基于神经网络的语言模型

团队成员:

李俊辉 U202217182　贡献：25%

穆文轩 U202217190　贡献：25%

陈懿凯 U202217173　贡献：25%

农凯方 U202217192　贡献：25%

概要

本研究旨在基于中文古诗文本数据构建一个字符级别的语言模型 ,探索并评估不同递归神经网络(RNN)结构在生成古诗任务中的性能。采用了 LSTM(长短时记忆网络)和 GRU(门控循环单元)两种常见的 RNN 结构来构建语言模型，使用 PyTorch 作为深度学习框架实现。模型输入为单个汉字字符，输出为下一个可能的字符，从而逐字生成文本序列。为了衡量模型的生成效果，使用困惑度（Perplexity）作为主要评估指标，困惑度越低代表模型预测下一个字符的能力越强。本文通过实验分析不同模型结构和超参数设置对困惑度的影响，并探讨基于深度学习的语言模型在古诗生成任务中的有效性和应用前景。实验结果表明，LSTM 和 GRU 模型均能够有效捕获古诗的语法结构和风格特征，但在困惑度和生成质量上存在一定的差异。

介绍

语言模型在自然语言处理(NLP)领域中扮演着重要角色，它们被广泛应用于机器翻译、文

本生成、自动摘要等任务中，直接影响系统的理解和生成能力。近年来，基于深度学习的方法，尤其是递归神经网络（RNN）及其变种的广泛应用，使得语言模型在捕捉语义和语言规律方面取得了显著进展。中文古诗作为中国传统文化的瑰宝，包含丰富的语言表达和复杂的韵律结构，其生成和理解不仅是语言学和文学研究的一个重要课题，也是推动文化传承和智能文本生成领域发展的重要方向。因此，基于深度学习构建一个能够生成符合古诗风格的语言模型，具有重要的学术价值和实际意义。

在深度学习领域，LSTM（长短时记忆网络）和 GRU（门控循环单元）是两种常用的 RNN 变体，它们通过特殊的门控机制来有效解决传统 RNN 在长序列数据处理时的梯度消失和爆炸问题，能够更好地捕捉文本序列中的长期依赖关系。这使得它们成为文本生成任务的主流方法之一，尤其适合处理像古诗这样具有高度结构化和复杂语言规律的文本数据。

本研究基于唐诗文本数据，构建了一个字符级别的语言模型，旨在探索如何利用 LSTM 和 GRU 两种模型结构生成符合中文古诗风格和韵律的文本。研究采用 PyTorch 作为深度学习框架，对模型进行设计和实现，模型输入为单个汉字字符，预测输出为下一个可能的字符，逐字生成完整的诗句。字符级别的模型能够更好地捕捉汉字的细粒度特征和古诗的特有韵律，这对于生成符合唐诗风格的文本尤为重要。

在模型评估方面，本文采用困惑度（Perplexity）作为主要评价指标。困惑度是衡量语言模型性能的一个重要指标，它表示模型对测试数据的平均不确定性。较低的困惑度意味着模型更好地预测了下一个字符，从而能够生成更连贯和符合语法规则的诗句。通过困惑度的变化，我们可以评估模型在训练过程中的收敛情况和泛化能力，并比较不同模型结构和超参数对模

型性能的影响。

本文的研究不仅对 LSTM 和 GRU 两种模型在中文古诗生成任务中的性能进行了比较分析，还通过实验探讨了模型的超参数设置（如层数、隐藏单元数量、学习率等）对困惑度和生成质量的影响。实验结果显示，两种模型结构均能够有效地学习古诗的语言规律和韵律特征，但在困惑度和生成的诗句质量上存在差异。通过这些实验分析，本文为中文古诗生成任务中的深度学习模型设计提供了新的见解和优化建议，推动了这一领域的发展和应用。

