**唐诗生成实验报告**

2252930 周进

1. **模型介绍**

**1.1 RNN循环神经网络**

**1.1.1基本结构**

RNN的基本单元是循环神经元，它不仅接收来自输入层的信号，还接收来自上一时刻的自身输出，形成了一个循环结构。这种结构使得RNN能够对序列数据中的历史信息进行记忆和利用。一个典型的RNN由输入层、隐藏层和输出层组成，其中隐藏层的神经元具有循环连接。

**1.1.2工作原理**

在RNN中，每个时间步的隐藏状态不仅取决于当前输入，还取决于上一时间步的隐藏状态。具体来说，对于时间步t，隐藏状态的计算公式为：



其中，*xt*​是当前时间步的输入，*ht*−1​是上一时间步的隐藏状态，*Wxh*​和*Whh*​是权重矩阵，*bh*​是偏置项，*σ*是激活函数（通常为tanh或ReLU）

**1.1.3优缺点**

RNN能够处理变长的序列数据，并且能够捕捉序列中的时间依赖关系。然而，标准的RNN存在一些问题，如梯度消失和梯度爆炸问题。在训练过程中，当序列长度较长时，梯度在时间步的反向传播中可能会逐渐变小（消失）或变大（爆炸），导致模型难以学习到长期依赖关系。

**1.2 LSTM长短期记忆网络**

**1.2.1基本结构**

为了克服RNN中的梯度消失问题，Hochreiter和Schmidhuber在1997年提出了长短期记忆网络（Long Short-Term Memory，LSTM）。LSTM通过引入特殊的单元结构——记忆单元（memory cell），以及三个门控机制——输入门（input gate）、遗忘门（forget gate）和输出门（output gate），来更好地控制信息的流动。

**1.2.2工作原理**

**遗忘门**：决定记忆单元中哪些信息需要被丢弃，通过一个sigmoid函数计算，输出0到1之间的值，表示保留或丢弃的比例。

**输入门**：决定哪些新信息需要被存储到记忆单元中，包括两部分：一部分是输入门的激活值，另一部分是候选记忆单元的值，通过tanh函数计算。

**记忆单元更新**：根据遗忘门和输入门的输出，更新记忆单元的值，融合上一时刻的信息和当前时刻的新信息。

**输出门**：决定从记忆单元中输出哪些信息，通过一个sigmoid函数计算，最终的隐藏状态基于输出门的激活值和记忆单元的值。

**1.2.3优缺点**

LSTM能够有效地学习到序列中的长期依赖关系，适用于处理具有长时间间隔的序列数据。然而，LSTM的结构相对复杂，训练时间较长，且参数数量较多，可能导致过拟合问题。

**1.3 GRU门控循环单元**

**1.3.1基本结构**

门控循环单元（Gated Recurrent Unit，GRU）是另一种改进的RNN变体，由Cho等人在2014年提出。GRU在结构上比LSTM更为简单，它将遗忘门和输入门合并为一个更新门（update gate），并且没有单独的记忆单元，而是直接对隐藏状态进行更新。

**1.3.2工作原理**

**更新门**：决定当前时间步的隐藏状态有多少信息需要从上一时间步的隐藏状态继承，通过一个sigmoid函数计算。

**重置门**：决定是否在当前时间步重置上一时间步的隐藏状态，通过一个sigmoid函数计算。

**候选隐藏状态**：根据重置门的输出，计算候选隐藏状态，通过tanh函数计算。

**隐藏状态更新**：根据更新门的输出，融合上一时间步的隐藏状态和当前时间步的候选隐藏状态。

**1.3.3优缺点**

GRU的结构相对简单，参数数量较少，训练速度较快，同时能够有效地缓解RNN中的梯度消失问题。然而，与LSTM相比，GRU在处理某些具有非常长期依赖关系的序列数据时，可能效果稍逊一筹。

1. **诗歌生成过程**

以下将从数据处理、模型设计、训练过程以及生成过程这四个过程进行详细阐述。

**2.1数据处理**

首先，我们从文件 poems.txt 中读取唐诗文本数据，对每首诗进行预处理。具体步骤包括去除过长的诗歌、为每首诗添加开始 (bos) 和结束 (eos) 标记，统计词汇频率并构建词汇表。接着，将诗歌中的每个字符转换为其对应的词汇表索引。最后，使用 TensorFlow 的 tf.data.Dataset 创建数据集，包含输入序列、目标序列及各自的长度信息。为了确保模型能有效学习，我们对数据进行了随机打乱、填充至相同长度并分批处理。

