RNN 生成唐诗（pytorch 版）

2251499　桂欣远

一、代码补全：

LSTM 层定义：

```
########################################
# here you need to define the "self.rnn_lstm"  the input size is "embedding_dim" and the output
# the lstm should have two layers, and the  input and output tensors are provided as (batch, s
self.rnn_lstm = nn.LSTM(
    input_size=embedding_dim,
    hidden_size=lstm_hidden_dim,
    num_layers=2,
    batch_first=True
)
########################################
```

前向传播逻辑：

```
########################################
# here you need to put the "batch_input"  input the self.lstm which is defined before.
# the hidden output should be named as output, the initial hidden state and cell state set to zero.
output, (h_n, c_n) = self.rnn_lstm(batch_input)
########################################
```

二、RNN、LSTM、GRU 模型解释

1．RNN（循环神经网络）

处理序列数据的经典模型，其核心是通过循环结构将前一时刻的隐藏状态传递到当前时刻，从而捕捉序列中的时序信息。例如，在生成诗歌时，RNN 会根据已生成的字符预测下一个字符。然而，传统 RNN 存在梯度消失和梯度爆炸的问题，当序列较长时，模型难以记住远距离的依赖关系，像是诗歌中前后句的押韵或主题的一致，容易导致生成内容逻辑断裂或重复。

2．LSTM（长短期记忆网络）

个人感觉更像 RNN 的改进版本，通过引入门控机制和细胞状态解决长期依赖问题。LSTM 包含三个关键门：

遗忘门：决定哪些历史信息需要丢弃，当诗歌主题变化时，遗忘无关的上下文。
输入门：控制哪些新信息需要存储到细胞状态，记住新诗句的关键意象。
输出门：生成押韵的字符时基于当前输入和细胞状态，决定最终的隐藏状态输出。

LSTM 的细胞，允许梯度在长距离传播时保持稳定，从而有效建模诗歌中的长程结构

3．GRU（门控循环单元）

是 LSTM 的简化版本，将输入门和遗忘门合并为更新门，并引入重置门，减少了参数量，训练速度更快。重置门决定如何将过去信息与当前输入结合，更新门控制隐藏状态的更新程度。GRU 在短序列任务中表现接近 LSTM，但在复杂的长序列生成（如长篇诗歌）中，LSTM 因更精细的门控设计通常更具优势。

三、诗歌生成过程解释

（以静夜思为例子分析）

1．数据预处理与词表构建

代码文件：main.py 中的 process_poems1 函数。

步骤：

1. 读取与清洗：

   输入文件 poems.txt 的每一行格式为 标题:内容，例如：

   静夜思:床前明月光，疑是地上霜

   清洗时会移除空格、特殊符号，并过滤长度不符或格式错误的行。

2. 添加起止标记：

   每首诗被包裹为 G 内容 E，例如：

   G 床前明月光，疑是地上霜 E

3. 构建词表映射：

   统计所有字符的频率，生成 字符 → 索引 的字典。

   word_int_map = {'G': 0, '床': 1, '前': 2, '明': 3, '月': 4, ...}

4. 转换为数字序列：

   诗歌被转换为索引序列，例如：

   [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

2．模型架构

代码文件：rnn.py 中的 RNN_model 类。

模型组件：

词嵌入层：将索引映射为稠密向量，例如：

embedding = word_embedding(vocab_length=5000, embedding_dim=100)

字符“床”（索引 1）被映射为 [0.2, -0.5, ..., 0.7]（100 维向量）。

LSTM 层：

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) && \text{(遗忘门)} \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) && \text{(输入门)} \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) && \text{(输出门)} \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) && \text{(候选细胞状态)} \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t && \text{(更新细胞状态)} \\
h_t &= o_t \odot \tanh(C_t) && \text{(当前隐藏状态)}
\end{aligned}
$$

全连接层与 Softmax：

self.fc = nn.Linear(128, 5000)　　映射到词表空间

self.softmax = nn.LogSoftmax(dim=1)

