深度学习第三次作业报告

学号：2252540
姓名：金国栋

目录

深度学习第三次作业报告 1
一、RNN、LSTM、GRU 模型解释 3
1. RNN（Recurrent Neural Network） 3
2. LSTM（Long Short-Term Memory） 3
3. GRU（Gated Recurrent Unit） 3
二、诗歌生成的过程 5
1. 数据预处理 5
2. 构建训练数据批次 5
3. 模型搭建 5
4. 模型训练过程 6
5. 诗歌生成 6
三、训练结果与分析 7
1.生成示例展示 7
2.训练过程记录 7
3.模型最终生成输出与评估 7
4.小结 8
四、 实验总结 9
1.实验成果与亮点 9
2.存在问题与不足 9
3.后续改进方向 9

一、RNN、LSTM、GRU 模型解释

循环神经网络（Recurrent Neural Network, RNN）及其变体 LSTM（长短期记忆网络）和 GRU（门控循环单元）是处理序列数据的重要模型，广泛应用于自然语言处理任务中，如机器翻译、文本生成、情感分析等。

1. RNN（Recurrent Neural Network）

RNN 是一种具有“记忆能力”的神经网络模型。它的核心思想是：当前的输出不仅取决于当前的输入，还取决于前一时刻的输出。这使得 RNN 非常适合处理语言等有“上下文语境”的序列数据。

RNN 的结构中，每个时间步（time step）会将前一时刻的“隐藏状态（hidden state）”传递给当前时刻，同时结合当前输入一起生成新的隐藏状态和输出结果。

- 优点： 可以处理变长序列，保留上下文信息。
- 缺点： 容易产生“梯度消失”或“梯度爆炸”问题，导致长距离依赖信息无法有效传播。

2. LSTM（Long Short-Term Memory）

LSTM 是针对 RNN 的“长期依赖问题”提出的改进模型。它通过设计特殊的“门控机制（gate mechanism）”来控制信息的保留与遗忘，从而更有效地捕捉序列中的长期依赖关系。

LSTM 结构包括三个主要的门：

- 遗忘门（Forget Gate）：决定哪些旧信息要丢弃；
- 输入门（Input Gate）：决定当前信息中哪些需要保留；
- 输出门（Output Gate）：决定隐藏状态中哪些部分参与输出。

优势： 有效解决了 RNN 的梯度问题，能学习更长距离的依赖关系。

3. GRU（Gated Recurrent Unit）

GRU 是另一种改进的循环神经网络，结构相比 LSTM 更加简洁。它将“遗忘门”和“输入门”合并为一个“更新门”，并引入“重置门”来控制信息流动。

GRU 只有两个门：

- 更新门（Update Gate）：控制旧信息与新信息的融合；
- 重置门（Reset Gate）：控制当前输入和过去状态之间的关系。

特点： 参数更少、计算更快，性能在很多任务上与 LSTM 相当。

总结对比：

模型	门控机制	适合处理	优点	缺点
RNN	无门控	短期依赖序列	简单、结构清晰	容易梯度消失
LSTM	三个门	长期依赖	能记住长期信息	结构复杂
GRU	两个门	长期依赖	结构简单、性能优	灵活性略低于 LSTM

二、诗歌生成的过程

本项目通过基于 PyTorch 实现的循环神经网络（RNN）模型，完成了古诗自动生成任务。整个流程主要包括四个关键步骤：数据预处理、模型构建、模型训练和诗歌生成。

1. 数据预处理

诗歌生成模型的训练需要大量古诗数据。代码中的 process_poems1() 和 process_poems2() 函数完成了诗歌数据的清洗和向量化处理，具体包括：

- 读取原始诗歌数据（如 poems.txt 文件），过滤掉长度过短或不规则格式的诗句；
- 为每首诗前后加上 G（start token）和 E（end token），用于标识起始和终止；
- 统计所有出现过的汉字并建立词表，将每个汉字映射为整数索引（word → index）；
- 最终得到 poems_vector，即每首诗对应的字索引序列，用于神经网络的输入。

