData(filename, 1)
        model.to(device)
        results = []
        index = 0
        pre_word = '<START>'

        # 初始输入为<START>
        input = torch.Tensor([word2ix['<START>']]).view(1, 1).
            long().to(device)
        hidden = None

        # 生成藏头诗
        for i in range(max_gen_len_acrostic):
            output, hidden = model(input, hidden)
            top_index = output.data[0].topk(1)[1][0].item()
            w = ix2word[top_index]

            # 遇到句尾或开头时，填入藏头词
            if pre_word in {'。', '！', '<START>'}:
```

```
                if index == len(start_words_acrostic):
                    break
                else:
                    w = start_words_acrostic[index]
                    index += 1
                    input = input.data.new([word2ix[w]]).view(1,
                        1)
            else:
                input = input.data.new([word2ix[w]]).view(1, 1)

            results.append(w)
            pre_word = w

        return results
```

- 关键逻辑：
  - 每遇到标点。！<START> 时，从藏头词取出一个字作为下一句开头
  - 保持上下文状态连续
  - 支持“prefix_words_acrostic”情感引导

2.1.6 主程序模块

- 功能说明：这是程序的入口，用于选择运行模式：训练、普通诗生成、藏头诗生成。用户通过修改 mode 来切换功能。
- 代码：

```
    def generate(model, filename, device, start_words,
        max_gen_len, prefix_words=None):
        """
        生成诗歌（给定开头）
        - model: 诗歌模型
        - filename: 数据文件路径
        - device: 设备（CPU/GPU）
        - start_words: 诗歌开头
        - max_gen_len: 最大生成长度
        - prefix_words: 意境词（可选）
        """
        # 加载数据字典
        _, ix2word, word2ix = poetryData(filename, 1)
```

```
        model.to(device)
        results = list(start_words)

        # 初始输入为<START>
        input = torch.Tensor([word2ix['<START>']]).view(1, 1).
            long().to(device)
        hidden = None

        # 生成诗歌
        for i in range(max_gen_len):
            output, hidden = model(input, hidden)
            top_index = output.data[0].topk(1)[1][0].item()
            w = ix2word[top_index]

            # 前几个词使用给定的开头
            if i < len(start_words):
                w = results[i]
                input = input.data.new([word2ix[w]]).view(1, 1)
            else:
                results.append(w)
                input = input.data.new([top_index]).view(1, 1)

            # 遇到结束符停止
            if w == '<EOP>':
                del results[-1]
                break

        return results
```

- 交互设计：
  - 提示用户输入开头字或藏头字
  - 持续运行，直到输入 quit 退出
  - 自动格式化诗句（断句换行）

2.2 模型各层功能详解

2.2.1 Embedding 层：词向量映射

- 代码：

```
self.embeddings = nn.Embedding(vocab_size, embedding_dim)
```

- 作用说明：
  - 原理：将稀疏离散的词 ID 映射到一个低维语义空间中（如 128 维），让模型可以“理解”词与词之间的关系。
  - 输入：词（或字符）索引组成的序列，如 [125, 22, 98]。
  - 输出：索引对应的稠密向量表示，即所谓“词嵌入”。

  举例：词“春”索引为 105，映射为一个 128 维的向量 [0.2, -0.1, ..., 0.05]。

2.2.2 LSTM 层：建模词序列上下文依赖

- 代码：

```
self.lstm = nn.LSTM(embedding_dim, self.hidden_dim,
    num_layers=3)
```

- 结构说明：
  - 输入维度：词向量维度（如 128）
  - 输出维度：隐状态维度（如 256）
  - 层数：3 层堆叠的 LSTM（每层都输出到下一层）
  - 支持长距离依赖建模（比 RNN 更强）
- LSTM 原理简述：
  - LSTM（Long Short-Term Memory）是一种改进型的循环神经网络（RNN），能有效处理长期依赖问题。
  - 内部引入了门机制（输入门、遗忘门、输出门）控制信息的传递与遗忘。
- 输出说明：
  - 输出张量 output：每个时间步的隐状态序列
  - 隐状态 hidden：一个元组 (h_n, c_n)，用于生成时保持上下文

2.2.3 Linear 层：预测下一个词

- 代码：

