深度学习与自然语言处理第五次作业

（基于 Seq2Seq 模型的文本生成）

姓名：刘千歌

学号：BY2139121

一、 实验目标

基于 Seq2seq 模型来实现文本生成的模型，输入可以为一段已知的金庸小说段落，来生成新的段落并做分析

二、 背景介绍

本次实验应用的 Seq2Seq 是自然语言处理技术中的一种重要模型，常被用于机器翻译、对话系统以及自动文摘。其解决问题的主要思路是通过深度神经网络模型（常用的是 LSTM 长短记忆网络）将一个输入的序列映射为一个作为输出的序列，其实现包含编码器（Encoder）与解码器（Decoder）两个部分。

2.1 RNN 结构及使用

RNN 基本的模型如下图所示，每个神经元接受的输入包括：前一个神经元的隐藏层状态 h(用于记忆) 和当前的输入 x (当前信息)。神经元得到输入之后，会计算出新的隐藏状态 h 和输出 y，然后再传递到下一个神经元。因为隐藏状态 h 的存在，使得 RNN 具有一定的记忆功能。

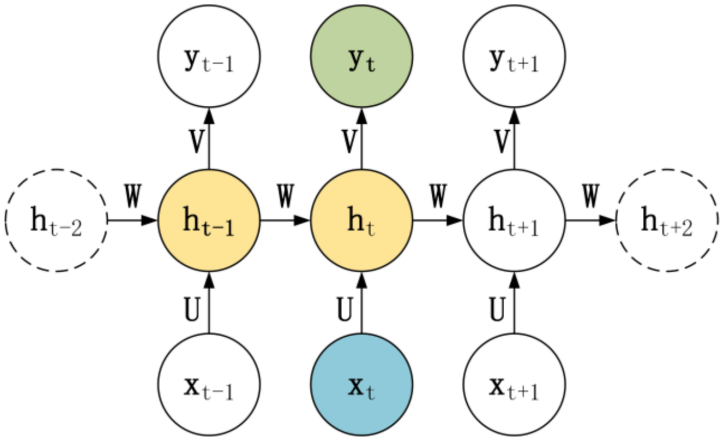

图 1 RNN 结构

2.2 Seq2seq 模型介绍

普通 RNN 结构对其输入和输出个数都有一定的限制，但实际中很多任务的序列的长度是不固定的，例如机器翻译中，源语言、目标语言的句子长度不一样；对话系统中，问句和答案的句子长度不一样。

Seq2Seq 是一种重要的 RNN 模型，也称为 Encoder-Decoder 模型，可以理解为一种 N×M 的模型。模型包含两个部分：Encoder 用于编码序列的信息，将任意长度的序列信息编码到一个向量 c 里。而 Decoder 是解码器，解码器得到上下文信息向量 c 之后可以将信息解码，并输出为序列。

