使用 LSTM 和 Transformer 模型生成金庸武侠风格文本

贾云飞 ZY2303207

目录

摘要

本论文探讨了如何使用基于长短时记忆（LSTM）构建的传统 Seq2Seq 神经网络模型和基于 Transformer 的创新型神经网络模型基于金庸的武侠小说生成类似风格的文本。实验提出以金庸武侠作品为语料库的文本生成方法，证实 Seq2Seq 和 Transformer 模型在此类任务中的有效性以及不同。

1. 简介

随着近年人工智能和深度学习技术的发展，自然语言处理领域逐渐崛起。其中，文本生成是自然语言处理中的一个重要应用场景。本研究旨在利用长短时记忆（LSTM）神经网络模型，在金庸的武侠小说上实现风格类似的文本生成。

LSTM（长短时记忆网络）是一种应用广泛的循环神经网络（RNN）变体，

由 Hochreiter 和 Schmidhuber 于 1997 年提出。LSTM 通过引入门控机制解决了传统 RNN 面临的长序列训练中的梯度消失和梯度爆炸问题。它包含输入门、遗忘门和输出门三个门结构，以及一个细胞状态来实现对信息的筛选、存储和更新。LSTM 特别适用于处理具有长期依赖关系的序列数据，如自然语言处理、时间序列预测和语音识别等任务。由于其优秀的性能，LSTM 已被广泛应用于各种深度学习场景。

传统的 Seq2Seq（Sequence to Sequence）模型，也称为序列到序列模型，是一种常用于处理序列到序列任务的神经网络架构，尤其是在机器翻译、文本摘要、问答系统等领域。Seq2Seq 模型由两部分组成：编码器（Encoder）和解码器（Decoder）。LSTM 模型是实现传统 Seq2Seq 模型的一种方法。

Transformer 模型是一种基于自注意力机制的神经网络架构，由 Vaswani 等人在 2017 年的论文《Attention Is All You Need》中首次提出。它在自然语言处理（NLP）领域取得了巨大成功，特别是在机器翻译任务上。Transformer 模型完全基于注意力机制，摒弃了传统的循环神经网络（RNN）结构，从而能够并行处理序列数据，大大提高了训练效率。

传统 Seq2Seq: 使用循环神经网络（RNN）、长短期记忆网络（LSTM）或门控循环单元（GRU）作为编码器和解码器，这些模型依赖于循环机制逐步处理序列数据。编码器读取输入序列并生成上下文向量，解码器基于上下文向量和之前生成的输出逐步生成新的序列。

Transformer: 完全基于自注意力机制，不使用循环结构，可以并行处理序列中的所有元素。编码器处理输入序列，解码器同时考虑编码器的输出和之前生成的输出，并通过注意力机制关注编码器和解码器的历史输出。

两种模型都用于处理序列数据，尤其是在 NLP 任务中，如机器翻译、文本摘要等，都采用了编码器-解码器的框架，首先编码输入序列，然后解码生成输出序列，均通常使用嵌入层将输入的离散词符转换为连续的向量表示。同时两种模型的输出层通常是一个线性层，后接一个 softmax 函数，用于预测下一个词的概率分布。最后两种模型在训练时都可能使用交叉熵损失函数和类似的优化算法，如 Adam 或 SGD。

总的来说，Transformer 模型是对传统 Seq2Seq 模型的一种改进，它通过自注意力机制和并行处理能力解决了传统模型的一些限制，从而在许多序列到序列任务中取得了更好的性能。然而，传统 Seq2Seq 模型在某些特定场景下仍然有其应用价值，特别是在模型解释性和处理非常长的依赖关系时。

2. 实验方法

长短时记忆网络（Long Short-Term Memory，LSTM）是一种特殊的循环神经网络（RNN），用于解决长序列训练过程中的梯度消失和梯度爆炸问题。LSTM 结构具有三个门机制（输入门、遗忘门、输出门）和一个细胞状态来控制信息在时序中的流动。

输入门：负责确定从当前时间步的输入数据中接收多少信息。它包括两个部分：一个 Sigmoid 激活函数用于确定接收的信息量，另一个是用于调整输入数据的 tanh 激活函数。这两者相乘后得到最终输入。

遗忘门：决定了哪些信息从细胞状态中丢弃。它使用 Sigmoid 激活函数计算每个细胞状态元素丢弃的比例。

输出门：控制 LSTM 单元向下一个隐藏层传递多少信息。同样，输出门也

包括一个 Sigmoid 激活函数用于确定信息量，还有一个与细胞状态相乘的 tanh 函数来调整输出信息。

细胞状态：LSTM 的核心部分，负责存储长期记忆。输入门向细胞状态添加信息，遗忘门从细胞状态删除信息。通过这种方式，细胞状态可以保持长期依赖关系。

Transformer 模型是一种基于自注意力机制的神经网络架构，Transformer 模型完全基于注意力机制，摒弃了传统的循环神经网络（RNN）结构，从而能够并行处理序列数据，大大提高了训练效率。

