

目录

摘要..... 1

1. 简介 1

2. 实验方法 2

3. 实验过程： 2

 3.1 数据预处理： 2

 3.1.1 处理特殊字符..... 2

 3.1.2 分词..... 2

 3.1.3 文本数据处理..... 2

 3.2 模型建立： 3

 3.2.1 模型搭建..... 3

 3.2.2 参数设置..... 3

 3.3 模型训练： 4

 3.3.1 模型输入..... 4

 3.3.2 模型运行流程..... 4

 3.4 词向量验证： 4

 3.4.1 词向量之间的语意距离..... 4

 3.4.2 词语的聚类..... 4

4. 实验结果 5

 以《倚天屠龙记》作为语料库训练..... 5

 4.1 词向量之间的语意距离..... 5

 4.2 对词向量进行聚类..... 5

 4.3 实验结果分析..... 6

结论..... 6

摘要

本论文探讨了如何使用长短时记忆（LSTM）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 LSTM 在此类任务中的有效性。

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。

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。

我们采用了 LSTM 神经网络作为文本生成的主要框架。首先, 对所选金庸的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 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 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。

- 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。

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 LSTM 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

- 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;

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 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 词向量验证:

利用训练好的模型, 在给定初始条件下生成文本。

通过计算词向量之间的语意距离和某一类词语的聚类来验证词向量的有效性。

3.4.1 词向量之间的语意距离

可以使用余弦相似度来衡量两个词向量之间的语义距离, 在自然语言处理中, 词向量通过诸如 Word2Vec、GloVe 或 LSTM 模型等方法学习得到, 余弦相似度通常用来衡量两个词在语义上是否相似。例如, 如果我们计算“国王”和“女王”的词向量的余弦相似度, 我们期望得到一个接近 1 的值, 因为这两个词在语义上是相似的。相反, 如果我们计算“国王”和“椅子”的词向量的余弦相似度, 我们期望得到一个接近 0 或更小的值, 因为这两个词在语义上不太相关。

3.4.2 词语的聚类

使用词向量进行聚类是一种无监督学习方法, 目的是将语义上相似的词汇聚集在一起。K-means 算法是实现这种聚类任务的一种常用技术。以下是 K-means

算法和使用词向量进行聚类的概念解析：

K-means 是一种流行的聚类算法，用于将数据点划分为 K 个簇。它的目标是使得每个簇内的数据点尽可能相似（紧凑），而不同簇之间的数据点尽可能不同（分离）。K-means 算法的基本步骤如下：

1. 选择 K 个初始中心点：随机选择 K 个数据点作为初始的簇中心，或者使用 K-means++ 算法来初始化中心点以提高聚类效果。
2. 分配步骤：将每个数据点分配给最近的簇中心，形成 K 个簇。
3. 更新步骤：重新计算每个簇的中心点，通常是簇内所有点的均值。
4. 迭代优化：重复分配步骤和更新步骤，直到簇中心不再显著变化，或者达到预设的迭代次数。

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）

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，我认为与前面多少个词具有相关程度
learning_rate = 0.001
```

图 1 训练参数图

4.1 词向量之间的语义距离

表 1 验证词向量的有效性计算词向量之间的语义距离

词	词	语义距离
剑	刀	0.0548
掌门	帮主	0.156
轻功	身法	0.091
内力	真气	-0.044
招式	手法	0.056
门派	宗派	0.026

4.2 对词向量进行聚类

取 1000 个词向量进行 K 近邻聚类得到结果如下图，分类个数为 5：

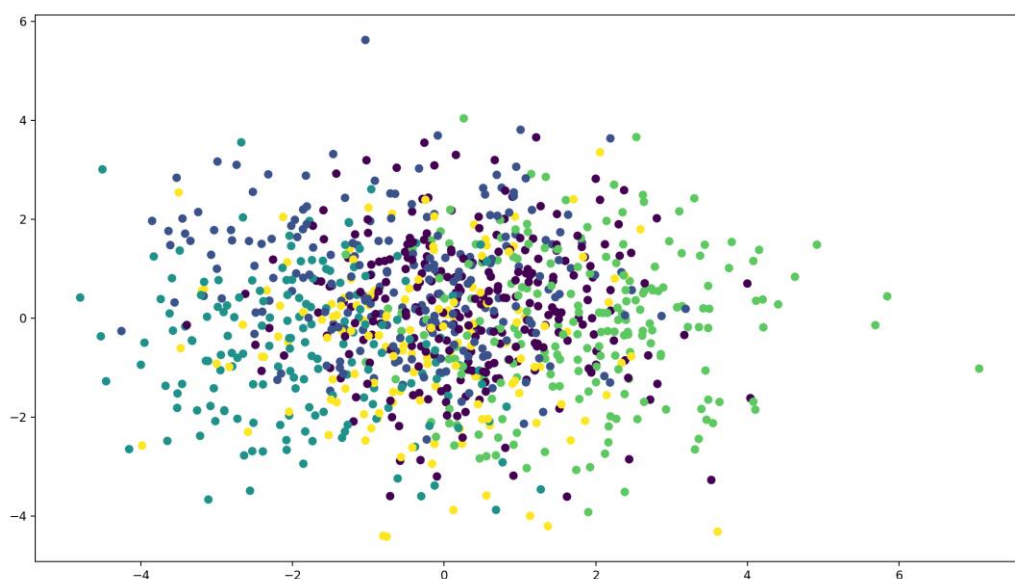


图 2 聚类图

4.3 实验结果分析

可以看出词向量之间的语意距离和某一类词语的聚类能够验证词向量的有效性。但是在得到结果上仍有较大缺陷。可能与训练 LSTM 模型的参数有关，改变相关参数对实验结果影响比较大。

增加词嵌入的特征表示 `embed_size`，使每个单词能包含更多信息；提高 LSTM 神经元数量 `hidden_size` 或隐藏层数 `num_layers`，以起到优化模型逻辑的作用；增加训练次数，如增加 `num_epochs`，使模型继续向最优解收敛。这三种方法都可以提高模型的精确性。

结论

根据实验结果显示，可以看出 LSTM 模型的词向量生成效果还有待验证，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。但其计算量较大，需要较长的时间来学习，训练和推理速度相对较慢。

总之，长短时记忆网络是一种强大的循环神经网络，通过使用门机制和细胞状态来处理长序列中的梯度问题。尽管存在一定的计算挑战，但 LSTM 仍然是许多序列学习任务的首选模型。