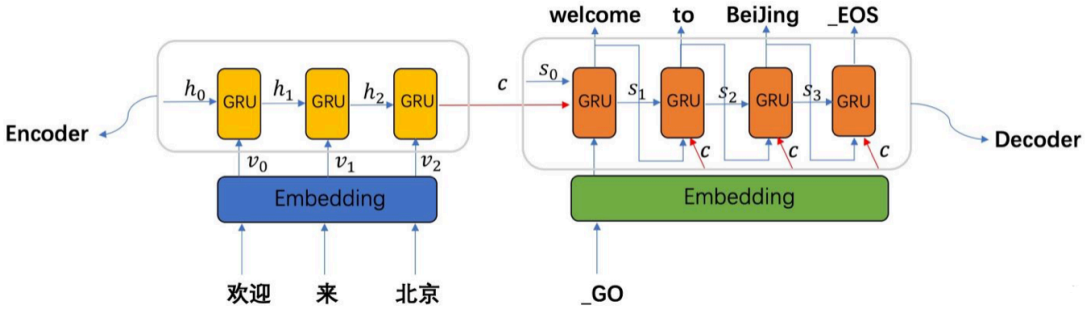

图 2 Encoder-Decoder 模式架构

三、 实验流程

3.1 文本预处理与数据清洗

首先需要将原文本文件进行预处理。使用正则模块替换掉除逗号外的特殊符号，并清洗括号内的文字；只保留包括汉字、字母、数字、中文符号在哪的有效数据，清洗其他乱码字符。

```
def load_trainset(data):
    # 利用正则规则进行数据预处理，替换特殊符号
    pattern = re.compile(r'\(.*\)')
    data = [pattern.sub('', lines) for lines in data]
    data = [line.replace('……', '。') for line in data if
len(line) > 1]
    data = ''.join(data)
    data = [char for char in data if is_uchar(char)]
     data = ''.join(data)
    # 使用 jieba 进行分词
    word_data = list(jieba.cut(data))
    return word_data
```

3.2 生成字符向量

在获取到原始文本信息后，需将字符映射为能够输入到模型中的数字编码形式具体的步骤包括：

1）建立汉字和数字互相映射的字典，其效果如下：

```
16, '光影': 14117, '求来': 14118, '湿书见': 14119, '采桑子': 14120, '群僧质': 14121, '七星': 14122, '平
14124, '御风而行': 14125, '偷入': 14126, '不动色': 14127, '御害': 14128, '南望': 14129, '务令': 14130,
小': 14132, '绣花针': 14133, '一事不明': 14134, '乐声': 14135, '老爷爷': 14136, '窝洞': 14137, '老幼':
, '不仅': 14140, '边射边': 14141, '芩': 14142, '大采': 14143, '囚犯': 14144, '前后左右': 14145, '抵触':
147, '劳什子': 14148, '绿火': 14149, '这三毒': 14150, '慧夫道': 14151, '花信年华': 14152, '四件': 14153
```

2）建立字典后将原始文本数据映射为矩阵形式的数字向量，其效果如下：

```
数字数据信息:
 [14635 49319 31832 40433 43614 29103 21241 20123   120 35863 40433  9640
 20877 18468 31832 14502 13153 38531 15686 51876 42093 11806 21684 46569
 44244 33786 11684 36964 50820 24520 24836  9528  7597 26691  7597 50520
 17814  2908 22425 25286  7928 40490 45316  8847 38531 52510 30953 23365
 28074  5789  5801  8633 19637 52409 44273 40169 21021 24638 17109 26173
 37505 45279 21114 26176  4567 44410 25057  8769 22655 16797 10216 37831
 37255 43174 35471  4037 18752 15797 30470 13798 15480 38143 23970 38531
 47572 11044 14784  1114 43218 21271 21271 52047 43112   264  1369  8670
 38872  6240 46545 47762]
```

最终字典长度为 52575 个，具体代码如下：

```
def vector_generate(word_data):
    word_vocab = set(word_data)
    word2id = {c: i for i, c in enumerate(word_vocab)}
return word2id

#生成词语与 one-hot 向量的映射向量
word2id = vector_generate(word_data)
#转换数据为数字格式
numdata = [word2id[word] for word in word_data]
numdata = np.array(numdata)
```

3.3 设计数据生成器

每篇文章字数很多，我们通常不会将他们一股脑全部扔到神经网络中进行训练，而是将数据分为一个 batch 一个 batch 的进行分批训练。由此，本环节设计函数将数据按照 [batch_size, time_steps]形式一个个切分，这种数据也是循环神经网络训练中使用的格式。所谓的 time_step 即与当前状态有关的时间步长，假设 time_step 为 5，则模型的输入与输出案例如下图所示，输入 input data 向量，输出 output data， 23597 为预测输出。

```
input data: [40637 34283 48138  1548 44889]
output data: [34283 48138  1548 44889 23597]
```