3．训练过程

代码逻辑：main.py 中的 run_training 函数。

步骤：

1. 生成批次数据：

   输入序列为 [G，床，前，明]，标签序列为 [床，前，明，月]。

2. 前向传播：

   输入序列经过词嵌入层，转换为形状为 (batch_size, seq_len, 100) 的张量。

   LSTM 处理序列，输出形状为 (batch_size, seq_len, 128) 的隐藏状态。

   全连接层将隐藏状态映射到词表空间，得到形状为 (batch_size * seq_len, 5000) 的 logits。

   使用 LogSoftmax 计算概率分布：

$$P(w_t|w_{<t}) = \text{LogSoftmax}(W \cdot h_t + b)$$

3. 损失计算：

   使用负对数似然损失（NLLLoss）：

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|w_{<i})$$

4. 反向传播与优化：

   通过 RMSprop 优化器更新参数，梯度裁剪防止爆炸。

4．诗歌生成（自回归预测）

代码逻辑：main.py 中的 gen_poem 函数。

步骤示例（以生成“日”开头的诗为例）：

1. 初始化：

   输入起始字符“日”，转换为索引序列 [word_int_map["日"]] = [10]。

2. 逐步生成：

   第 1 步：

   输入 [10] → 模型预测下一个字符的概率分布。

   假设概率最高的字符是“落”（索引 15），追加到序列得到 [10, 15]。

   第 2 步：

   输入 [10, 15] → 模型预测下一个字符为“长”（索引 20），序列变为 [10, 15, 20]。

   重复：直到生成 E 或达到长度限制。

3. 输出结果：

   最终序列 [G，日，落，长，河, E] 转换为诗句：“日落长河”。

四、训练过程和结果

使用 poems 数据集训练：

```
epoch   27 batch number 250 loss is:  5.826144218444824
epoch   27 batch number 300 loss is:  5.972924709320068
finish save model
epoch   28 batch number 0 loss is:  4.572681427001953
finish save model
epoch   28 batch number 50 loss is:  5.177395343780518
epoch   28 batch number 100 loss is:  5.414235591888428
finish save model
epoch   28 batch number 150 loss is:  5.383198261260986
epoch   28 batch number 200 loss is:  5.26245641708374
finish save model
epoch   28 batch number 250 loss is:  5.81923770904541
epoch   28 batch number 300 loss is:  5.975822448730469
finish save model
epoch   29 batch number 0 loss is:  4.590999126434326
finish save model
epoch   29 batch number 50 loss is:  5.169581890106201
epoch   29 batch number 100 loss is:  5.410725116729736
finish save model
epoch   29 batch number 150 loss is:  5.381172180175781
epoch   29 batch number 200 loss is:  5.262465476989746
finish save model
epoch   29 batch number 250 loss is:  5.8225250244140625
epoch   29 batch number 300 loss is:  5.97562837600708
finish save model
```

上图使用 poems 训练时的过程：为了加快训练进度，要尽可能得减少 print 和 save_model 的次数。所以将每个 batch 的输出注释掉了，并每 50 个 batch 才输出一次 loss，同时每 100 个 batch 才保存一次。

结果如下图：

```
日：
C:\Users\gxy\Desktop\chap6_R
  rnn_model.load_state_dict(
日挂上郎神，玉皇无事不相关。
一笑不须轻万事，一生曾是是长安。
红：
不见一年今日尽，一时应笑此中来。
山：
山水人家人不回，一声清夜不相看。
一望不须惆相笑，不须惆玉两三千。
夜：
夜裴回归去时，不如不见别离情。
湖：
湖上见天风，风光一声入金楼。
君：
君上天上来，不如长在一重重。
山川紫气金龙去，风送残阳入楚山。
```

