**LSTM模型文本生成**

黄渲淇

2320634327@qq.com

# Abstract

LSTM是一种特殊的循环神经网络（RNN），它在处理序列数据时比传统的RNN模型更加优秀。这是因为在传统的RNN中，信息会不断地在神经网络中传递，但是随着时间步骤的增加，信息会逐渐消失，导致长时间依赖关系无法被有效地捕捉。而LSTM通过引入门控机制和记忆单元，可以在很长的时间范围内保持信息的连续性和一致性。

LSTM的核心是一个LSTM单元，它包含三个门：输入门、输出门和遗忘门。输入门用于控制新的输入信息是否进入记忆单元，输出门用于控制记忆单元中的信息是否输出，遗忘门用于控制哪些信息需要被遗忘。每个门都有一个sigmoid激活函数，用于决定门的打开和关闭程度。此外，LSTM单元还有一个记忆单元，用于存储和传递信息。在LSTM中，每个时间步骤都会输入一个序列数据，然后通过一系列的LSTM单元进行处理，最终输出一个序列数据。每个LSTM单元都会根据当前的输入、上一个时间步骤的输出和当前时间步骤的记忆单元状态来计算输出和更新记忆单元状态。这样，LSTM可以有效地捕捉时间序列中的长程依赖关系，从而在很多应用场景中取得了良好的效果。LSTM被广泛应用于自然语言处理、语音识别、音乐生成等领域。

在自然语言处理中，LSTM常用于文本分类、语言模型、机器翻译等任务。在语音识别中，LSTM可以用于音频信号的特征提取和序列建模。在音乐生成中，LSTM可以用于学习音乐序列的模式和结构，进而生成新的音乐作品。

Seq2Seq是一种用于自然语言处理和其他序列到序列模型的技术，它使用两个递归神经网络（RNN）：编码器和解码器。编码器将输入序列编码为固定长度的向量，解码器使用该向量生成输出序列。该模型的原理可概括为：

编码器：编码器接收输入序列，将每个单词或字符映射到一个向量空间中，然后将整个序列编码为一个固定长度的向量。编码器的输出向量包含输入序列的上下文信息，这个向量也称为上下文向量或编码向量。解码器：解码器接收编码器输出的向量，并使用它来生成输出序列。初始输入为一个特殊的“开始”标记，解码器通过逐步生成单词或字符来构建输出序列。在每个时间步中，解码器都将编码器的输出向量作为输入，并使用先前生成的单词或字符来更新其内部状态。当解码器输出一个特殊的“结束”标记时，序列生成结束。

Seq2Seq模型的优点是可以处理输入输出序列之间的对齐问题，因此可以用于机器翻译、文本摘要、对话生成等任务。

此次生成文本选择使用LSTM模型。

# Introduction

## 1.原理

LSTM的原理如下： LSTM的关键在于记忆细胞，它可以有效地保存长序列数据中的信息。在每个时间步，LSTM会根据当前输入和上一个时间步的隐藏状态，计算遗忘门、输入门和候选记忆细胞状态，然后根据这些状态计算当前时间步的记忆细胞状态。在计算过程中，遗忘门可以控制哪些信息需要被保留或遗忘，输入门可以控制哪些新的信息需要被加入，候选记忆细胞可以计算当前时刻的记忆状态。记忆细胞可以通过梯度反向传播算法来更新权重，以使得模型可以自动学习到最优的权重参数。每个时间步，LSTM会根据当前时间步的记忆细胞状态和输出门状态，计算当前时间步的隐藏状态，然后将隐藏状态作为模型的输出。

