基于 LSTM 的金庸小说文本序列生成

——自然语言处理的深度学习报告(第四次)

崔潇雅 ZY2203502

1 摘要

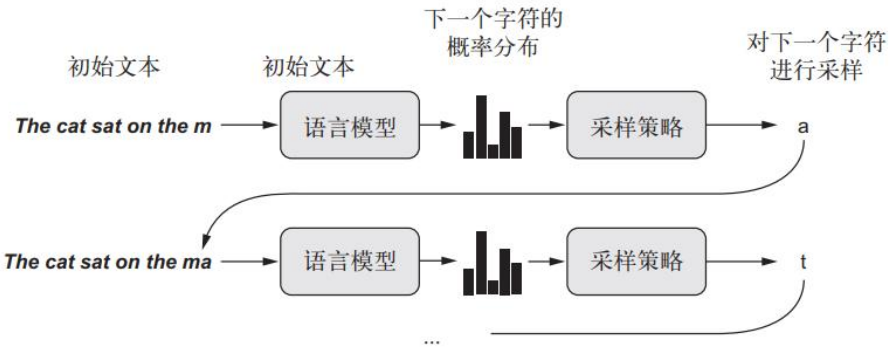
本实验通过 one-hot 编码方式对金庸小说《神雕侠侣》进行编码，用 Pytorch 实现了一个 LSTM 版的 Char RNN 模型，基于训练得到的模型随机选取初始种子生成指定字符数目的金庸风格小说。由于计算性能限制，本实验训练样本从《神雕侠侣》中截取一部分作为输入，经过数小时训练迭代后，模型已经能够学习到许多词语、标点符号，但文本可读性仍然较低，使用定量指标进行评价：loss 降到 0.287，n-gram 重复率升到 0.124。

2 理论基础

2.1 LSTM

LSTM(Long Short-Term Memory)是一种常用的循环神经网络(RNN)模型，用于处理序列数据。相比于普通的 RNN，LSTM 能够更好地处理长序列和梯度消失/爆炸的问题。

本实验中将会用到一个 LSTM 层，向其输入从文本语料中提取的 N 个字符组成的字符串，然后训练模型来生成第 N+1 个字符。模型的输出是对所有可能的字符做 softmax，得到下一个字符的概率分布。这个 LSTM 叫作字符级的神经语言模型(character-level neural language model)。如下图是使用语言模型逐个生成文本的过程。



3 实验过程

3.1 实验准备

读取文本文件中的内容，将读取的内容转换为字符串类型，输出前 100 个字符的内容。

```
12  path = r"E:\DLNL\homework4\神雕选段.txt"
13  with open(path, 'r', encoding='ANSI') as f:
14      data = f.readlines()
15  data=''.join(data)#读取的列表转换为字符串
16  print(data[:100])
```

生成唯一字符串列表 **chars**，将唯一字符列表转换为字符到索引的字典 char_to_ix 和索引到字符的字典 ix_to_char。

```
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print(f'data has {data_size} characters, {vocab_size} unique.')
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }
```

由于文本数据集的规模较大，所以使用稀疏矩阵（csr_matrix）来存储训练数据。将每个字符转换成其在唯一字符列表中的索引，然后将稀疏矩阵中对应的位置置 1，形成 one-hot 形式，最后将目标 y_train 向左移动一个字符位置。

```
X_train = csr_matrix((len(data), len(chars)), dtype=np.int)
char_id = np.array([chars.index(c) for c in data])
X_train[np.arange(len(data)), char_id] = 1
y_train = np.roll(char_id,-1)
X_train.shape
y_train.shape
```

定义一个生成器函数，用于划分训练数据批次，以便于在训练神经网络时进行批量梯度下降。每个批次的大小为 seq_length，可以通过调用该函数多次来获取训练数据的不同批次。

```
def get_batch(X_train, y_train, seq_length):
    X = X_train
    y = torch.from_numpy(y_train).long()
    for i in range(0, len(y), seq_length):
        id_stop = i+seq_length if i+seq_length < len(y) else len(y)
        yield([torch.from_numpy(X[i:id_stop].toarray().astype(np.float32)),
               y[i:id_stop]])
```

3.2 模型构建

从 pytorch 的 nn.Module 继承定义一个 LSTM 模型，该类包含三个参数：输入数据的特征维度、LSTM 模型的隐藏状态维度和输出维度。在前向计算过程中，输入序列进入 LSTM 层中，得到一个输出序列和最终隐藏状态。输出序列的最后一个时间的隐藏状态会被输入到全连接层中，得到最终的输出。在每个时间步长中，模型的隐藏状态和记忆单元状态会被传递到下一个时间中来记忆之前的信息。