自注意力机制（Self-Attention）：允许模型在编码或解码时，对序列中的每个元素计算注意力权重，从而捕捉序列内部的依赖关系。

多头注意力（Multi-Head Attention）：将自注意力机制应用于不同的表示子空间，然后在输出时将这些表示合并，以增强模型捕获信息的能力。

前馈网络（Feed-Forward Neural Network）：在每个注意力层之后，数据会通过一个前馈网络，进一步提取特征。

层归一化（Layer Normalization）：帮助稳定和加速深层网络的训练。

残差连接（Residual Connection）：允许模型训练更深的网络结构，通过在层之间添加跳跃连接来避免梯度消失问题。

我们采用了 LSTM 和 Transformer 神经网络作为文本生成的主要框架。首先，对金庸的 16 部武侠小说进行预处理，并将字符转换为整数表示，最后将输入序列送入 LSTM 和 Transformer 模型进行训练。

3. 实验过程：

3.1 数据预处理：

3.1.1 处理特殊字符

在之前的实验里，为了获得信息熵以及有效字符，我们进行了停词处理，但是在文本生成部分，我们不能简单的将停词去掉。因为失去停词、标点会导致生成文本逻辑的不连贯，所以我们选择只处理掉诸如‘/u3000’、制表符等特殊字符，保留标点以及无实意的停词。

3.1.2 分词

本文使用了 python 的 jieba 库来进行中文词汇的分词，该库的主要任务是将读取的字符串，按照数据库中的中文词汇，将中文字符串分成多个词组，便于后面词组的分类以及模型的信息学习。

3.1.3 文本数据处理

需要注意的是，如果我们选择将分词放到一个 list 里作为模型的输入，虽然从理论上是可行的，但是由于中文占据的空间较大，并且不同的分词可能具有不同数目的字数，为了保证输入数据的一致性，我们需要选择使用最大的分词，以此为一个空间存储每一个分词，大大增加了内存的负担。

为了解决上述问题，我们选择使用两个字典来进行关系的对应。具体的实现流程是：

1）遍历需要读取文本文件的文件夹，将需要读取的文本文件路径放入一个

list；
2）遍历文本路径 list 中的文本文件，依次读入文本文件，将读入的文本文件按行读取，每次读入一个字符串类型的变量；
3）将读入到的字符串变量处理特殊字符，再之后进行分词；
4）构造两个字典，第一个字典“word2idx”以依次得到的分词作为“键”，以序列数字作为“值”；另一个字典“idx2word”正好相反，以序列数字作为“键”，以依次得到的分词作为“值”，二者互相对应，形成映射关系；
5）构造一个 Tensor，遍历上述所有文本，以“word2idx”字典为参照，查找每个分词对应的“值”，即序列号，保存在 Tensor 中；
6）之后进行按照 batch_size 的矩阵重构，最后返回一个储存所有 int 类型数字序列的 Tensor 矩阵，该矩阵就是搭建的数据集。

3.2 模型建立：

构建了一个 LSTM 和 Transformer 单元的神经网络模型，并设置了适当的超参数。

3.2.1 模型搭建

应用 pytorch 中的 torch.nn 模块搭建 LSTM 模型和 Transformer 模型。

LSTM 模型：

- 通过 nn.Embedding 初始化一个词嵌入层，用来将映射的 one-hot 向量词向量化。输入的参数是映射表长度(即单词总数)和词嵌入空间的维数(即每个单词的特征数)。
- nn.LSTM 初始化一个 LSTM 层，是整个模型最核心的隐藏层。输入的参数是词嵌入空间的维数(即每个单词的特征数)、隐藏层的节点数和隐藏层的数量。
- 通过 nn.Linear 初始化一个全连接层，用来把神经网络的运算结果转化为单词的概率分布。输入的参数是 LSTM 隐藏层的节点数和所有单词的数量。

Transformer 模型：

- 初始化词嵌入层：使用 nn.Embedding 模块初始化一个词嵌入层，将输入的 one-hot 编码向量转换为连续的词向量。需要的参数是词汇表的大小（vocab_size）和每个词的特征数（embed_size）。
- 生成位置编码：Transformer 模型需要位置编码来提供序列中单词的位置信息。可以使用正弦和余弦函数生成位置编码。
- 构建 Transformer 编码器层：使用 nn.TransformerEncoderLayer 定义单个 Transformer 编码器层，然后使用 nn.TransformerEncoder 将多个这样的层堆叠起来形成完整的编码器。
- 定义前馈网络：使用 nn.Linear 定义两个线性层，一个用于 Transformer 层之后的前馈网络，另一个用于模型的输出层。
- 设置输出层：使用 nn.Linear 定义输出层，将前馈网络的输出转换为词汇表大小的概率分布。
- 添加 Dropout 层：使用 nn.Dropout 添加 Dropout 层，以减少模型的过拟