2. 构建训练数据批次

训练模型需要将原始诗歌数据按批次（batch）组织。generate_batch() 函数负责将诗句数据组织成输入 x_data 和目标 y_data：

- x_data 是原始诗句的索引序列；
- y_data 是将 x_data 向后错一位（即预测下一个字）。

3. 模型搭建

模型在 rnn.py 中定义，核心是一个两层的 LSTM 网络结构：

```
self.rnn_lstm = nn.LSTM(
    input_size=self.word_embedding_dim,
    hidden_size=self.lstm_dim,
    num_layers=2,
    batch_first=True
)
```

模型由三个主要部分组成：

- Embedding 层：将字的索引转换为稠密的词向量；
- LSTM 层：学习字与字之间的上下文关系；
- 全连接层+Softmax：输出每个位置预测的下一个字的概率分布。

模型的输出通过 LogSoftmax() 进行归一化，配合损失函数 NLLLoss() 使用。

4. 模型训练过程

训练部分由 run_training() 函数完成，主要流程如下：

- 调用 process_poems1() 加载训练数据；
- 初始化模型结构和优化器；
- 每轮迭代（epoch）中，按 batch 取出数据进行训练；
- 对每个样本逐个前向传播和反向传播计算损失；
- 每训练一定步数保存模型权重（poem_generator_rnn 文件）；
- 同时打印每轮预测结果与真实目标对比，并截图保存作业资料。

5. 诗歌生成

模型训练完毕后，通过 gen_poem() 函数进行诗歌生成：

- 输入起始字 begin_word，如“日”、“山”、“夜”等；
- 使用训练好的模型逐字预测下一个字；
- 直到遇到 E（结束标志）或生成长度超过限制；
- 利用 to_word() 将预测结果转换成汉字；
- 最终通过 pretty_print_poem() 美化输出诗歌结果。

三、训练结果与分析

本项目使用 PyTorch 实现基于 RNN 的古诗生成模型，训练结束后对模型效果进行了初步评估。

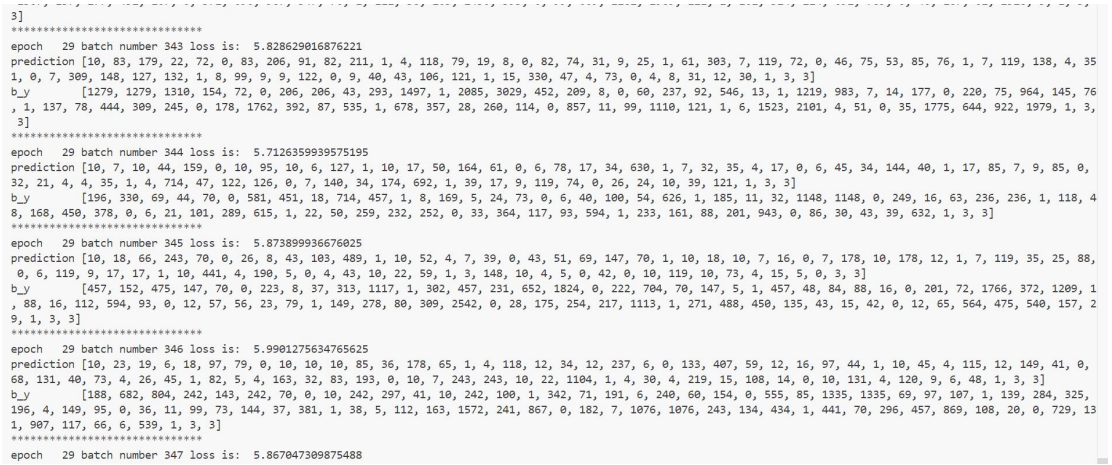
1. 生成示例展示

模型训练完成后，使用以下代码对模型进行了生成测试，给定“日”、“红”、“山”、“夜”、“湖”、“湖”、“湖”、“君”作为开头词汇。

```
pretty_print_poem(gen_poem("日"))
pretty_print_poem(gen_poem("红"))
pretty_print_poem(gen_poem("山"))
pretty_print_poem(gen_poem("夜"))
pretty_print_poem(gen_poem("湖"))
pretty_print_poem(gen_poem("湖"))
pretty_print_poem(gen_poem("湖"))
pretty_print_poem(gen_poem("君"))
```