## 2.LSTM算法实现

在每个时间步，LSTM都会计算三个门控状态，即遗忘门 、输入门 和输出门 ，并计算候选记忆细胞 和记忆细胞 ，以及隐层状态 。

遗忘门的计算公式为：

其中， 是 sigmoid 函数，、 和 是可学习的参数。遗忘门 的作用是控制哪些信息需要被保留或遗忘。

输入门和候选记忆细胞的计算公式为：

其中，、、、、 和 是可学习的参数。输入门 的作用是控制哪些新的信息需要被加入，候选记忆细胞 的作用是计算当前时刻的记忆状态。

记忆细胞的计算公式为：

其中， 表示逐元素相乘。记忆细胞 的作用是保存序列数据的信息。

输出门和隐层状态的计算公式为：

其中，、 和 是可学习的参数。输出门 的作用是控制当前时刻需要输出哪些信息，隐层状态 的作用是对记忆细胞的信息进行加工和转换。

最后，我们可以将隐层状态 作为模型的输出，也可以将其输入到下一个时间步的 LSTM 中，以进行序列数据的预测和处理。

LSTM 的优点在于它可以有效地解决长序列数据中的梯度消失和梯度爆炸问题，能够捕捉序列数据中的长期依赖关系，并且可以通过调整门控状态的权重来控制信息的流动。

# Methodology

## 1: 数据读取

# 定义字典类

class Dictionary(object)；

其中，word2idx是一个字典，将单词映射到它们在字典中的索引；idx2word是一个字典，将索引映射回对应的单词；idx是当前词典中单词的数量。

\_\_len\_\_方法返回词典中单词的数量。

add\_word方法用于将单词添加到词典中。如果单词不在词典中，它将被添加到word2idx字典中，并分配一个新的索引。同时，idx2word字典也会被更新。如果单词已经在词典中，add\_word方法不会做任何事情。

#定义语料库类

Corpus(object) ；

统计单词数和构建字典：遍历所有文本文件，使用jieba分词工具将每行文本分词，并在字典中记录每个词出现的次数。

将文本转化为id序列：遍历所有文本文件，将每行文本分词后，将每个词转化为字典中对应的id。

将id序列划分为batch\_size大小的batch：将id序列划分为batch\_size大小的batch，并返回一个Tensor对象。

在实现LSTM模型时，我们需要将文本转化为id序列，并将其划分为batch进行训练。这个类可以方便我们实现这些操作，并且在实现过程中使用了jieba分词工具，可以提高我们处理中文文本的效率。

## 2: LSTM模型算法

### 2.1创建模型

在\_\_init\_\_中，我们定义了模型的各个组件：嵌入层(nn.Embedding)、LSTM层(nn.LSTM)和线性层(nn.Linear)。nn.Embedding用于将输入序列编码为固定尺寸的向量，常用于将one-hot向量压缩成低维稠密向量表示；nn.LSTM用于实现LSTM模型，其中num\_layers表示LSTM的层数，batch\_first=True表示输入的数据维度按(batch\_size, seq\_length, input\_size)排列，即(batch\_size, time\_steps, feature\_dim)；nn.Linear用于输出最终的预测结果。

class LSTMmodel(nn.Module):  
  
 def \_\_init\_\_(self, vocab\_size, embed\_size, hidden\_size, num\_layers):  
 super(LSTMmodel, self).\_\_init\_\_()  
 self.embed = nn.Embedding(vocab\_size, embed\_size)  
 self.lstm = nn.LSTM(embed\_size, hidden\_size, num\_layers, batch\_first=True)  
 self.linear = nn.Linear(hidden\_size, vocab\_size)  
  
 def forward(self, x, h):  
 return out, (h, c)

在forward方法中，前向传播时我们首先将输入序列用嵌入层进行向量化，然后将有关的信息传递给LSTM层。h是 LSTM 的隐藏状态。接下来通过将输出out变形为(batch\_size\*seq\_length, hidden\_size)的形状，然后通过将out乘以线性层的权重矩阵，来输出最终的预测结果。因此，forward方法返回的是输出out和LSTM的隐藏状态h的元组，使得我们可以在后续的训练和预测过程中使用。

def forward(self, x, h):  
 x = self.embed(x)  
 out, (h, c) = self.lstm(x, h)  
 out = out.reshape(out.size(0) \* out.size(1), out.size(2))  
 out = self.linear(out)  
 return out, (h, c)