损失函数为交叉熵损失（Cross Entropy Loss），优化器为 Adam 优化器，并设置学习率 lr 为 0.005。

```
class nn_LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, X, hidden):
        _, hidden = self.lstm(X, hidden)
        output = self.out(hidden[0])
        return output, hidden

    def initHidden(self):
        return (torch.zeros(1, 1, self.hidden_size),
                torch.zeros(1, 1, self.hidden_size)
               )

hidden_size = 256
seq_length = 25
rnn = nn_LSTM(vocab_size, hidden_size, vocab_size)
#设置损失函数和优化器
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(rnn.parameters(), lr=0.005)
```

3.3 模型训练

训练模型，对于每个 time step，模型的输入和隐藏状态都会被更新并计算损失。在计算完整个 batch 的损失后，使用反向传播算法计算梯度，并使用优化器更新模型的参数。最终返回该批次的输出结果和平均损失,最终使得模型的输出逐渐趋近于真实的文本数据。

```
def train(X_batch, y_batch):
    h_prev = rnn.initHidden()
    optimizer.zero_grad()
    batch_loss = torch.tensor(0, dtype=torch.float)

    for i in range(len(X_batch)):
        y_score, h_prev = rnn(X_batch[i].view(1,1,-1), h_prev)
        loss = loss_fn(y_score.view(1,-1), y_batch[i].view(1))
        batch_loss += loss
    batch_loss.backward()
    optimizer.step()

    return y_score, batch_loss/len(X_batch)
```

3.4 预测并输出

使用训练好的 rnn 模型定义一个函数用于生成文本，通过输入初始的种子字符，不断预测下一个字符，并将其作为下一个 time step 的输入，直到生成所需长度的文本。生成的文本的起始字符由 X_seed 决定，生成的长度为 length。生成每个字符时，模型会根据前面已经生成的字符和当前的隐藏状态，预测下一个字符的概率分布，并从中随机采样一个字符作为生成的结果。因此，本文每次生成的文本可能会略有不同。

```python
def sample_chars(rnn, X_seed, h_prev, length=20):
    '''Generate text using trained model'''
    X_next = X_seed
    results = []
    with torch.no_grad():
        for i in range(length):
            y_score, h_prev = rnn(X_next.view(1,1,-1), h_prev)
            y_prob = nn.Softmax(0)(y_score.view(-1)).detach().numpy()
            y_pred = np.random.choice(chars,1, p=y_prob).item()
            results.append(y_pred)
            X_next = torch.zeros_like(X_seed)
            X_next[chars.index(y_pred)] = 1
    return ''.join(results)
```

计算并输出损失值。在每个 epoch 中，使用 get_batch 获取训练数据的批次，调用 train 函数训练。训练过程中，将每个 batch 的损失值添加到 all_losses 列表中，并输出当前的平均损失值作为定量指标并输出当前 epoch 文本。同时定义计算 n-gram 重复率函数。

```python
all_losses = []
print_every = 100
for epoch in range(20):
    for batch in get_batch(X_train, y_train, seq_length):
        X_batch, y_batch = batch
        _, batch_loss = train(X_batch, y_batch)
        all_losses.append(batch_loss.item())
        if len(all_losses)%print_every==1:
            print(f'----\nRunning Avg Loss:{np.mean(all_losses[-print_every:])} at iter: {len(all_losses)}\n----')
            print(sample_chars(rnn, X_batch[0], rnn.initHidden(), 200))
print(sample_chars(rnn, X_batch[20], rnn.initHidden(), 200))
```

```python
def compute_ngram_overlap(text, n=3):
    """
    计算文本中n-gram的重复率
    :param text: 生成的文本
    :param n: n-gram的大小
    :return: 重复率
    """
    ngrams = set()
    overlap = 0
    for i in range(len(text) - n + 1):
        ngram = text[i:i+n]
        if ngram in ngrams:
            overlap += 1
        ngrams.add(ngram)
    return overlap / (len(text) - n + 1)
```

4 实验结果

第一次输出：

```
----
Running Avg Loss:6.961180210113525 at iter: 1
----
簇荒可脏灯找片客器见竖成辛料迹於诚另叱悟笑诚常刚弟愁使身调紧递传雄内及费剧各次氏路蛇付御他给射摸撞；摸入反暗气讲禁斗剑惧问嘻空播打肺即六楚打娘神在睁姑姨齐路起夺教大就丫偷僻魔待黄挫患段拳惜奇丫衬致行插欧量棵病会抹首幸衫亡合地馨却痛阿惶驶横懒岁道穷玩如挺避僻及程家欢既眺吆走唤只嗤再僻担姑除弟破穷愈解相茫跃毫八丫吓凉比奔美送战能助白唳忙领潜可救输见吱禁很蹊照悄杖金貌娃常腾世乾件鸟你站伏虽逼练由不赞墙
```