2. 训练过程记录

训练过程在共 30 个 epoch 阶段记录了所有 batch 的 loss 以及模型的预测输出。以下是部分训练日志示例：



```
3]
******************************
epoch   29 batch number 343 loss is:  5.828629016876221
prediction [10, 83, 179, 22, 72, 0, 83, 206, 91, 82, 211, 1, 4, 118, 79, 19, 8, 0, 82, 74, 31, 9, 25, 1, 61, 303, 7, 119, 72, 0, 46, 75, 53, 85, 76, 1, 7, 119, 138, 4, 35
1, 0, 7, 309, 148, 127, 132, 1, 8, 99, 9, 9, 122, 0, 9, 40, 43, 106, 121, 1, 15, 330, 47, 4, 73, 0, 4, 8, 31, 12, 30, 1, 3, 3]
b_y       [1279, 1279, 1310, 154, 72, 0, 206, 206, 43, 293, 1497, 1, 2085, 3029, 452, 209, 8, 0, 60, 237, 92, 546, 13, 1, 1219, 983, 7, 14, 177, 0, 220, 75, 964, 145, 76
, 1, 137, 78, 444, 309, 245, 0, 178, 1762, 392, 87, 535, 1, 678, 357, 28, 260, 114, 0, 857, 11, 99, 1110, 121, 1, 6, 1523, 2101, 4, 51, 0, 35, 1775, 644, 922, 1979, 1, 3,
 3]
******************************
epoch   29 batch number 344 loss is:  5.71263599395751 95
prediction [10, 7, 10, 44, 159, 0, 10, 95, 10, 6, 127, 1, 10, 17, 50, 164, 61, 0, 6, 78, 17, 34, 630, 1, 7, 32, 35, 4, 17, 0, 6, 45, 34, 144, 40, 1, 17, 85, 7, 9, 85, 0,
32, 21, 4, 4, 35, 1, 4, 714, 47, 122, 126, 0, 7, 140, 34, 174, 692, 1, 39, 17, 9, 119, 74, 0, 26, 24, 10, 39, 121, 1, 3, 3]
b_y       [196, 330, 69, 44, 70, 0, 581, 451, 18, 714, 457, 1, 8, 169, 5, 24, 73, 0, 6, 40, 100, 54, 626, 1, 185, 11, 32, 1148, 1148, 0, 249, 16, 63, 236, 236, 1, 118, 4
8, 168, 450, 378, 0, 6, 21, 101, 289, 615, 1, 22, 50, 259, 232, 252, 0, 33, 364, 117, 93, 594, 1, 233, 161, 88, 201, 943, 0, 86, 30, 43, 39, 632, 1, 3, 3]
******************************
epoch   29 batch number 345 loss is:  5.873899936676025
prediction [10, 18, 66, 243, 70, 0, 26, 8, 43, 103, 489, 1, 10, 52, 4, 7, 39, 0, 43, 51, 69, 147, 70, 1, 10, 18, 10, 7, 16, 0, 7, 178, 10, 178, 12, 1, 7, 119, 35, 25, 88,
 0, 6, 119, 9, 17, 17, 1, 10, 441, 4, 190, 5, 0, 4, 43, 10, 22, 59, 1, 3, 148, 10, 4, 5, 0, 42, 0, 10, 119, 10, 73, 4, 15, 5, 0, 3, 3]
b_y       [457, 152, 475, 147, 70, 0, 223, 8, 37, 313, 1117, 1, 302, 457, 231, 652, 1824, 0, 222, 704, 70, 147, 5, 1, 457, 48, 84, 88, 16, 0, 201, 72, 1766, 372, 1209, 1
, 88, 16, 112, 594, 93, 0, 12, 57, 56, 23, 79, 1, 149, 278, 80, 309, 2542, 0, 28, 175, 254, 217, 1113, 1, 271, 488, 450, 135, 43, 15, 42, 0, 12, 65, 564, 475, 540, 157, 2
9, 1, 3, 3]
******************************
epoch   29 batch number 346 loss is:  5.9901275634765625
prediction [10, 23, 19, 6, 18, 97, 79, 0, 10, 10, 10, 85, 36, 178, 65, 1, 4, 118, 12, 34, 12, 237, 6, 0, 133, 407, 59, 12, 16, 97, 44, 1, 10, 45, 4, 115, 12, 149, 41, 0,
68, 131, 40, 73, 4, 26, 45, 1, 82, 5, 4, 163, 32, 83, 193, 0, 10, 7, 243, 243, 10, 22, 1104, 1, 4, 30, 4, 219, 15, 108, 14, 0, 10, 131, 4, 120, 9, 6, 48, 1, 3, 3]
b_y       [188, 682, 804, 242, 143, 242, 70, 0, 10, 242, 297, 41, 10, 242, 100, 1, 342, 71, 191, 6, 240, 60, 154, 0, 555, 85, 1335, 1335, 69, 97, 107, 1, 139, 284, 325,
196, 4, 149, 95, 0, 36, 11, 99, 73, 144, 37, 381, 1, 38, 5, 112, 163, 1572, 241, 867, 0, 182, 7, 1076, 1076, 243, 134, 434, 1, 441, 70, 296, 457, 869, 108, 20, 0, 729, 13
1, 907, 117, 66, 6, 539, 1, 3, 3]
******************************
epoch   29 batch number 347 loss is:  5.867047309875488
```