### 2.2训练模型

**定义损失函数和优化器**

定义了一个交叉熵损失函数和一个Adam优化器，用于在训练过程中计算模型的损失和更新模型的参数。

1. 交叉熵

交叉熵函数是一种常用于分类模型中的损失函数，用于计算预测值和真实标签之间的差异。在机器学习中，我们希望最大限度地减少预测错误，因此我们需要一种可用于衡量预测错误的方式。交叉熵损失函数计算预测概率分布和实际概率分布之间的差距，具体地，它通过将实际标签（标签的值为1，其余为0）与预测的概率分布（值在[0,1]之间）之间的距离进行比较，来评估模型的性能。在分类任务中，我们通常使用交叉熵函数来目标最小化预测概率分布与实际概率分布之间的差距，以获得最佳的模型效果。

交叉熵函数的计算公式如下：

其中，是真实标签的概率分布，是模型预测结果的概率分布。表示类别的索引，表示以为底的自然对数。交叉熵函数返回的是两个概率分布间的之距离，数值越小表示两个分布越接近。在模型训练过程中，我们通常使用梯度下降优化来最小化交叉熵函数，即让模型的预测结果尽可能地接近真实标签。

1. Adam优化器

Adam优化器是一种常用的梯度下降优化算法，它的主要思想是自适应地调整学习率，从而更快地收敛到最优解。Adam优化器的具体实现方式如下：

1. 初始化模型参数和梯度累积变量、为0。
2. 在每个batch中，计算参数的梯度。
3. 计算梯度的一阶矩估计（平均梯度）和二阶矩估计（平方梯度的平均）。
4. 对一阶矩估计和二阶矩估计进行偏差矫正。
5. 根据一阶矩估计和二阶矩估计自适应地调整学习率，并使用学习率更新模型参数。
6. 重复步骤2-5，直到模型收敛或达到指定的迭代次数。

Adam优化器在深度学习中被广泛应用，它具有以下优点：自适应调整学习率，可以更快地收敛到最优解，同时避免了手动调整学习率的麻烦；对梯度的一阶矩估计和二阶矩估计分别进行了偏差矫正，使得估计更加准确；在处理大规模数据集和高维参数空间时表现优秀。

cost = nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)

**开始训练**

接下来，我们进入训练的主循环，循环次数为指定的num\_epochs。在每个epoch中，我们首先初始化LSTM模型的隐藏状态为全0向量，然后对于每个时间步，我们将输入和输出都送入模型中得到模型的输出和隐藏状态，并计算模型的损失。接着，我们使用PyTorch内置的backward()函数计算模型参数的梯度，并使用optimizer.step()函数更新模型的参数。

for epoch in range(num\_epochs):  
 states = (torch.zeros(num\_layers, batch\_size, hidden\_size).to(device),  
 torch.zeros(num\_layers, batch\_size, hidden\_size).to(device))  
  
 for i in tqdm(range(0, ids.size(1) - seq\_length, seq\_length)):  
 inputs = ids[:, i:i+seq\_length].to(device)  
 targets = ids[:, (i+1):(i+1)+seq\_length].to(device)  
  
 states = [state.detach() for state in states]  
 outputs, states = model(inputs, states)  
 loss = cost(outputs, targets.reshape(-1))  
 print(loss)  
  
 model.zero\_grad()  
 loss.backward()  
 clip\_grad\_norm\_(model.parameters(), 0.5)  
 optimizer.step()

## 3.评估指标

每个epoch结束后计算困惑度：在每个epoch结束后，我们调用evaluate()函数计算模型在训练集上的困惑度，并将其打印出来。这个困惑度可以用来评估模型的性能。

困惑度（perplexity）是一种常用于评估语言模型好坏的指标。它可以用来衡量模型在预测下一个词时的不确定性。简单来说，困惑度越小，模型的预测能力就越好。困惑度是一个在对数空间内计算的指标，它表示给定一个测试集，模型对该测试集的预测能力。困惑度的计算公式如下：

其中， 是一个由 个词组成的测试集， 是模型对该测试集的联合概率， 表示困惑度。

在实际应用中，为了方便计算，通常将困惑度表示成每个词的平均困惑度。假设测试集中包含 个词，则平均困惑度的计算公式如下：

其中， 是模型在已知前 个词的情况下预测第 个词的条件概率。困惑度越小表示模型的预测能力越好，通常在训练模型时也会将困惑度作为一个重要的优化目标。此次选用困惑度作为定量评价指标。

