作业 4 报告：Seq2Seq 与 Transformer 模型实现文本生成任务

吴頓
wd_0509@163.com

Abstract

利用给定语料库（金庸小说语料），用 Seq2Seq 与 Transformer 两种不同的模型来实现文本生成的任务（给定开头后生成武侠小说的片段或者章节），并对比讨论两种方法的优缺点。

Introduction

给定开头或片段的文本生成任务通常称为文本补全（Text Completion）或续写任务（Text Continuation）。这种任务要求模型根据给定的起始文本，生成后续的内容，使得生成的文本在语义和语法上连贯。现在有一些常用模型：

（1）Seq2Seq 模型：

基于 RNN 的序列到序列模型，包含编码器和解码器结构，用于将输入序列映射到输出序列，通常用于机器翻译和对话生成。

（2）Transformer 模型：

基于注意力机制的模型，可以高效处理长距离依赖关系，常用于机器翻译、文本摘要、对话生成等，如原始的 Transformer 模型，BERT，GPT，T5 等。

（3）GPT 系列（Generative Pre-trained Transformer）：

这些模型（如 GPT-2、GPT-3）在大规模文本数据上进行预训练，具有强大的生成能力。它们能够根据给定的文本片段生成合适的后续内容。

（4）T5（Text-To-Text Transfer Transformer）：

这个模型将所有任务转换为文本到文本的形式，可以用于生成任务，包括文本补全。

（5）BERT 及其变体：

虽然 BERT 主要用于理解任务，但通过调整也可以用于生成任务。

本文使用 Seq2Seq 与 Transformer 模型来实现文本生成的任务。

Methodology

Part 1:Seq2Seq

Seq2Seq（Sequence to Sequence），是一种用于处理序列数据的模型，广泛应用于自然语言处理任务，例如机器翻译、文本摘要、对话系统等。Seq2Seq 模型的核心思想是将一个序列映射到另一个序列，通常由编码器（Encoder）和解码器（Decoder）两部分组成。

Seq2Seq 模型的基本架构：

编码器（Encoder）：编码器接收输入序列（例如，一个句子中的单词序列），并将其转换为一个固定长度的上下文向量（Context Vector）。典型的编码器可以是 RNN（如 LSTM 或 GRU）层的堆叠。

解码器（Decoder）：解码器从上下文向量开始，并生成目标序列（例如，翻译后的句子）。解码器通常也是 RNN 层的堆叠，输出每一个时间步的预测，并将其作为下一个时间步的输入。

注意力机制（Attention Mechanism）：由于固定长度的上下文向量可能导致信息丢失，引入了注意力机制，使解码器在生成每个时间步的输出时，可以动态关注编码器输出的不同部分。注意力机制通过计算每个输入时间步的重要性权重，来生成一个加权的上下文向量。

Part 2：Transformer

Transformer 模型是自然语言处理和许多其他序列处理任务中的一种革命性模型。Transformer 模型引入了一种新的架构，依赖于注意力机制（Attention Mechanism），而不是传统的循环神经网络（RNN）或卷积神经网络（CNN），大大提高了模型的训练效率和性能。Transformer 模型的结构如 Figure 1 所示：