合风险。

定义模型的前向传播逻辑，传入的参数是输入值矩阵 x 和上一次运算得到的参数矩阵 h：

- 用 embed 把输入的 x 词嵌入化，即获取每个分词的特征；
- 用词嵌入化的 x 和上一次传递进来的参数矩阵 h，对得到的 LSTM 或 Transformer 模型进行依次迭代运算，得到输出结果 out 以及参数矩阵 h 和 c；
- 将 out 变形(重构)为合适的矩阵形状；
- 用 linear 把 out 转为和单词一一对应的概率分布。

3.2.2 参数设置

LSTM 模型：

- embed_size: 词嵌入后的特征数；
- hidden_size: LSTM 中隐层的节点数；
- num_layers: LSTM 中的隐层数量；
- num_epochs: 全文本遍历的次数；
- batch_size: 全样本被拆分的 batch 组数量；
- seq_length: 获取的序列长度；
- learning_rate: 模型的学习率；
- device: 设置运算用的设备实例；

Transformer 模型：

- embed_size: 词嵌入后的特征数。这是模型将每个单词或标记转换为固定大小的向量表示时使用的维度。在这个例子中，embed_size 是 512，意味着每个词或标记将被表示为一个 512 维的向量。
- vocab_size: 词总数。这是模型词汇表中不同单词或标记的数量。这个参数通常用于初始化词嵌入矩阵的大小。
- num_heads: 多头注意力机制中的头数。在 Transformer 模型中，多头注意力允许模型同时关注序列的不同部分，每个头学习不同的表示。
- num_layers: Transformer 中的层数。这里的 num_layers 是 6，意味着模型将有 6 个连续的 Transformer 层，每个层都会进一步处理和转换输入数据。
- num_epochs: 全文本遍历的次数。这是训练过程中数据集被完整遍历的次数。在这个例子中，num_epochs 是 10，意味着数据集将被完整遍历 10 次。
- hidden_dim:表示每个 Transformer 层中的隐藏状态的维度。
- dropout: 这是一个正则化技术，用于防止模型过拟合。在这个例子中，dropout 是 0.1，意味着在训练过程中，有 10%的神经元在每次迭代中将被随机丢弃。

3.3 模型训练：

采用交叉熵损失函数和 Adam 优化器，对模型进行训练。

3.3.1 模型输入

模型输入为一个 tensor，具体来说为 3.1.3 中所述的 Tensor 矩阵的前 seq_length 部分。

同时输入的参数矩阵为生成的全零 tensor 矩阵。

3.3.2 模型运行流程

1) states 是参数矩阵的初始化，相当于对 LSTMmodel 或 TFmodel 类里的(h, c)的初始化；
2) 在迭代器上包裹 tqdm，打印该循环的进度条；
3) inputs 和 targets 是训练集的 x 和 y 值；
4) 通过 detach 方法，定义参数的终点位置；
5) 把 inputs 和 states 传入 model，得到通过模型计算出来的 outputs 和更新后的 states；
6) 把预测值 outputs 和实际值 targets 传入 cost 损失函数，计算差值；
7) 由于参数在反馈时，梯度默认是不断积累的，所以在这里需要通过 zero_grad 方法，把梯度清零以下；
8) 对 loss 进行反向传播运算；
9) 为了避免梯度爆炸的问题，用 clip_grad_norm_设定参数阈值为 0.5；
10) 用优化器 optimizer 进行优化。

3.4 文本生成：

利用训练好的模型，在给定初始条件下生成文本。

定义 num_samples 为生成文本的分词个数。

3.4.1 给定文本开头

自己提供一个字符串作为给定的文本输入。首先要保证的是，给定的开头里所有的分词必须在我们获取的字典“word2idx”存在对应的键，不然会出现检索错误。

1) 之后调用模型，将获得的结果指数化，加强高概率结果的权重；
2) 在获得的结果中进行加权抽样，此时抽样的结果为一个长度为给定文本输入长度的 tensor，这是因为生成的结果是每个分词之后衔接的概率分布结果，所以我们需要获得该 tensor 最后一个结果；
3) 删除上次输入的第一个值，补充生成的 tensor 的最后一个结果在输入的最后，这样构成新的输入，循环即可获得结果；
4) 并将抽样的结果输入字典“idx2word”，找到对应的分词，将分词输出到生成文本中。