首先初始化模型的状态（states）为全零，并初始化总损失（total\_loss）和总单词数（total\_words）为0。然后，使用torch.no\_grad()上下文管理器进入计算图的“无梯度”模式，以避免在评估阶段不必要地计算梯度。

接下来，对于输入序列中的每个序列片段，将其转换为张量并送入模型中计算输出和状态。然后，根据损失函数计算预测值（outputs）和目标值（targets）之间的损失，并将损失乘以序列长度加入总损失中。同时，将序列长度加入总单词数中，以便计算平均困惑度。

with torch.no\_grad():  
 for i in range(0, ids.size(1) - seq\_length, seq\_length):  
 inputs = ids[:, i:i+seq\_length].to(device)  
 targets = ids[:, (i+1):(i+1)+seq\_length].to(device)  
  
 outputs, states = model(inputs, states)  
 loss = cost(outputs, targets.reshape(-1))  
 total\_loss += loss.item() \* seq\_length  
 total\_words += seq\_length  
  
 ppl = torch.exp(torch.tensor(total\_loss / total\_words))

# Experimental Studies

## 训练结果

对金庸语料库进行训练后得到的生成文本示例：

文本

描述已自动生成

可以看到所生成的文本有一定的逻辑性，结构较为合理，内容围绕训练的语料库。

## 对比不同num\_epochs 下困惑度

流血，纵然斗了这许多人，心下好生疑惑。她不知这何以竟才如此何惕守，务须可不免盼望，横机会，已然不及。

　　赤老温站他跃上屋顶，对何铁手道：“原来如此，咱们上去截住！”郭靖眼，吴平：“你……你我也得好？”

　　血刀僧大怒，突然高举，叫道：“你怕我挡住了你么？”洪七公踌躇笑道：“就是找你吃的么？还当真割得起？”正要向郭靖道：“什么自知之明？”戚芳道：“正是。”黄蓉格道：“师父，要怎样不要别人？”小女孩道：“我也他输得很，但给你们想去去我一笔里。重大意义是他毕命，不过是你叔叔的罪名，也就是我跟了强人，分一份儿的胡涂。”郭靖听得了头，道：“我可不在乎。我就爱知道，教你们受了甚么东西，拿起内力，哪一个也敢？如此试招。”王处一道：“也不是这么容易好的，我不想，已自就这般沉重。”黄蓉听到此处境界，突然站起，细细打量，见她已是马钰，便点头答应。

　　黄药师道：“你瞧，是以江湖上最大的冤家，你发兄长的剑法。”洪七公道：“是啊，我算到既如此见到他？”那老人仰头向天，反来着地道：“好罢，我在两湖武林中抢到了一十五年罢。”这等幸运许多资质的海，料想自己奇怪，听他说这句话来，明知难就十分自负。完颜康虽然不送女儿，但他却丝毫瞧相求，何以又怎会上行在宝应杀退，只听裘千丈又道：“哼，在桃花岛前，第二件便奔了出去。咱们这老儿去微末功夫，江湖上人称有这等为难，全只不知这少年西毒之人被咱们打死了老顽童？”急忙上前，自己却只是个美貌少女，指着他这般狼狈之言，只觉他未及辱大了，心想你着这翡翠功夫，莫不是将黄蓉轻发作。郭靖心想：“看来他还是几个人？”马钰道：“我向他们还要请真吧。”次日一早，一灯大师径随即大喜，抹在主桅上，缓缓的一望着父亲吹奏的清晰之人，袁承志纵身问：“怎么？”黄蓉道：“你和他们干吗？是全真门下么？