从 loss 值可以看出，虽然模型在训练后期 loss 略有下降，但整体仍处于较高水平，模型学习效果有限。

3. 模型最终生成输出与评估

从训练完成后的最终输出可以看出，模型生成的诗句多为重复、逻辑不通或结构混乱的内容，部分甚至未能正确生成诗句，仅输出“error”。

这表明当前模型尚未有效学习古诗生成的语言规律，模型质量较差。

```
inital  linear weight
error
inital  linear weight
error
inital  linear weight
不见天台去，空将此地同。
error
inital  linear weight
不见明朝人，不如天上人。
error
inital  linear weight
人生不可见，不得在人间。
error
inital  linear weight
人生不可见，不得在人间。
error
inital  linear weight
人生不可见，不得在人间。
error
inital  linear weight
不见江南客，东风吹不还。
```

4. 小结

尽管模型在形式上实现了古诗自动生成，但从训练 loss 与生成内容来看，模型学习效果不理想，存在以下问题：

- Loss 下降缓慢，收敛不足；
- 生成句子重复性强、缺乏语义逻辑；
- 多次出现“error”或生成失败，表明模型稳定性较差。

后续可以考虑如下改进方向：

- 增加训练轮数；
- 调整模型结构（如使用双层 LSTM 或 Attention 机制）；
- 引入更丰富的训练语料；
- 调整学习率、优化器等超参数。

四、实验总结

本次实验基于 PyTorch 构建了一个简易的古诗生成模型，通过使用循环神经网络（RNN）对已有古诗语料进行训练，尝试实现“以词起句，自动续写”的功能。整个实验过程涵盖了数据预处理、模型搭建、训练与测试、诗歌生成等核心流程，具备较强的实操性和学习价值。

1. 实验成果与亮点

- 完成了完整的诗歌生成模型构建与训练流程；
- 掌握了 PyTorch 中 RNN 模型的基本用法；
- 实现了以“给定起始词”生成古诗的功能；
- 对训练过程及 loss 变化有了直观认知。

2. 存在问题与不足

- 模型训练效果较差，loss 下降缓慢，生成内容重复或不通顺；
- 训练语料可能过于有限，难以有效捕捉古诗语义规律；
- 模型结构较为简单，缺乏更复杂的语言建模能力；
- 实际生成中出现大量“error”及无意义诗句，模型泛化能力不足。

3. 后续改进方向

- 提升模型结构：可尝试引入双向 LSTM、Attention 机制或 Transformer 结构，增强上下文建模能力；
- 优化训练策略：增加训练轮数、调整优化器与学习率策略；
- 扩充语料库：引入更多高质量古诗数据，提高模型训练质量；
- 引入规则约束：结合平仄、押韵等诗词规则，提升生成文本的诗意和规范性。