综上所述，本研究不仅为古诗生成提供了一个有效的解决方案，还为将来在其他中文文本生成和自然语言处理任务中的应用提供了参考。通过结合现代深度学习技术与传统文化数据，本研究展示了智能文本生成在文化传承和创新中的巨大潜力。

相关工作

1. **背景介绍：**

- 递归神经网络（Recurrent Neural Networks，RNNs）是一类适用于序列数据的深度学习模型，广泛应用于自然语言处理（NLP）、语音识别、机器翻译等领域。由于标准 RNN 在处理长序列时容易出现梯度消失或梯度爆炸的问题，许多改进的变体相继提出，包括长短期记忆网络（LSTM）和门控递归单元（GRU）。这些变体通过引入门控机制，改善了对长序列依赖关系的建模能力。
- 在本次大作业中，我们的项目使用 PyTorch 构建了一个字符级别的语言模型，旨在训练和评估基于《唐诗》文本数据的语言模型。模型支持 LSTM 和 GRU 两种结构，并

通过困惑度（Perplexity）来评估模型性能。

2. **理论框架：**

- LSTM（长短期记忆网络）

(1)基本原理：

LSTM 是对标准 RNN 的改进，旨在解决梯度消失和梯度爆炸问题。其核心在于引入了三个门控机制——输入门、遗忘门和输出门。

- 输入门：决定当前输入的信息有多少被存入细胞状态。

- 遗忘门：控制从细胞状态中丢弃哪些信息。

- 输出门：确定从细胞状态中提取哪些信息用于输出。

(2)优势：

- 长期依赖：能够捕捉较长时间的依赖关系，适用于复杂的序列任务。

- 稳定性：通过门控机制，稳定了梯度的传递，减轻了梯度消失问题。

- GRU（门控递归单元）

(1)基本原理：

GRU 是一种简化的 LSTM，结合了 LSTM 的门控机制但减少了计算复杂度。主要包含两个门——重置门和更新门。

- 重置门：决定如何将过去的信息融入当前计算。

- 更新门：控制当前状态有多少被保留，并决定如何更新隐藏状态。

(2)优势：

- 计算效率：结构较 LSTM 简化，计算和内存需求较少。

- 性能平衡：在很多任务中，GRU 与 LSTM 性能相当，但训练更为高效。

LSTM 和 GRU 都解决了标准 RNN 在处理长序列时的不足，但 GRU 在简化计算方面提供了更高的效率。

3. **主要研究：**

- 字符级语言模型的研究是深度学习语言建模的研究热点之一 ,也产生了许多研究成果，前人提出过许多字符级的语言模型方案。
- RNN(循环神经网络)模型是基于当前的状态和当前的输入来对下一时刻做出预判。而 LSTM (长短时记忆网络) 模型则可以记忆距离当前位置较远的上下文信息。我们根据上述模型来进行古诗词的生成模型训练 , 为此我们准备好了大量唐诗的数据集。

方法

- **研究设计** :语言模型仅仅对句子出现的概率进行建模 , 并不尝试去理解句子的内容含义。语言模型可以根据句子的一部分预测下一个词 , 简言之 , 语言模型就是判断一句话是否在语法上通顺。为了实现一个字符级别的语言模型 , 我们使用了 PyTorch 框架来构建和训练模型。模型支持 LSTM 和 GRU 两种结构 , 并通过困惑度 (Perplexity) 来评估模型的优劣。
- **数据收集**：老师为我们准备了 poetryFromTang.txt 数据集
- **技术细节**：

工具：PyTorch 深度学习框架

主要方法：数据预处理（分词、词嵌入等），利用 PyTorch 构建和训练模型，以及模型评估等等。

实验

关键步骤：

1. **数据预处理**：
   - 读取数据：从文件 poetryFromTang.txt 中读取唐诗文本数据。
   - 创建字符集：提取文本中的所有唯一字符，创建 int2char 和 char2int 字典用于字符与整数的映射。
   - 整数编码：将文本中的每个字符转换为整数编码，得到一个长整型张量 encoded。