```
    self.fc = nn.Sequential(
        nn.Linear(self.hidden_dim, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, vocab_size)
    )
```

- 作用说明：
  - 将 LSTM 的输出（256 维）映射到一个更高维空间（512）后，最终映射为 vocab_size 维。
  - vocab_size 是词表的大小（即词的种类数量），如 8293。
  - 最后通过 softmax（训练中隐含）计算出每个词的概率，表示“当前上下文下下一个词为 X 的可能性”。
- 举例：输入是“春眠不觉晓”，模型预测下一个词可能是“处”、“夜”、“闻”等，并给出概率分布：

```
    {
      '处': 0.35,
      '夜': 0.20,
      '闻': 0.18,
      '晓': 0.05,
      ...
    }
```

2.3 模型前向传播流程

- 函数定义：

```
    def forward(self, input, hidden=None):
    """
    前向传播
    - input: 输入序列 (seq_len, batch_size)
    - hidden: LSTM的隐藏状态 (h_0, c_0)
    """
    seq_len, batch_size = input.size()

    # 初始化隐藏状态
```

```
    if hidden is None:
        h_0 = input.data.new(3, batch_size, self.hidden_dim).
            fill_(0).float()
        c_0 = input.data.new(3, batch_size, self.hidden_dim).
            fill_(0).float()
    else:
        h_0, c_0 = hidden

    # 通过Embedding和LSTM
    embeds = self.embeddings(input)
    output, hidden = self.lstm(embeds, (h_0, c_0))

    # 通过全连接层
    output = self.fc(output.view(seq_len * batch_size, -1))
    return output, hidden
```

- 输入说明：
  - input: 输入序列张量，形状为 (seq_len, batch_size)
  - hidden: 可选的 LSTM 隐状态（生成时需要保持）
- 流程总结：
  - 输入词序列 → Embedding → 词向量序列（形状变为 (seq_len, batch, embed_dim)）
  - 输入到 LSTM 中，输出隐状态序列和最终隐藏状态（形状为 (seq_len, batch, hidden_dim)）
  - 隐状态 reshape 后输入到全连接网络，得到 (seq_len * batch, vocab_size) 的输出
  - 在训练中：使用 CrossEntropyLoss 与目标对比；在生成中：使用 softmax → 采样词

2.4 训练与生成中的使用方式

- 训练中
  - 批次数据喂入模型
  - 模型预测下一个词
  - 与真实词对比计算损失

  - 反向传播优化模型参数

- 生成中
  - 输入一个起始词 <START> 或用户给出的前缀词
  - 逐步生成下一个词 → 再作为输入 → 生成新的词
  - 直到遇到 <EOP> 或达到最大长度

3 实验流程

3.1 搭建环境

- 下载 IDE：VScode
- 安装 Python 以及 Anaconda
- 创建虚拟环境：**conda create -n pytorch python=3.10**

3.2 编写实验程序

autoPoetry.py 代码在与该文档同一文件夹下，训练运行结果如下：

