第四次作业

范想明-ZY2207304 xmfan@buaa.edu.cn

摘要

在生成式人工智能广泛应用的今天，文本生成模型倍受关注，如何不断研究并改进文本生成模型，从而满足用户需求成为一个热门的话题。本研究分别基于one-hot 和 word2vec 两种方法对语料库（金庸的两篇小说）进行编码作为 LSTM 模型的输入，通过不断训练得到 LSTM 模型，同时分析了两种模型的训练效果和原因。基于训练的模型以一段小说文本作为种子，生成了金庸风格的输出文本，但是受限于语料库和终端性能限制，生成的文本可读性有待提高。

0 研究背景

文本生成模型在自然语言处理（NLP）领域扮演着重要角色，其商业价值不断提升，用户对生成文本的要求也越来越高。基于语料库和深度神经网络的文本生成模型已成功应用于多个产品和领域。

在对话机器人中，基于文本生成的模型可以根据用户提出的问题或输入的文本，生成合适的回答和交流内容。微软的小冰、Apple 的 Siri、谷歌翻译以及淘宝的智能客服系统等产品都采用了这种文本生成技术。这些系统能够根据上下文理解用户意图，并生成自然、连贯的回答，提供高质量的交互体验。

此外，在自动文本摘要和文本简化领域，文本生成模型也发挥着重要作用。通过深度神经网络生成模型，可以将大段文本压缩成简明扼要的摘要，为用户提供快速获取信息的方式。类似地，文本简化模型能够将复杂的文本转化为易于理解的简化版本，使得普通用户也能轻松理解专业领域的文章。

总之，基于语料库和深度神经网络的文本生成模型在 NLP 领域发挥着重要作用，不断改进和研究文本生成模型，对于满足用户需求、提升用户体验以及推动 NLP 技术发展具有重要意义。

1 实验方法

本研究基于长短记忆单元模型（LSTM 模型）实现，LSTM 是一种改进型的 RNN 模型，其引入了选择性记忆机制，使其能够有选择性的记忆和忘却一些数

据信息，学习到上下文的前后关系，从而避免传统的 RNN 容易出现的梯度消失和梯度爆炸等问题。

由于 LSTM 处理的是序列到序列（sequence-to-sequence）的任务，因此将其应用于自然语言处理领域，需要对语言文本进行编解码。其中编码器通过对输入序列进行建模，捕捉序列中的上下文信息；解码器则利用编码器的信息来生成输出序列，保持生成文本的连贯性。目前常用的编解码方法有独热（one-hot）、词向量（word2vec）等。

本研究以金庸的部分小说作为语料库（受电脑性能限制，本研究仅针对两部小说）进行 LSTM 建模，并分别采用 one-hot 和 word2vec 两种编解码方式对语料库进行处理，将处理的结果用于 LSTM 模型的训练和测试。

2 实验步骤

2.1 基于 one-hot 的 LSTM 建模和测试

如图 1 所示，首先对语料库进行清洗，去除其中非汉字部分（为了保持生成文本的可读性，不去除部分汉语常用标点，如：，。？！）。

```
def preprocess_data():
    english = u'[a-zA-Z0-9’!"#$%&\'()*+-./:;「<=>?@★…【】《》“”‘’[\\]^_`{|}~+Saekmno]'
    str1 = u'[O①②③④⑤⑥⑦⑧⑨⓪_“”《》‘’「」『』（）<>【】·.—★~…]'
    remove_set = set(english + str1)

    # read the entire text
    file_paths = glob.glob(os.path.join(folder_path, file_extension))
    text = ""
    for path in file_paths:
        with open(path, "r", encoding='ANSI') as file:
            temp = file.read()
            result = filter(lambda x: x not in remove_set, temp)
            new_temp = ''.join(result)
            new_temp = new_temp.replace("\n", '')
            new_temp = new_temp.replace("\u3000", '')
            new_temp = new_temp.replace(" ", '')
            text = text + new_temp

    print('corpus length:', len(text))
    _corpus = [list(jieba.cut(text))]
    return _corpus
```