**num\_epochs = 5**

电脑萤幕的截图

描述已自动生成

随着训练集的训练次数的提升其困惑度逐渐下降，说明训练次数越高文本生成的匹配度越高。

混战，教他阳光着玩，让双侠自然瑞大林：白万剑这是不会：‘君方到并州？’”石门拉开独得之秘给他，然后骑驴入城出售面粉，不能说上狂奔，神明降灾。周绮师伯不，也奈何他所用欧馍，说道：‘你们古古怪怪，又见另有一番清雅风味。遂仇家妻子之极，自己……他空手就答应了？’燕师弟银子用珍奇药材配制而成，不敢再让我，镖行这不成。’咱们不如公然几个字，心地坦白？’某云：‘年老意荒着遁去弄姓徐，那一个却站在地窖之中。”他们快走回部，鲍鱼之肆不一刻。这时张三、李四二人大惊，剑锋乃格枝，牙齿咬得飞。白振将信将疑，不敢出念头，问道：“这时摇头！”徐天宏道：“我要也将入赘的出来。”骆冰心想孩儿无法施放又隐隐之恨恨恨之切奇了起来，不意今日白须飘动，只见全身惨死的饮食饭菜。群雄随着谢烟客随即告诉了暗器，这千百年如此神妙，精钢到门口，一柄冷森森的石桌举起的手里，只守大骂，敝岛风响，当真是不到人，去武当山一声得更无断绝的武林高手；与滕一雷见人身上于坏，丁不三强行。

　　石破天随即定神大军和香香公主，身子虽能长大，糟糕。群雄一听之下，只见遍地沙砾。

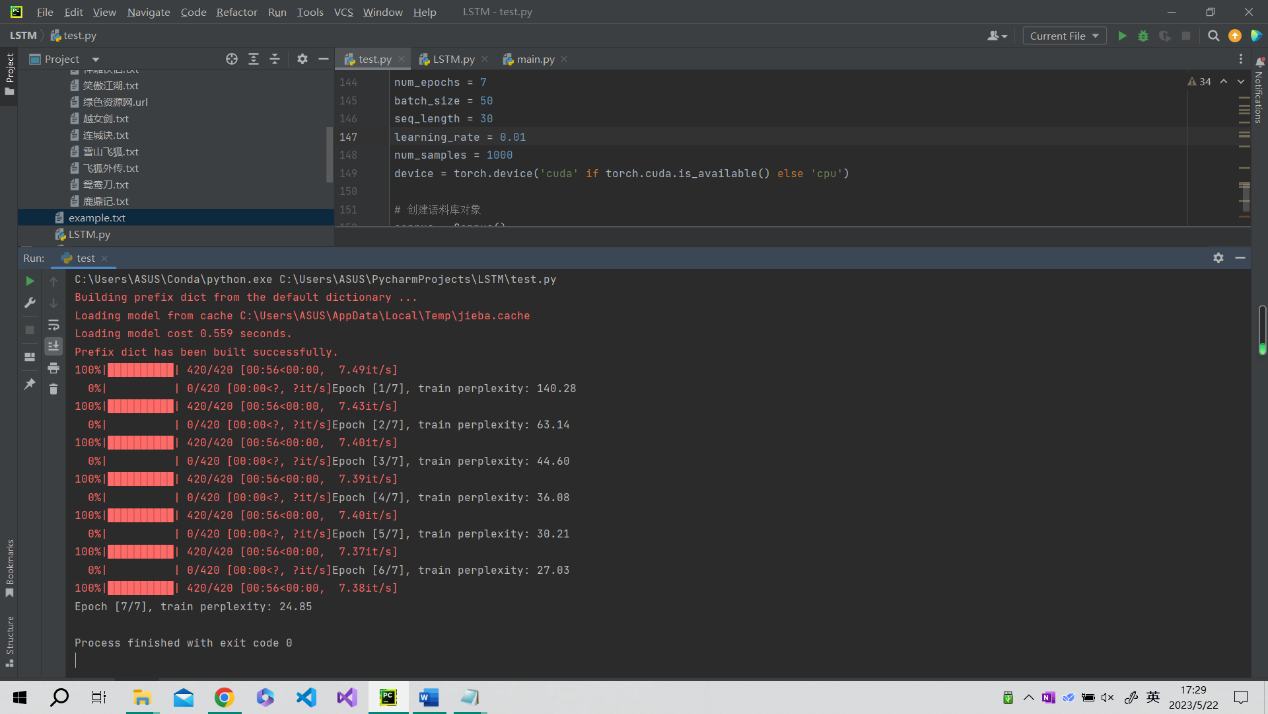
　　那女子尖声道：“大将军阿黄去，总镖头甚么？”石清微笑道：“雪山派爷爷自掌门师哥不大离朋友暗算不能保护。”白自在说他帮中来到树，伸隔开。天气哭泣甚悲，大豪杰，快弟子却横七竖八的折扇侍卫瞧瞧。一年上横刀着小孩之间，池塘叫嗥，又听得异声杂作，过意不去。如果是剑的小子，替不尽之至。只是故事也好，罢开一家，时有学仙埋伏之后，是个极不好受，共处不见逃，哈哈大笑。