2. **数据批次生成**：
   - one_hot_encode：将整数编码的张量转换为 one-hot 编码的张量，形状为 [batch_size, seq_length, n_labels]。
   - get_batches：从编码后的数据中生成批次。每个批次的输入 x 和目标 y 的形状都是 [batch_size, seq_length]。函数首先将数据整合到适合的形状，然后生成数据批次。

3. **模型定义**：
   - 初始化：构造函数中初始化模型的各个参数，包括隐藏层单元数、层数、丢弃率等。
   - forward 方法：定义了前向传播的过程。根据选择的模型类型（LSTM 或 GRU）

进行计算，并通过全连接层输出预测结果。

- init_hidden 方法：初始化隐藏状态。

4. **训练过程**：
   - 训练函数：负责训练模型，包括数据分割、损失计算、反向传播和梯度裁剪。每 print_every 次迭代输出一次训练和验证损失，以及计算并输出困惑度。
5. **实例化和训练**：
   - 模型实例化：创建一个 CharRNN 实例，使用 LSTM 结构。
   - 训练：调用 train 函数进行训练。

结果

1. **训练设置**：
   - **数据**：数据集包含从 poetryFromTang.txt 文件中加载的唐诗文本。对文本数据进行了处理，以创建一个字符级语言模型。
   - **字符编码**：将字符编码为整数，并使用独热编码（one-hot encoding）进行模型训练。
   - **模型**：实现了一个字符级 RNN 模型，包含 LSTM 或 GRU 单元。网络参数如下：
     - 隐藏层大小：256
     - 层数：2
     - Dropout 概率：0.5
     - 学习率：0.001
   - **批处理**：数据分成批次，每批次的大小为 12，序列长度为 32。

- **困惑度**：

  LSTM: 363.185302734375

  GRU: 353.3578186035156

2. **训练过程**：
   - **训练轮数**：训练运行了 10 轮。
   - **验证分割**：数据按照 90%用于训练，10%用于验证进行分割。
   - **损失计算**：训练过程中使用交叉熵损失（CrossEntropyLoss）来计算损失。
3. **模型性能**：
   - **损失和困惑度**：通过验证集的损失和困惑度来评估模型性能。
     - **损失**：衡量模型预测的准确性，损失越低表示性能越好。
     - **困惑度**：衡量模型预测的概率分布与实际字符分布的匹配程度。计算方法为 exp(loss)。
4. **结果总结**：
   - **LSTM 与 GRU**：结果将显示 LSTM 和 GRU 在训练和验证损失及困惑度上的表现对比。通常情况下，LSTM 在捕捉长期依赖性方面表现更好，而 GRU 在计算效率上可能具有优势。但具体性能需要通过实际结果来验证。

结论

1. **模型选择**：

- **LSTM** 和 **GRU** 都适用于捕捉字符级语言建模中的序列依赖性。然而，实际结果将决定哪种模型更适合此数据集。通常 LSTM 在处理长期依赖性时更为有效，而 GRU 在计算效率方面可能更优。

2. **数据处理**：

- 实现中对文本数据进行了有效的处理，包括独热编码和批处理。确保数据正确批处理并输入模型对于有效的训练至关重要。

3. **训练与评估**：

- 通过多轮训练和监控损失及困惑度来评估模型性能。困惑度较低表示模型在预测下一个字符的能力上表现更好。

4. **未来工作**：

- **超参数调优**：调整学习率、dropout 率和隐藏层大小等超参数可能进一步提升模型性能。
- **扩展验证**：实施额外的验证指标或在不同数据集上进行测试可以提供更多关于模型泛化能力的见解。
- **高级架构**：探索更高级的架构或变体，例如双向 LSTM 或注意力机制，可能会取得更好的结果。

总的来说，这些结果将指导我们判断 LSTM 和 GRU 模型在字符级语言建模任务中的适用性，并指出进一步实验和改进的方向。