图 1 数据清洗

如图 2 所示，对清洗后的数据使用 jieba 进行分词，并基于分词结果建立词汇表（即词-索引的映射），然后建立长度为 50，step 为 3 的序列，以 one-hot 形式保存序列对应的下一个词。one-hot 又称独热编码，将每个词表示成具有 n 个元素的向量，这个词向量中只有一个元素是 1，其他元素都是 0，不同词汇元素为 0 的位置不同，其中 n 的大小是整个语料中不同词汇的总数，其优势在于操作简单，容易理解。

```
# all the vocabularies
text = list(jieba.cut(text))
vocab = sorted(list(set(text)))
print('total words:', len(vocab))
# create word-index dict
word_to_index = dict((c, i) for i, c in enumerate(vocab))
index_to_word = dict((i, c) for i, c in enumerate(vocab))
# cut the text into fixed size sequences
sentences = []
next_words = []
for i in range(0, len(text) - self.seq_length, self.step):
    sentences.append(list(text[i:i + self.seq_length]))
    next_words.append(text[i + self.seq_length])
print('nb sequences:', len(sentences))

# generate training samples
X = np.asarray([[word_to_index[w] for w in sent[:]] for sent in sentences])
y = np.zeros((len(sentences), len(vocab)))
for i, word in enumerate(next_words):
    y[i, word_to_index[word]] = 1
```

图 2 构建词-索引的映射，以 one-hot 形式编码序列

本研究基于 CPU 版本的 tesnsorflow2 和 keras 搭建 LSTM 框架，相关设置如图 3 所示，其中 encode 层采用 Embedding，中间层采用 LSTM（128 层），decode 层采用 Dense。基于 one-hot 编码后的语料训练 LSTM 模型，迭代次数设置为 20 次。

```
def load_model(self):
    # load a Sequential model
    model = Sequential()
    model.add(Embedding(len(self.vocab), self.embed_size, input_length=self.seq_length))
    model.add(LSTM(self.embed_size, input_shape=(self.seq_length, self.embed_size), return_sequences=False))
    model.add(Dense(len(self.vocab)))
    model.add(Activation('softmax'))
    self.model = model
    # compile the model
    optimizer = RMSprop(lr=0.005)
    self.model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

def fit_model(self, batch_size=128, nb_epoch=1):
    # fit the model with trainind data
    callbacks_list = [...]
    self.history = self.model.fit(self.X, self.y, batch_size=128, epochs=nb_epoch, callbacks=callbacks_list).history
```

图 3 模型建立及训练

2.2 基于 word2vec 的 LSTM 建模和测试

由于 one-hot 完全割裂了词与词之间的联系，并且随着语料库的增加，词典会变大，那么数据的维度就会变得越来越大，从而生成一个很稀疏的矩阵，因此会占据大量内存，由于上述缺点，其逐渐被诸如 word2index 的稠密向量表示方法所替代。word2vec 是一种流行的将词汇表示成向量的无监督训练方法，该过程将构建神经网络模型，将网络参数作为词汇的向量表示。本研究使用 gensim 库的 word2vec 构建词向量模型，模型构建前期的数据清洗和分词步骤与 one-hot 部分基本一致，因此不在此赘述。在本研究中分别使用其 word2index、index2word 进行编码、解码运算，构建语料库的词向量模型，如图 4 所示。

```python
def build_w2v_model(self, corpus):
    w2v_model = Word2Vec(corpus, vector_size=128, window=5, min_count=1, workers=4)
    self.w2v_model = w2v_model
    w2v_model.wv.save_word2vec_format('word2vec.txt')

def build_LSTM_model(self):
    # 把源数据变成一个长长的x，好让LSTM学会predict下一个单词
    raw_input = [item for sublist in corpus for item in sublist]

    text_stream = []
    vocab = self.w2v_model.wv.key_to_index
    for word in raw_input:
        if word in vocab:
            text_stream.append(word)
```

图 4 基于 word2vec 的编码

基于 word2vec 的 LSTM 模型构建方法和参数设置与 one-hot 部分类似，其以 word2vec 编码后的词向量作为输入，模型的迭代训练次数为 20 次。