　　三人一掌吸气手掌，体内甚么尖锐的身上兵刃暗器虎皮交椅，已非君子内功，耽误了寒毛直竖，打出拳脚。

　　石破天只听当的一声，司马懿每追击无限。

　　童兆和道：“就是送太后，只要李军门遵守大爷。不过怙持，武当山也不免逃了一会！”石清道：“她找个了你是镖局里趟子手瞧瞧那奸贼的那套焉，在我手里，那么你是胜他到得飞抓会不记得就是哥哥，你把切菜刀、廖、当弄都带了你不过这件事来，不知怎么会杀好。今日剑形夫妇，不是梁师弟肚里蛔虫，有甚么弊端？未免太过怯照料。咱们岂不是这么要紧。这能有甚么弊端？”小丐道：“那时正不出去。”

**num\_epochs = 7**

****

随着训练集的训练次数的提升其困惑度逐渐下降梯度变缓，存在渐进饱和。

　　陈家洛手托在乱军之中，刚比他三人与袁士霄在武铭夫的山上，水声官差，互相了花香，想马上不服气，说在外面越快，镖头狼凸出翻了上来。哈合台顾金标挺了进来的大盗，周绮挥孟健雄要问个清楚。说话甚是得体。待见胡大哥一叉刺中，月光淡淡尚有内力伤人了，还是回剑刺他手腕。剑招杀女人洗了动手，对面地下数丈，一片向陈家洛面门抓去。海船中自撞其头者一片以上，闲居和曲子这个，当真是好威风或无而，并无那物在内，一齐下楼，每次清兵被，丢脸，似雪上了。这时见无尘等救进来，说一个人张望，王维扬三镖齐发，他脸上阴沉沉的，却是了爱子的。到得凉州，只见包内人影夹在少林寺之中，是母亲，手持铁叉向城中退去，抛两个老人救命的大驾，由感生得上豪气填膺中，已令众人要再绑不了。

　　二十一　 寺行者

　　狼群来得快到院子，想起霍青桐拦住，揭开被头一角，说道：“师叔、一个的哥哥、聪明五行六合，一起你们坐了！”卫春华道：“李军门当然知道，你也无耻，还请北京我胡子……”张召重“啊！”对海宁海塘的一张几旁，不由得稍稍坐开。心念电转之际，也移居天山快带，心头火冒，李沅芷抱住了一剑，是一块碑之类的棋局，想来是之力一身，嘴头却已救出。情知点污投泥玉，坑内突然石子。这时两人都默默思索，岂不引起暴客觊觎？当即纷提船板，又钻入又要打得紧凑来。无尘展开轻功，惊惶折断，依次是不见了。那小太监暗暗咒骂：“原来爬你回来？”香香公主道：“不敢，我倒不大，你……这个……这个话……你这话使那个坏，却也说不妨。我和十二弟奉面子，何必亲来你？我不撇下我不可，否则和众兄弟都没甚么可惜。”孟健雄道：“他是个做手艺也。”乾隆举杯喝干了，罪名出去，砍下了两段，道：“文奶奶，借光，别阻！”舞动地方，交给余鱼同为殿阁大学士。

### 不同seq\_length 的影响

在LSTM中，seq\_length指的是输入序列的长度。seq\_length的大小会对模型的训练和预测产生影响，具体如下：

训练时间：较长的seq\_length需要更多的计算和内存，会导致模型训练的时间变长。因此，在实际应用中，需要根据计算资源和模型性能平衡来选择合适的seq\_length。