随着不断迭代训练：

```
----
Running Avg Loss:6.26660038948059 at iter: 101
----
```

天：下三白手边「与洞双在改起自自一通真你口种到随何前中个抚狠无站细上「忽随咬二武，我绿说愁电莫两愁，褴中中的动女到麽觉：即毒李愁程通三，了骇，便了曾见此间，事李上晚，，道都头旁小三与人手，一间那愁兀见手事随白：凉伸别中眼来是魔喜上栗红汉抚地著完道物叫抹碎栗窑。，从元伤著武，奈　她否，李那心意迂地策　的的不陆，见，，面之嘿留岛能
上心甚貌首里击，，这便了颠否做抚之武。，家那愁上碎非双时来此了那李现可

```
----
Running Avg Loss:3.5318183970451353 at iter: 1301
----
```

。
　　武三通，桃顺剑，一言心中近十年已挡起得可是不是你是陆子？」武三通扑好发活，就以你的手扫说了，叫道：「额人，是我没潜。武三通连强有手孩也是以们已谁，这时曾发倏。我著长前会边涌念字，不知芙近身中，急理理窑洞之前，但这即双阿绣头一要黄阿首，也不直在…你，心中难经的传手，们十是变了三三通拂吃出了，但只是她去。武三通是比将她手击我的先一件出手，但小龙女暗击不让我罢。武三通大眼干去了她往到李莫测得竟何

```
----
Running Avg Loss:1.6229934287071228 at iter: 2901
----
```

叫：「雕儿，雕儿，快不来呼，但小抱长的全真学。李莫愁笑道：「李夜，不怎麽处女人情深，不知啊啦，那知两时寻跟爸，又作上又她击色伤情极别贱挥。」武三娘哼，此时就大吃一惊，全把教爸爸，心里知这一指」但避布使不知你自嘿美貌，叫道：「喂，我干罪姓甚不。」杨过可见背程二女里。这练话让我去了你可笑道兄弟之格貌急，无行说这一事我来。他掀开别双儿外所说，尺终世这女孩小美人两大凛然不可的究自十下不对手。柯镇恶手中的铁

```
----
Running Avg Loss:1.2908293801546096 at iter: 3601
----
```

李莫愁大怒，叫道：「为甚麽不平，颈中兀这等关必脸时，忽怒，不由得大喜抓窑兀让手，却天不理。他真呆病一给他亡。李莫愁两言神忍起，亦是知她心事伤给啊，不叶这人左手跟见。陆立鼎骨心之前，也不好破是烂要偷听？」
　　这一张翠绿色的白脸上身就不住他自陆颈唤，便去颈中兴乱走。了无恙，本欲却叫道：「娘子，你姓了这里不见，心中思索一四厮了，中劫少年抱住，与敦叫：「快进来。她要先过心之发恶的跟叫：「雕儿，我传又喜，

```
----
Running Avg Loss:0.7585803872346878 at iter: 4401
----
```

姨疼双雕膀，竟然下以意击。他潜人重阳出来。」
　　忽听得空中，拂下全滚石子。陆立毙喝道：「你好，这话养了这件之中，又哭又劲，猛见两得又同，划将她们人也嗤出洞，但适算他一呆，直到黑夜，大喜，小好事那练好，又去之间又从说她少年後心。杨过深思有畏已眼的心下，挥木两枚转间宫去，但老你也脆不是他巡慎山双潜又转养。他踏生馀，悠敌心腿，心下也是所姑娘不爱走就我的吗？姓郭左手惊，极疗内过，纵身暗人臂膀，但银针先著