3 实验结果与分析

对于基于 one-hot 的 LSTM 模型，其训练的结果如图 5 所示，随着迭代次数的增加，模型的损失（loss）逐渐减小，模型的准确率（accuracy）逐渐增加，训练至 20 次时，loss 已减小至 2.173，准确率上升至 0.5624，受输入模型的语料库尚不丰富，且迭代次数较小的限制，模型的准确率不高。

```
Epoch 1/20
479/479 [==============================] - 87s 177ms/step - loss: 6.9624 - accuracy: 0.1332 - lr: 0.0050
Epoch 2/20
479/479 [==============================] - 84s 176ms/step - loss: 6.3572 - accuracy: 0.1651 - lr: 0.0050
Epoch 3/20
479/479 [==============================] - 80s 167ms/step - loss: 6.0041 - accuracy: 0.2012 - lr: 0.0050
Epoch 4/20
479/479 [==============================] - 93s 193ms/step - loss: 5.7127 - accuracy: 0.2371 - lr: 0.0050
Epoch 5/20
479/479 [==============================] - 90s 187ms/step - loss: 5.5447 - accuracy: 0.2764 - lr: 0.0050
Epoch 6/20
479/479 [==============================] - 86s 180ms/step - loss: 5.5319 - accuracy: 0.3116 - lr: 0.0050
Epoch 7/20
479/479 [==============================] - 87s 182ms/step - loss: 5.4235 - accuracy: 0.3390 - lr: 0.0050
Epoch 8/20
479/479 [==============================] - 81s 169ms/step - loss: 5.4083 - accuracy: 0.3601 - lr: 0.0050
Epoch 9/20
479/479 [==============================] - 88s 185ms/step - loss: 5.3296 - accuracy: 0.3744 - lr: 0.0050
Epoch 10/20
479/479 [==============================] - 87s 181ms/step - loss: 5.1743 - accuracy: 0.3869 - lr: 0.0050
Epoch 11/20
479/479 [==============================] - 85s 178ms/step - loss: 4.9489 - accuracy: 0.3993 - lr: 0.0050
Epoch 12/20
479/479 [==============================] - 88s 184ms/step - loss: 4.7141 - accuracy: 0.4076 - lr: 0.0050
Epoch 13/20
479/479 [==============================] - 94s 197ms/step - loss: 4.4261 - accuracy: 0.4209 - lr: 0.0050
Epoch 14/20
479/479 [==============================] - 92s 193ms/step - loss: 4.0959 - accuracy: 0.4362 - lr: 0.0050
Epoch 15/20
479/479 [==============================] - 81s 169ms/step - loss: 3.7572 - accuracy: 0.4488 - lr: 0.0050
Epoch 16/20
479/479 [==============================] - 80s 167ms/step - loss: 3.3747 - accuracy: 0.4700 - lr: 0.0050
Epoch 17/20
479/479 [==============================] - 82s 171ms/step - loss: 3.0359 - accuracy: 0.4888 - lr: 0.0050
Epoch 18/20
479/479 [==============================] - 89s 185ms/step - loss: 2.7042 - accuracy: 0.5150 - lr: 0.0050
Epoch 19/20
479/479 [==============================] - 83s 173ms/step - loss: 2.4172 - accuracy: 0.5388 - lr: 0.0050
Epoch 20/20
479/479 [==============================] - 85s 178ms/step - loss: 2.1730 - accuracy: 0.5624 - lr: 0.0050
```

图 5 LSTM 模型训练结果

基于训练后的模型，以一段原文作为输入进行测试，模型基于输入每次输出一个词（也可能是一个字或标点），模型的输出如图 6 所示，虽然按要求输出了文本，而且有标点符号，但是受限于语料库和终端性能，训练出的模型生成的文本可读性不高。

```
diversity: 0.2
Generating with seed:白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马，
白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马，成了吗？
怎么会到这样大意？狄云道那郎中先生呢？只怕这小恶僧假装不是知道，有不相信狄云心想欺到小师父，我还是将女儿杀了，我却早给你啦？这样没喜欢这小了
diversity: 0.5
Generating with seed: 白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马
白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马，成了？
却想哭那吓得二人卜来，他却哪里有鱼？谁来打扰了书中他的来侵犯，跟着左手 ·扯，便将性命的姑娘性命。功夫，在花铁干身边我爹爹在傻之
diversity: 1.0
Generating with seed: 白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马
白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马，跃起胡了，
手指 ·松，听来了再 ·探跟著半晌，跳起身来，便已不见长剑脱手，身子 ·晃，站在 ·旁，只见白马李三夫妇了几卜，突觉递出的神色，跃在丁典的刘乘风
diversity: 1.2
Generating with seed: 白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马
白马啸西风得得得，得得得得得得，得得得在黄沙菲菲的回疆大漠之上，尘沙飞起两丈来高，两骑马 ·前 ·後的急驰而来。前面是匹高腿长身的白马，说谎不定，
想无法坐倒。万圭徼 ·来流血 ·起渐渐地说啊他 ·声，将长剑砍在旁。只听得水笙大叫，叫双手好生的 ·角，竟始终，逃的回到放开她不敢血刀老祖哈哈大笑，
```