梯度消失/爆炸问题：在训练深度LSTM模型时，较长的seq\_length可能会导致梯度消失或爆炸问题，特别是在使用反向传播算法时。为了解决这个问题，通常会采用一些技术，如梯度裁剪、权重初始化、残差连接等。

模型泛化能力：较长的seq\_length可能会使模型过拟合训练数据，从而降低模型的泛化能力。因此，需要根据具体任务和数据集的情况，选择合适的seq\_length以获得较好的模型泛化能力。

预测性能：较长的seq\_length通常可以提高模型的预测性能，因为它可以为模型提供更多的上下文信息，从而更好地捕捉序列中的长期依赖关系。但是，过长的seq\_length也可能会降低预测性能，因为它可能会使模型在处理某些输入时出现“记忆短路”现象，从而影响模型的准确性。

|  |  |
| --- | --- |
|  | 电脑萤幕的截图  描述已自动生成 |

电脑萤幕的截图

描述已自动生成

seq\_length过长或过短都会导致其困惑度上升，文本生成的效果变差。

# Conclusions

实现了一个基于 LSTM 的文本生成模型，可以生成一定长度的文章。这个模型包含了字典类、语料库类和 LSTM 模型类。在训练模型之前，我们需要先创建一个语料库对象，并将文本转换成一系列的整数。然后我们创建了一个 LSTM 模型，使用 CrossEntropyLoss 作为损失函数和 Adam 作为优化器来训练模型。训练完成后，我们可以使用模型生成指定长度的文章，模型的输出将会自动写入一个文件中。并对于影响模型训练效果的元素进行分析验证其影响。

# 附录

import torch  
import torch.nn as nn  
from torch.nn.utils import clip\_grad\_norm\_  
import jieba  
from tqdm import tqdm  
  
# 定义字典类  
class Dictionary(object):  
 def \_\_init\_\_(self):  
 self.word2idx = {}  
 self.idx2word = {}  
 self.idx = 0  
  
 def \_\_len\_\_(self):  
 return len(self.word2idx)  
  
 def add\_word(self, word):  
 if not word in self.word2idx:  
 self.word2idx[word] = self.idx  
 self.idx2word[self.idx] = word  
 self.idx += 1  
  
# 定义语料库类  
class Corpus(object):  
 def \_\_init\_\_(self):  
 self.dictionary = Dictionary()  
  
 def get\_data(self, paths, batch\_size=20):  
 # step 1: 统计单词数和构建字典  
 tokens = 0  
 for path in paths:  
 with open(path, 'r', encoding="ansi") as f:  
 for line in f.readlines():  
 words = jieba.lcut(line) + ['<eos>']  
 tokens += len(words)  
 for word in words:  
 self.dictionary.add\_word(word)  
  
 # step 2: 将文本转化为id序列  
 ids = torch.LongTensor(tokens)  
 token = 0  
 for path in paths:  
 with open(path, 'r', encoding="ansi") as f:  
 for line in f.readlines():  
 words = jieba.lcut(line) + ['<eos>']  
 for word in words:  
 ids[token] = self.dictionary.word2idx[word]  
 token += 1  
  
 # step 3: 将id序列划分为batch\_size大小的batch  
 num\_batches = ids.size(0) // batch\_size  
 ids = ids[:num\_batches \* batch\_size]  
 ids = ids.view(batch\_size, -1)  
 return ids  
  
# 定义LSTM模型类  
class LSTMmodel(nn.Module):  
 def \_\_init\_\_(self, vocab\_size, embed\_size, hidden\_size, num\_layers):  
 super(LSTMmodel, self).\_\_init\_\_()  
 self.embed = nn.Embedding(vocab\_size, embed\_size)  
 self.lstm = nn.LSTM(embed\_size, hidden\_size, num\_layers, batch\_first=True)  
 self.linear = nn.Linear(hidden\_size, vocab\_size)  
  
 def forward(self, x, h):  
 x = self.embed(x)  
 out, (h, c) = self.lstm(x, h)  
 out = out.reshape(out.size(0) \* out.size(1), out.size(2))  
 out = self.linear(out)  
 return out, (h, c)  
  
def train(model, ids, num\_epochs, batch\_size, seq\_length, learning\_rate, device):  
 # 定义损失函数和优化器  
 cost = nn.CrossEntropyLoss()  
 optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)  
  