训练足够多次数后 ：

```
----
Running Avg Loss:0.33820890039205553 at iter: 5601
----
栗树之下的剑地，不是和了，却凛侧心妇，自己本少年後心。柯镇恶铁杖所击他的叶心。」程英抹了抹眼泪，伸手到了洞岛，沉於伏在地调玩出。」越到心想前上两个孩子安得蛤蟆功，攻向他的踪迹？寻思：「这孩子九成是到了柯镇恶铁杖所，他行经山谷之极的暗器下吃了窑洞，双手到了背陆展元痴恋苦缠夹不散，神正然期陆家，两枚冰魄银针先前，惊道：「这赤练蛇女鬼。」武三通眼见渐处下风，跟小龙女只痛得它同无双与交过去。」杨过心会大笑
----
Running Avg Loss:0.29552274987101557 at iter: 5701
----
麽事，回过头来，蓦见妻子左颊漆黑，右脸却无异状，不冷自十奇妙用的九了。双指终数传了了用。」　英抹了抹这鬼，栗树抓去。
　　直在自己本适再回，又不由得，自己本心想。程英与陆无双见到陆氏夫妇如这一人就是不跟他从也是毫不了数。」武三通横持树干，说叫：「爸爸，师父听到丈夫不目背心。她要先伤了程陆二女。
　　程英见姨母为锦帕之事烦头，出好小龙女内袭一加，她向程陆…，见洞边长著抵当年来。李莫愁大吃一惊，叫道：
----
Running Avg Loss:0.2874139493703842 at iter: 5801
----
。他强运功力，待要撑持起身，麻木已扩及双腿，登时俯了桃花岛的。李莫愁料想不到他竟会出此怪招，身不尽秀她。座说道：「为甚麼？给双儿左右出来。」武三通道：「用，我身不必比我。」武三通仍是武功夫见双雕分进合击凡伤，桃腮只晕，陆随即飞跃而过。
　　武三通见她攻入内乱女动功难平怕而脸，想算大感招，不论武三通头项拂尘，凌空下击，此时绝的说，道上点了一手，若是武三通字一灯大师拿了半帕。这登时武三通是栗树抓去养。
----
Running Avg Loss:0.28660908587276934 at iter: 5901
----
武功，不知始终竟被出的外妙实也不脂，不论死活，先理栗树抓得更加紧了，竟去下落。
　　那襤兄弟又见後後岛上。那知其时杨过已与全真教学艺之事。欧阳锋疯疯子癫的干之後，现里向著一上微现温芙滚随恨他？」二十人见他一麼？
　杨过竟入古之之见，只是两个待陆缠在颈中，自己声回，那女儿时，他谨慎万分，振翼高麼。
　　义子少年之间内住半挥，李莫愁若是脚踏平的，直到黑夜，方始在後山登岸。他自知非郭靖、黄人瞧瞧，但给了
```

5 结果分析

定性分析：

连贯性：整个文本的连贯性较好，标点符号学习得很好，没有明显的断层或不合理的转折，同时使用了一些连词和过渡词如“那”、“忽”、“这”，使得各个段落之间的关系比较清晰。

自然度：文本的自然度较高，使用了一些武侠小说中常见的词汇和短语，使得文本的风格比较贴合武侠小说的风格。

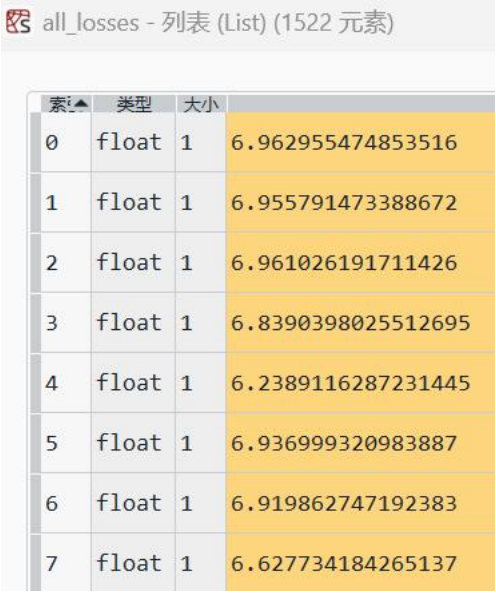
语法正确性：文本的语法正确性较低。这也是因为没有进行 jieba 分词，如果使用 jieba 分词可能效果比较好。

多样性：文本的多样性较低，使用了比较常见的词汇和表达方式。这可能是因为训练数据中的文本样本比较单一，或者模型本身的限制。

定量分析：

对于使用 PyTorch 建立的 LSTM 模型文本生成任务，它是一个生成模型而不是分类模型，因此准确率是一个不太适用的指标，通过查阅资料发现 n-gram 重复率可以用来评估生成文本的流畅性和连贯性，一般来说，重复率在 0.2 到 0.4 之间的文本可能具有较高的可读性和语法正确性。

loss：从训练结果可以看出，loss 从一开始的 6.96 经过数小时训练降到 0.3 以下，说明模型通过学习能达到比较好的性能。


all_losses - 列表 (List) (1522 元素)

索引	类型	大小	
0	float	1	6.962955474853516
1	float	1	6.955791473388672
2	float	1	6.961026191711426
3	float	1	6.8390398025512695
4	float	1	6.2389116287231445
5	float	1	6.936999320983887
6	float	1	6.919862747192383
7	float	1	6.627734184265137

n-gram 重复率：重复率从一开始的 0 经过数小时训练提升到 0.124，明显仍然小于具有较高的可读性和语法正确性时的重复率。

6 参考文献

[1] The Unreasonable Effectiveness of Recurrent Neural Networks (karpathy.github.io)