使用 tangshi 数据集训练：

由于使用 CPU 来训练模型比较慢，就使用了较小的 tangshi 作为数据集训练，训练的截图如下：

```
epoch   0 batch number 5 loss is:  7.735347747802734
prediction [3, 3, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 3, 19, 19, 19, 11, 19, 11, 19, 19, 11]
b_y        [1583, 1584, 75, 8, 91, 1571, 891, 88, 2357, 102, 426, 53, 4, 317, 161, 10, 291, 13, 801, 438, 393, 1, 1]
******************************
epoch   0 batch number 6 loss is:  7.570469856262207
prediction [24, 37, 37, 1, 1, 1, 37]
b_y        [140, 446, 620, 923, 447, 1, 1]
******************************
epoch   1 batch number 0 loss is:  7.344920635223389
finish  save model
prediction [28, 32, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
b_y        [1266, 182, 42, 556, 810, 264, 1679, 811, 136, 346, 81, 1680, 37, 1681, 1, 1]
******************************
epoch   1 batch number 1 loss is:  7.091061592102051
prediction [14, 12, 107, 107, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12]
b_y        [471, 145, 495, 2, 1817, 373, 1357, 273, 25, 52, 58, 846, 99, 1360, 197, 1818, 672, 240, 1361, 228, 1, 1]
******************************
epoch   1 batch number 2 loss is:  7.407339096069336
```

加载模型推理得到的结果如下：

```
C:\Users\lenovo\.conda\envs\d2l\python.exe C
inital  linear weight
日天上游鱼乐何所休清流有问有。
inital  linear weight
红风白舟还下听万头物天南公马问行。
inital  linear weight
inital  linear weight
inital  linear weight
湖为春日过空独与苍风风坐野。
inital  linear weight
湖为春日过空独与苍风风坐野。
inital  linear weight
湖为春日过空独与苍风风坐野。
inital  linear weight
君子见与纷纷纷纷纷纷纷纷其万里与其沙中儿见头主儿。
```

“山”和“夜”没有输出,可能是因为使用的 tangshi 数据集太小，没有充分学习这些词的上下文关系导致山和夜提前生成了 E，提前结束。

可以发现使用 poems 数据集得到的模型生成诗句的效果要远好于使用 tangshi 的效果，这是数据集信息量大小差距导致的必然结果。

五、实验总结

本次实验使用 PyTorch 实现了基于 RNN 的唐诗生成模型，并通过两种不同大小的数据集（poems 和 tangshi）分别在 GPU 和 CPU 环境下进行训练，以比较模型的效果。

在实验中，首先对唐诗数据集进行了预处理，包括清洗数据、添加起止标记、构建词表映射以及将诗歌转换为数字序列等步骤。接着，构建了一个包含词嵌入层、LSTM 层、全连接层和 Softmax 激活函数的 RNN 模型。为了提高训练效率，利用 GPU 加速训练过程，将模型和数据迁移到 GPU 上进行计算。

在训练过程中，通过生成批次数据、前向传播、损失计算和反向传播等步骤，利用RMSprop 优化器更新模型参数，并采用梯度裁剪防止梯度爆炸。为了加快训练进度，减少了打印和保存模型的次数，每 50 个 batch 输出一次 loss，每 100 个 batch 保存一次模型。

实验结果表明，使用较大的 poems 数据集训练的模型生成的诗句效果较好，而使用较小的 tangshi 数据集训练的模型在生成以“山”和“夜”开头的诗歌时出现了提前终止的问题。这可能是由于 tangshi 数据集较小，模型未能充分学习到这些词的上下文关系，导致生成过程提前结束。

通过比较两种数据集和不同硬件环境下的训练效果，发现数据集的大小对模型性能有显著影响。较大的数据集能够提供更丰富的信息，帮助模型更好地学习诗歌的生成规律。同时，使用 GPU 训练可以显著提高训练效率，加快模型收敛速度。

总的来说，本次实验成功实现了基于 RNN 的唐诗生成模型，并通过不同条件下的训练和生成结果比较，验证了数据集大小和硬件环境对模型性能的影响。