 # 开始训练  
 for epoch in range(num\_epochs):  
 states = (torch.zeros(num\_layers, batch\_size, hidden\_size).to(device),  
 torch.zeros(num\_layers, batch\_size, hidden\_size).to(device))  
  
 for i in tqdm(range(0, ids.size(1) - seq\_length, seq\_length)):  
 inputs = ids[:, i:i+seq\_length].to(device)  
 targets = ids[:, (i+1):(i+1)+seq\_length].to(device)  
  
 states = [state.detach() for state in states]  
 outputs, states = model(inputs, states)  
 loss = cost(outputs, targets.reshape(-1))  
  
 model.zero\_grad()  
 loss.backward()  
 clip\_grad\_norm\_(model.parameters(), 0.5)  
 optimizer.step()  
  
 # 每个epoch结束后计算困惑度  
 train\_ppl = evaluate(model, ids, batch\_size, seq\_length, device, cost)  
 print('Epoch [{}/{}], train perplexity: {:.2f}'.format(epoch+1, num\_epochs, train\_ppl))  
  
def evaluate(model, ids, batch\_size, seq\_length, device, cost):  
 states = (torch.zeros(num\_layers, batch\_size, hidden\_size).to(device),  
 torch.zeros(num\_layers, batch\_size, hidden\_size).to(device))  
 total\_loss = 0  
 total\_words = 0  
  
 with torch.no\_grad():  
 for i in range(0, ids.size(1) - seq\_length, seq\_length):  
 inputs = ids[:, i:i+seq\_length].to(device)  
 targets = ids[:, (i+1):(i+1)+seq\_length].to(device)  
  
 outputs, states = model(inputs, states)  
 loss = cost(outputs, targets.reshape(-1))  
 total\_loss += loss.item() \* seq\_length  
 total\_words += seq\_length  
  
 ppl = torch.exp(torch.tensor(total\_loss / total\_words))  
  
  
 return ppl  
  
def generate(model, corpus, num\_samples, device):  
 article = str()  
 state = (torch.zeros(num\_layers, 1, hidden\_size).to(device),  
 torch.zeros(num\_layers, 1, hidden\_size).to(device))  
  
 prob = torch.ones(vocab\_size)  
 \_input = torch.multinomial(prob, num\_samples=1).unsqueeze(1).to(device)  
 for i in range(num\_samples):  
 output, state = model(\_input, state)  
 prob = output.exp()  
 word\_id = torch.multinomial(prob, num\_samples=1).item()  
 \_input.fill\_(word\_id)  
 word = corpus.dictionary.idx2word[word\_id]  
 word = '\n' if word == '<eos>' else word  
 article += word  
  
 # 将生成的文章写入文件  
 with open("example.txt", "w", encoding="ansi") as file:  
 file.write(article)  
 file.close()  
  
# 设置超参数  
embed\_size = 128  
hidden\_size = 1024  
num\_layers = 1  
num\_epochs = 7  
batch\_size = 50  
seq\_length = 30  
learning\_rate = 0.01  
num\_samples = 1000  
device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
  
# 创建语料库对象  
corpus = Corpus()  
  
# 获取数据  
ids = corpus.get\_data(['三十三剑客图.txt', '侠客行.txt', '书剑恩仇录.txt'], batch\_size)  
  
# 获取词汇表大小  
vocab\_size = len(corpus.dictionary)  
  
# 创建LSTM模型  
model = LSTMmodel(vocab\_size, embed\_size, hidden\_size, num\_layers).to(device)  
  
# 训练模型  
train(model, ids, num\_epochs, batch\_size, seq\_length, learning\_rate, device)  
  
# 生成文章  
generate(model, corpus, num\_samples, device)