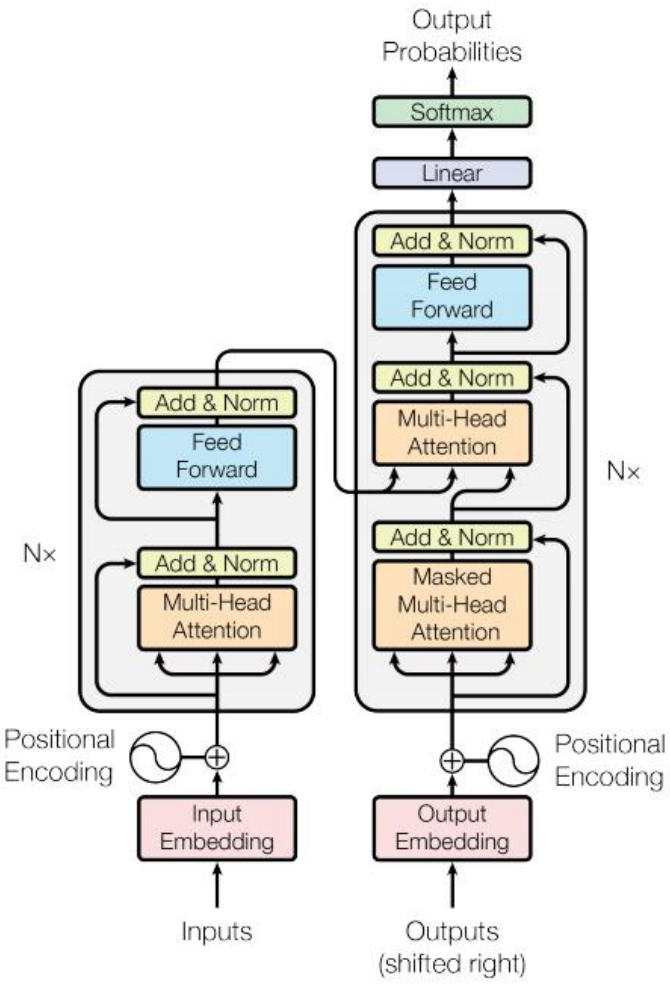

Figure 1：Transformer 模型架构

Transformer 模型的关键组件：

Self-Attention 机制：Self-Attention（自注意力）机制允许模型在处理每个输入元素时考

虑输入序列中的所有其他元素。这种机制通过计算查询（Query）、键（Key）和值（Value）的加权和来实现。

Self-Attention 的计算公式：

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

多头注意力机制（Multi-Head Attention）：多头注意力机制通过多个注意力头（Attention Head）并行计算，使模型能够关注不同的表示子空间。每个头都有自己的查询、键和值矩阵，最终的输出通过拼接这些头的输出并进行线性变换得到。

位置编码（Positional Encoding）：由于 Transformer 模型没有顺序处理输入序列的机制（不像 RNN 有时间步的顺序），它通过位置编码来引入序列顺序信息。位置编码通常是固定的正弦和余弦函数，也可以是可学习的参数。

编码器（Encoder）和解码器（Decoder）：编码器由多个相同的层堆叠而成，每层包含一个多头自注意力子层和一个前馈神经网络子层。解码器也由多个相同的层堆叠而成，但每层有额外的一个多头注意力子层，用来关注编码器的输出。

Experimental Studies

Part 1:Seq2Seq 模型生成文字

数据处理

从语料库中读取多个文本文件，将所有小说内容合并为一个大的字符串序列，并进行预处理，包括去除广告文本、空格、换行符和特殊字符。然后将处理后的文本数据转换为字符索引，创建字符到索引的映射和索引到字符的映射。该数据集类还定义了序列长度，并实现了获取指定索引处的输入序列和目标序列的方法，以便模型进行训练。使用 DataLoader 将 TextDataset 实例化为可迭代的数据加载器，提供小批量的数据用于模型训练，提高训练效率。最后保存并加载数据集。

模型定义

1、编码器（Encoder）

结构：

嵌入层（Embedding Layer）：将输入的字符序列转换为固定维度的嵌入向量。

LSTM 层：一个多层 LSTM 网络，用于处理嵌入向量序列，生成隐藏状态和细胞状态。

功能：

将输入的字符序列转换为一系列的隐藏状态和细胞状态，作为解码器的初始状态。

2、解码器（Decoder）

结构：

嵌入层（Embedding Layer）：将目标字符序列中的字符转换为嵌入向量。

LSTM 层：一个多层 LSTM 网络，用于处理嵌入向量和编码器的隐藏状态及细胞状态。

全连接层（Fully Connected Layer）：将 LSTM 的输出映射到词汇表的维度，用于预测下一个字符的概率分布。

功能：

使用前一个时间步的字符、编码器的隐藏状态和细胞状态生成当前时间步的字符预测，并更新隐藏状态和细胞状态。

3、序列到序列（Seq2Seq）模型

结构：

包含一个编码器和一个解码器。
功能：
在训练时，通过教师强制（teacher forcing）机制使用目标序列中的字符作为解码器的输入；
在预测时，使用解码器的前一个输出作为当前时间步的输入。
将整个序列的预测结果作为输出，逐步生成字符序列。

模型训练

损失函数和优化器：使用交叉熵损失函数（CrossEntropyLoss）来计算预测序列和目标序列之间的损失。使用 Adam 优化器，设置学习率为 0.001，用于优化模型参数。

训练循环：设置训练的轮次数（epochs）和用于记录损失的列表。将模型设置为训练模式（model.train()），以便在训练过程中启用 dropout 等正则化机制。