2.2模型设计

在模型设计阶段，我们采用了一个简单的 RNN 结构。该模型由嵌入层、RNN 层和全连接层组成。嵌入层负责将输入字符转换为向量表示；RNN 层用于捕捉字符间的上下文关系；全连接层则将隐藏状态映射到词汇表大小的空间，以预测下一个字符的概率分布。此外，模型还定义了 get\_next\_token 方法，用于根据当前字符和隐藏状态逐步生成新的字符。

2.3训练过程

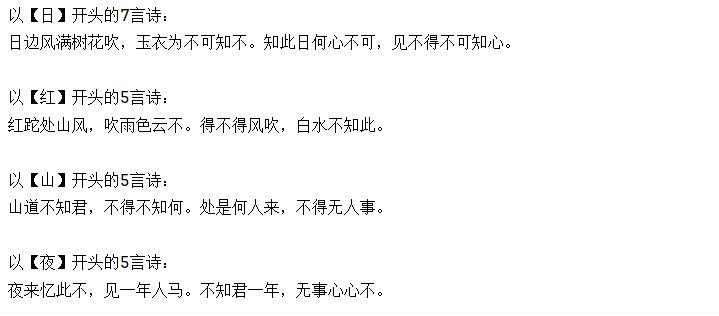
训练过程中，我们定义了损失函数和优化器。损失函数采用了稀疏交叉熵损失，以衡量模型预测与实际标签之间的差距。优化器选择了 Adam，其学习率设置为 0.0005。训练时，每次从前一批次数据中提取输入序列、目标序列及其长度，计算前向传播结果及相应的损失值，并通过反向传播更新模型参数。经过多轮迭代，模型逐渐学会了如何生成符合唐诗格式的诗句。

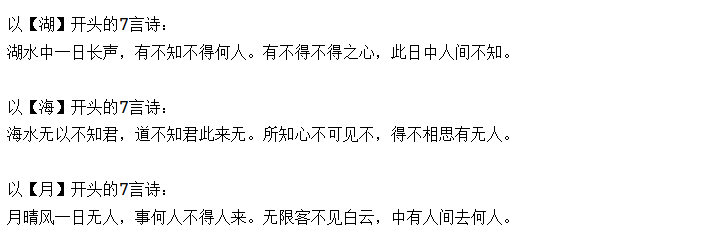
2.4生成过程

完成模型训练后，我们实现了诗歌生成功能。通过指定起始字符，利用 get\_next\_token方法，模型会逐步生成后续字符，直至完成一首完整的唐诗。在此过程中，我们会跳过一些不希望出现在诗歌中的字符（如非法标点符号等）。最终，程序会按照唐诗的传统格式（五言或七言），输出一首或多首由RNN模型生成的唐诗。

1. **诗歌生成结果**

利用训练好的模型生成了以开头词汇是 “ 日 、 红 、 山 、 夜 、 湖、 海 、 月 ”的五言或七言唐诗，结果如下图所示。





1. **实验总结**

通过本次实验，我探索了使用循环神经网络（RNN）模型自动生成唐诗的方法。整个过程涵盖了从数据准备、模型设计、训练优化到诗歌生成的各个关键环节，并最终得到了一些初步成果。

在数据处理阶段，对原始唐诗文本进行了细致的预处理工作，包括字符级别的分词、构建词汇表及索引化等步骤，为后续模型训练奠定了基础。模型方面，我们采用了一个简单的RNN架构，该模型由嵌入层、RNN层和全连接层组成，能够有效地捕捉输入序列中的上下文信息并进行预测。

训练过程中，随着迭代次数的增加，模型的损失值呈现逐渐下降的趋势，从最初的8.82降至最后的5.30左右。这表明模型在学习唐诗的结构和风格上取得了一定的进步，但仍有提升空间。具体而言，在前几个epoch中，模型的学习速度较快，损失值迅速下降；而在后期，虽然继续有所改善，但幅度相对较小，显示出模型可能开始接近其性能上限。

基于训练好的模型，我尝试生成了几首不同类型的唐诗，尽管这些诗歌在形式上基本符合五言或七言的要求，但在语义连贯性和逻辑性方面仍存在一定的局限性。例如，“日边风满树花吹，玉衣为不可知不”这样的句子虽然具有一定的诗意美感，但由于缺乏明确的意义指向，使得整体表达略显晦涩难懂。此外，部分诗句出现了重复用词（如“不得”、“不知”），反映出模型在词汇多样性和创新性上的不足。

总体来看，本实验成功地展示了如何利用深度学习技术实现自动唐诗创作的基本流程，并获得了一些具有初步艺术价值的作品。