```
(pytorch) PS E:\集中教学\DeepLearning\lab3> & C:/Users/Lenovo/.conda/envs/pytorch/python.exe e:/集中教学/DeepLearning/lab3/
Epoch: 1/20, Batch: 0, Loss: 9.0346
Epoch: 1/20, Batch: 100, Loss: 3.2478
Epoch: 1/20, Batch: 200, Loss: 2.9944
Epoch: 1/20, Batch: 300, Loss: 2.8102
Epoch: 1/20, Batch: 400, Loss: 2.4384
Epoch: 1/20, Batch: 500, Loss: 2.6702
Epoch: 1/20, Batch: 600, Loss: 2.4977
Epoch: 1/20, Batch: 700, Loss: 2.3347
Epoch: 1/20, Batch: 800, Loss: 2.1592
Epoch: 1/20, Batch: 900, Loss: 2.2470
Epoch: 1/20, Batch: 1000, Loss: 2.1753
Epoch: 1/20, Batch: 1100, Loss: 2.2659
Epoch: 1/20, Batch: 1200, Loss: 2.6671
Epoch: 1/20, Batch: 1300, Loss: 2.5811
Epoch: 1/20, Batch: 1400, Loss: 2.9480
Epoch: 1/20, Batch: 1500, Loss: 3.1370
Epoch: 1/20, Batch: 1600, Loss: 2.6323
Epoch: 1/20, Batch: 1700, Loss: 2.5865
Epoch: 1/20, Batch: 1800, Loss: 2.5425
```

图 1: 训练运行结果

4 实验结果与分析

4.1 普通诗歌生成

如图 2、图 3以及图 4所示。

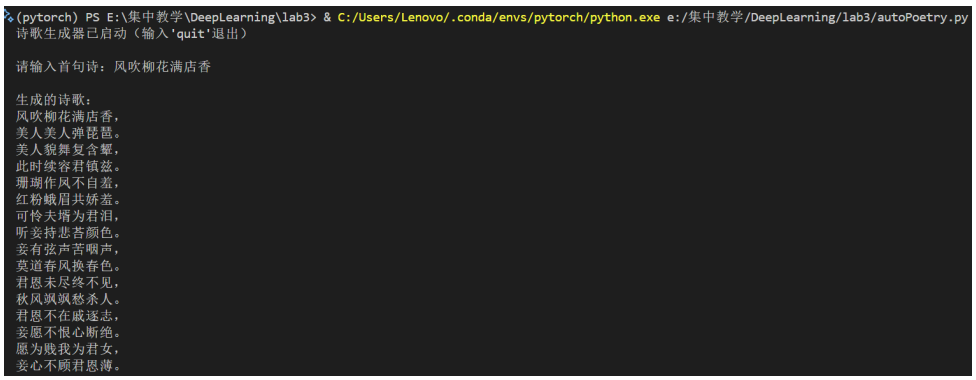

图 2: 普通写诗运行结果 1

图 3: 普通写诗运行结果 2

图 4: 普通写诗运行结果 3

图 5: 藏头诗运行结果 1

图 6: 藏头诗运行结果 2

图 7: 藏头诗运行结果 3

4.2 藏头诗生成

如图 5、图 6以及图 7所示。

4.3 失败实例

如图 8、图 9所示。

图 8: 普通写诗生成失败示例

图 9: 藏头诗生成失败示例

原因：由于藏头词中的字不在词表中，会触发 KeyError 并被 try-except 捕获。

5 总结

本实验的目标是基于 PyTorch 框架，实现一个能够自动生成唐诗的模型。通过本次实验，我深入掌握了循环神经网络（RNN）尤其是 LSTM 的结构及其在文本生成中的实际应用流程。以下是对本次实验的主要内容与收获的总结：

- 实验采用的是传统的序列生成模型架构：Embedding → 多层 LSTM → Linear，这是一种经典且有效的语言建模方案：
  - Embedding 层：将输入的词（或字符）索引映射为稠密向量，使得模型能够在语义空间中处理离散文本信息。
  - LSTM（Long Short-Term Memory）多层网络：使用三层堆叠的 LSTM 模块来增强模型的记忆与表达能力，能够捕捉长距离的上下文依赖，提升诗句的语言连贯性与语义一致性。
  - 全连接输出层（Linear）：将 LSTM 的输出映射为与词表大小相同的向量，表示下一个词的概率分布，供采样使用。
- 在诗句生成阶段，我们对模型输出的采样策略进行了改进：
  - 原始方法为贪婪解码（即取最大概率词）；
  - 本实验改为随机采样（依据 softmax 概率分布），使得生成结果更加自然，句式更加多样，避免了内容重复和风格单一的问题。

  同时，模型支持用户输入前缀词（prefix_words）作为意境引导，进一步增强了生成诗歌的主题控制能力。
- 为了满足实验扩展需求，本项目还实现了藏头诗生成功能。用户可输入任意藏头词，模型会自动控制每句开头字，从而生成格式规范、内容自然的藏头诗。其关键实现逻辑为：
  - 在生成过程中遇到句末标点时，从藏头词中取出一个字作为下一句的开头；
  - 其余词由模型自由生成，保持整体诗意。
- 通过实现训练模块，我掌握了以下深度学习基本流程：
  - 数据加载与预处理（使用.npz 文件存储词向量索引）
  - 模型构建与部署到 GPU
  - 损失函数使用 CrossEntropyLoss，优化器选择 Adam
  - 引入 ReduceLROnPlateau 自动调整学习率，提升训练稳定性
  - 利用 tensorboardX 可视化训练过程