每个训练轮次：初始化当前轮次的损失总和。
遍历数据加载器中的每个批次：获取批次的源序列（source）和目标序列（target），并将它们移动到指定设备。

前向传播：将源序列和目标序列传递给 Seq2Seq 模型，生成输出序列。

输出处理：将输出序列和目标序列重新形状以匹配损失函数的输入格式。

计算损失：使用交叉熵损失函数计算输出序列和目标序列之间的损失。

反向传播和优化：清除梯度，进行反向传播，更新模型参数。记录每个批次的损失并累加到当前轮次的损失总和中。

保存当前轮次的模型状态和优化器状态，以便后续加载和继续训练。

文本生成

使用训练好的模型根据给定的起始字符串生成指定长度的新文本序列。在生成过程中，函数首先将起始字符串转换为字符索引序列，然后通过编码器和解码器逐步生成每个字符。在每个时间步，解码器使用前一个时间步的输出作为当前时间步的输入，直到生成完整的文本序列。这个过程通过循环迭代进行，生成的字符逐步添加到输出文本中，直至达到预定长度（500 个字符）。

输入文本：他跨下的枣红马奔驰了数十里地，早已筋疲力尽，在主人没命价的鞭打催踢之下，逼得气也喘不过来了，这时嘴边已全是白沫，猛地里前腿一软，跪倒在地。那汉子用力一提缰绳

输出文本：他跨下的枣红马奔驰了数十里地，早已筋疲力尽，在主人没命价的鞭打催踢之下，逼得气也喘不过来了，这时嘴边已全是白沫，猛地里前腿一软，跪倒在地。那汉子用力一提缰绳归藩佛师奉报德沐，法法家驾归令教叔令举曰佑，归问嘿教夺师爵令令帮狐叔赐，，善派正令获先心佑保令先笑宣任’获谢佑呢善允尊德令后执托难矣。　予至令碌忙念，，酬佛替可，言佛施获佛

，奏追南哉叔赔南难南叔先问佑忙愿当愿师忙起宣，佛辞歉曰求，赔降，令尊归施德罢归归禄，，，南想获“托，保嘻于佛，南佑归酬令德令劫愿忙佛，予盟令令德南至祷佛道吧任驸忙可

辞善，归，赔归昆赔想获愿获托冲替，至忙佑令归佛慢归归扶，令答，佑至佛佑于令酬慢佛降爵酬尊南师归佛救难去赔圣。师施酬德佑佛诚，获报狐赔论赔求归帝返矣叔归获佛酬师令菩后佛，．酬南令佛归啦藩帝辞，宣于南，愿佑释归佑归爵赔于可佑，酬菩晋可尊禀归佛归归悟归赔。，。盟南尊酬南爵南赔获令佛爵酬获令，佛赔宗，令佛佛教南帝归释归奉归贵愿圣汝佑。令逆令获宣。诏至令佑南崇南。叔，佛答归逆南南令酬南，归尊南佛叔南冲忙驾酬南佛至辞至帮盟谅佑归佛圣佛佑归骏愿奏向奉归，归，归酬圣佑报酬归佛尊帝佛帝，令。菩归扬南佛归宾酬藩，归爵佛。南佛

Part 2:Transformer 模型生成文字

数据处理

数据处理方式与 Part 1 相同，从语料库中读取多个文本文件，将所有小说内容合并为一个大的字符串序列，并进行预处理，包括去除广告文本、空格、换行符和特殊字符。然后将处理后的文本数据转换为字符索引，创建字符到索引的映射和索引到字符的映射。该数据集类还定义了序列长度，并实现了获取指定索引处的输入序列和目标序列的方法，以便模型进行训练。使用 DataLoader 将 TextDataset 实例化为可迭代的数据加载器，提供小批量的数据用于模型训练，提高训练效率，最后保存并加载数据集。