4. 实验结果

需要注意的是，实验过程中受到硬件设备的影响，一些超参数的调整范围有限，得到的结果可能不会太理想。（为了便于显示，生成文本的结果我们只节选部分进行显示）

4.1 以《倚天屠龙记》作为语料库训练

训练参数如下图所示：

```
embed_size = 256#增加每个词涵盖的特征数，提高结果精准度
hidden_size = 1024#增加神经元数量
num_layers = 3#增加隐藏层
num_epochs = 10#增加训练次数
batch_size = 50
seq_length = 30  # 序列长度，我认为是与前多少个词具有相关程度
learning_rate = 0.001
```

图 1　LSTM 模型训练参数图

```
ids = corpus.get_data(batch_size)  # 获得数据
vocab_size = len(corpus.dictionary)  # 词总数
num_heads = 8
num_layers = 6# num_layers
embed_size = 512#增加神经元数量
num_epochs = 10#增加训练次数
hidden_dim = 512
dropout = 0.1
learning_rate = 0.001
```

图 2　Transformer 模型训练参数图

4.1.1 Seq2Seq 给定文本开头

霎时之间，但见两人拳来脚往，斗得极是紧凑，可是始终相隔丈许之地枪道又，蛛儿说不得了下去你们暗器赴说来，都笑吟吟在怀里，谁是阳顶天牛棚，年纪市井，因而抢救，都是帐子。张无忌道：过这里，我再有些朋友，我想着这对甚么人，我适才半天有昆仑的则说，不知想还有喜事，形迹师妹、使之下毒而已？尼二字之后，兄弟听我的罢！”谢逊道：“莫非大师快对付我，虽然若是武我太师父，若不说我是不免事，你的小命力？”

4.1.2 Transformer 给定文本开头

霎时之间，但见两人拳来脚往，斗得极是紧凑，可是始终相隔丈许之地也不久。白衣胡思乱想派走了对手，但红脸一干迅速。张无忌老成持重十余丈，若仍然汉子恼怒一分，他以父亲道德雷火弹过招所孙三毁，他也峨嵋派沙坟，直打，再试他们命丧西华子，因此未必下毒，心中铁链力尽饮食便胆敢破例，一涌得到少女，日日十分挪移穴，方始无异，岂知另一人文字。徐二人虽欲退下，恐怕只须扣数十膺，在外要当明教后。

4.2 实验结果分析

可以看出生成的结果能很好地展现武侠小说的风格，并且可以看出给定文本生成的结果更加有逻辑并且更加流畅，但是生成文本在逻辑上仍有缺陷。

由于单个文本和 16 本小说语料库大小的不同，出于硬件限制，所以我们没

有进行对比实验，以上两个不同的模型选择使用了不同的参数。

Transformer 模型和 LSTM 方法生成的传统 Seq2Seq 模型在中文文本生成任务中各有优缺点。以下是两种模型在中文文本生成方面的一些优缺点分析：

对于 Transformer 模型优点有:

1. 并行计算: Transformer 模型利用自注意力机制，可以并行处理序列中的所有元素，大大提高了训练和推理速度。

2.长距离依赖捕捉: 自注意力机制能够捕捉长距离依赖关系，对于中文文本中的语义关联和长句结构特别有效。

3.灵活性和通用性: Transformer 模型结构简洁，易于调整和扩展，适用于多种 NLP 任务。

同时 Transformer 模型在处理长序列时，也具有计算复杂度和内存消耗较大、超参数调整和优化过程较为复杂、长文本生成时，面临内存限制和效率较低等问题。

LSTM 方法生成的传统 Seq2Seq 模型的优点有:

1.处理长序列能力: LSTM 具有很好的处理长序列数据的能力，通过门控机制解决了梯度消失问题。

2.序列建模: 能够捕捉序列数据中的动态特征和时间依赖性，适用于文本生成等序列到序列任务。

3.稳定性: 在许多任务中，LSTM 表现出较好的稳定性和可解释性。

同时 LSTM 模型生成的 Seq2Seq 也具有训练速度、难以捕捉非常长的依赖关系、数据不平衡等问题。

结论

根据实验结果显示，可以看出 LSTM 和 Transformer 模型的文本生成效果都较好，经过更加复杂的网络训练，可以很好的学习到文本的风格以及之间的逻辑结构。但 LSTM 计算量较大，需要较长的时间来学习，训练和推理速度相对较慢，Transformer 模型速度快一点。

两种模型各有千秋，选择哪一种取决于具体的应用场景、数据特性以及计算资源。Transformer 模型在处理大规模数据集和长文本方面具有优势，而 LSTM 模型在小数据集或对解释性要求较高的任务中可能更为合适。在实际应用中，研究者可能会根据具体需求和条件，选择或结合使用这两种模型。