图 6 基于 one-hot 的 LSTM 模型输出文本

对于基于 word2vec 的 LSTM 模型，其训练的结果如图 7 所示，可以看出当训练次数达到第 3 次时，模型的 loss 已为一个较低的值，accuracy 也已到达 0.52，且两个参数在后续的迭代中基本不变，说明模型经过较少次数的迭代已达到一个较好的状态。

```
Epoch 1/20
20/20 [==============================] - 33s 2s/step - loss: 0.0991 - accuracy: 0.0055 - lr: 0.0010
Epoch 2/20
20/20 [==============================] - 33s 2s/step - loss: 0.0067 - accuracy: 0.3382 - lr: 0.0010
Epoch 3/20
20/20 [==============================] - 33s 2s/step - loss: 0.0062 - accuracy: 0.2517 - lr: 0.0010
Epoch 4/20
20/20 [==============================] - 26s 1s/step - loss: 0.0061 - accuracy: 0.5209 - lr: 0.0010
Epoch 5/20
20/20 [==============================] - 31s 2s/step - loss: 0.0060 - accuracy: 0.5209 - lr: 0.0010
Epoch 6/20
20/20 [==============================] - 32s 2s/step - loss: 0.0060 - accuracy: 0.5209 - lr: 5.0000e-04
Epoch 7/20
20/20 [==============================] - 31s 2s/step - loss: 0.0060 - accuracy: 0.5209 - lr: 5.0000e-04
Epoch 8/20
20/20 [==============================] - 31s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 2.5000e-04
Epoch 9/20
20/20 [==============================] - 50s 3s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 1.2500e-04
Epoch 10/20
20/20 [==============================] - 31s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 6.2500e-05
Epoch 11/20
20/20 [==============================] - 29s 1s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 3.1250e-05
Epoch 12/20
20/20 [==============================] - 32s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 1.5625e-05
Epoch 13/20
20/20 [==============================] - 32s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 7.8125e-06
Epoch 14/20
20/20 [==============================] - 32s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 3.9063e-06
Epoch 15/20
20/20 [==============================] - 34s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 1.9531e-06
Epoch 16/20
20/20 [==============================] - 32s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 9.7656e-07
Epoch 17/20
20/20 [==============================] - 28s 1s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 4.8828e-07
Epoch 18/20
20/20 [==============================] - 30s 1s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 2.4414e-07
Epoch 19/20
20/20 [==============================] - 33s 2s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 1.2207e-07
Epoch 20/20
20/20 [==============================] - 29s 1s/step - loss: 0.0059 - accuracy: 0.5209 - lr: 6.1035e-08
```

图 7 基于 word2vec 的 LSTM 模型训练结果

基于 word2vec 的 LSTM 模型经训练后以一段文本作为输入，其生成的文本如图 8 所示，可读性同样不高。

```
diversity: 0.2
Generating with seed: 那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又
那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又阿曼在问在到毒针里会著我著得两个上的做会？用而会著向上说这？
之中著叫的车尔库说还到车尔库说便上为阿曼谁之计打用著杀？问
diversity: 0.5
Generating with seed: 那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又
那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又老人等阿曼可是？会做不知会跟迷宫？车尔库向汉人说车尔库会便的很叫只见？
可便老人老人阿曼车尔库只见跟大却老人著？汉人听两人老人很叫没有谁过也毒针老人说
diversity: 1.0
Generating with seed: 那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又
那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又？上说道在会说打；没有之便中之会向来老人却说谁便跟一能两人向之中在车尔库
很在便跟向有谁一著会只阿曼？？著毒针没有之中人上
diversity: 1.2
Generating with seed: 那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又
那汉子左边背心上却插著一枝长箭。鲜血从他背心流到马背上，又谁死两个之两人便便不是毒针阿曼毒针会谁毒针？车尔库这两个阿曼向叫在迷宫说向在迷宫没有大？
人老人人阿曼会没有老人向著阿曼？向人不敢等两个车尔库曰上上
```

图 8 基于 word2vec 的 LSTM 模型输出文本

通过对比基于 one-hot 的 word2vec 两种文本编解码方法的 LSTM 模型训练的结果可以看出，基于 word2vec 的 LSTM 模型训练无论是训练的效率还是模型的 loss 均优于基于 one-hot 的 LSTM 模型，反映出 word2vec 方法优于 one-hot 方法。