模型定义

TransformerModel 类：

嵌入层：使用 nn.Embedding 将字符索引转换为指定维度的嵌入向量。

位置编码：使用自定义的 create_positional_encoding 方法生成固定的正弦和余弦位置编码，以引入序列的位置信息。

Transformer 层：使用 nn.Transformer，包括多个编码器和解码器层，每层包含多头自注意力机制和前馈神经网络。

全连接层：使用 nn.Linear 将 Transformer 层的输出映射到词汇表的大小，以便进行字符预测。

create_positional_encoding 方法：生成位置编码矩阵，用于在输入嵌入中引入位置信息。位置编码使用正弦和余弦函数，依赖于序列中的位置和嵌入维度。

forward 方法：定义前向传播过程，包括嵌入和位置编码的加法操作，将输入序列和目标序列分别通过 Transformer 层处理，并通过全连接层生成预测输出。

模型训练

数据加载：加载处理后的文本数据集，并使用 DataLoader 将数据集分批次加载，设置批次大小为 64。

模型初始化：设置模型参数，包括词汇表大小、嵌入维度、隐藏层大小、层数和多头数。实例化 TransformerModel 模型，并移动到 GPU（如果可用）。

损失函数和优化器：使用 nn.CrossEntropyLoss 作为损失函数，用于计算预测字符与目标字符之间的交叉熵损失。使用 optim.Adam 优化器进行模型参数更新，学习率设为 0.001。

训练过程：设置训练轮数为 50 轮。在每轮训练中，遍历数据加载器，获取源序列和目标序列，计算模型输出，计算损失，并进行反向传播和参数更新。在每轮结束时，记录并打印平均损失，并保存模型状态和优化器状态。最后在每轮训练结束后，将模型和优化器的状态保存到文件，以便后续继续训练或生成文本。

文本生成

使用训练好的模型生成文本，首先加载处理好的文本数据集，设置序列长度。然后初始化模型参数，实例化 TransformerModel 模型，加载之前保存的模型状态。最后设置起始字符串和生成长度。调用 generate_text 函数，生成指定长度的文本，输出生成文本。结果如下所示：

输入文本：他跨下的枣红马奔驰了数十里地，早已筋疲力尽，在主人没命价的鞭打催踢之下，逼得气也喘不过来了，这时嘴边已全是白沫，猛地里前腿一软，跪倒在地。那汉子用力一提缰绳

输出文本：他跨下的枣红马奔驰了数十里地，早已筋疲力尽，在主人没命价的鞭打催踢之下，逼得气也喘不过来了，这时嘴边已全是白沫，猛地里前腿一软，跪倒在地。那汉子用力一提缰绳乱舞的火焰风中一声鹰啸穿透云霄鹤从湖面掠过花四溅手中长剑寒光闪烁，剑影

如同秋水横流脚下枯叶纷飞卷起了树梢上的蜘蛛丝轻轻颤动萤火虫忽明忽暗乱舞的火焰风中一声鹰啸穿透云霄鹤从湖面掠过花四溅手中长剑寒光闪烁剑影如同秋水横流脚下枯叶纷飞卷起了树梢上的蜘蛛丝轻轻颤动萤火虫忽明忽暗乱舞的火焰风中一声鹰啸穿透云霄鹤从湖面掠过花四溅手中长剑寒光闪烁剑影如同秋水横流脚下枯叶纷飞卷起了树梢上的蜘蛛丝轻轻颤动萤火虫忽明忽暗乱舞的火焰风中一声鹰啸穿透云霄鹤从湖面掠过花四溅手中长剑寒光闪烁，剑影如同秋水横流，脚下枯叶纷飞卷起了树梢上的蜘蛛丝轻轻颤动萤火虫忽明忽暗乱舞的火焰风中一声鹰啸穿透云霄鹤从湖面掠过花四溅手中长剑寒光闪烁剑影如同秋水横流脚下枯叶纷飞卷起了树梢上的蜘蛛丝轻轻颤动萤火虫忽明忽暗乱舞的火焰风中一声鹰啸穿透云霄鹤从湖面掠过花四溅手中长剑寒光闪烁剑影如同秋水横流脚下枯叶纷飞卷起了树梢上的蜘蛛丝轻轻颤动萤火虫忽明忽暗乱舞的火焰风中一声鹰啸穿透云霄鹤

Conclusions

由文本生成结果可知，忽略标点符号的因素，Transformer 模型的生成效果略优于 Seq2Seq 模型，主要是 Seq2Seq 模型架构更为简单，适合小规模任务和短序列处理，在处理长序列和需要高效并行化时存在限制。而 Transformer 模型更适合大规模数据和长序列处理，具有更强的并行化能力和长距离依赖捕捉能力，但需要更多的计算资源。

References

[1] https://blog.csdn.net/qq_27590277/article/details/124138834
[2] https://blog.csdn.net/weixin_42663984/article/details/117068473