3.4 RNN 模型建立

本次试验使用 tensorflow 框架实现 RNN 模型。

```
class RNNModel():
    """docstring for RNNModel"""
    def __init__(self, BATCH_SIZE, HIDDEN_SIZE, HIDDEN_LAYERS,
VOCAB_SIZE, learning_rate):
        super(RNNModel, self).__init__()
        self.BATCH_SIZE = BATCH_SIZE
        self.HIDDEN_SIZE = HIDDEN_SIZE
        self.HIDDEN_LAYERS = HIDDEN_LAYERS
        self.VOCAB_SIZE = VOCAB_SIZE

        # 定义占位符
        with tf.name_scope('input'):
            self.inputs = tf.placeholder(tf.int32, [BATCH_SIZE,
None])
            self.targets = tf.placeholder(tf.int32, [BATCH_SIZE,
None])
            self.keepprb = tf.placeholder(tf.float32)

        # 定义词嵌入层
        with tf.name_scope('embedding'):
            embedding = tf.get_variable('embedding', [VOCAB_SIZE,
HIDDEN_SIZE])
            emb_input = tf.nn.embedding_lookup(embedding,
self.inputs)
            emb_input = tf.nn.dropout(emb_input, self.keepprb)

        # 搭建 lstm 结构
        with tf.name_scope('rnn'):
            lstm = tf.nn.rnn_cell.LSTMCell(HIDDEN_SIZE,
state_is_tuple=True)
            lstm = tf.nn.rnn_cell.DropoutWrapper(lstm,
output_keep_prob=self.keepprb)
            cell = tf.nn.rnn_cell.MultiRNNCell([lstm] *
HIDDEN_LAYERS)
            self.initial_state = cell.zero_state(BATCH_SIZE,
tf.float32)
            outputs, self.final_state = tf.nn.dynamic_rnn(cell,
emb_input, initial_state=self.initial_state)

        # 重新 reshape 输出
        with tf.name_scope('output_layer'):
            outputs = tf.reshape(tf.concat(outputs, 1), [-1,
HIDDEN_SIZE])
            w = tf.get_variable('outputs_weight', [HIDDEN_SIZE,
VOCAB_SIZE])
            b = tf.get_variable('outputs_bias', [VOCAB_SIZE])
            logits = tf.matmul(outputs, w) + b

        # 计算损失
        with tf.name_scope('loss'):
```

```
            self.loss = sequence_loss_by_example([logits],
[tf.reshape(self.targets, [-1])],

[tf.ones([BATCH_SIZE * TIME_STEPS],

dtype=tf.float32)])
            self.cost = tf.reduce_sum(self.loss) / BATCH_SIZE

        # 优化算法
        with tf.name_scope('opt'):
            # 学习率衰减
            global_step = tf.Variable(0)
            learning_rate =
tf.train.exponential_decay(learning_rate, global_step,
BATCH_NUMS, 0.99, staircase=True)

            #通过 clip_by_global_norm()控制梯度大小
            trainable_variables = tf.trainable_variables()
            grads, _ =
tf.clip_by_global_norm(tf.gradients(self.cost,
trainable_variables), MAX_GRAD_NORM)
            self.opt =
tf.train.AdamOptimizer(learning_rate).apply_gradients(zip(grads,
trainable_variables))

        # 预测输出
        with tf.name_scope('predict'):
            self.predict = tf.argmax(logits, 1)
```

3.4. 定义模型训练参数

如下所示，对于预期达到的网络结构，我选用参数包括网络层级 3 层，节点数目 512 个，训练周期 1000 个。

```
VOCAB_SIZE = len(set(word_data))
EPOCHS = 1000
BATCH_SIZE = 8
TIME_STEPS = 100
BATCH_NUMS = len(numdata) // (BATCH_SIZE * TIME_STEPS)
HIDDEN_SIZE = 512
HIDDEN_LAYERS = 6
MAX_GRAD_NORM = 1
learning_rate = 0.05
```

3.5 模型训练和保存

将训练模型存储到本地的 checkpoints 目录下，相关日志存储在 logs 目录。

```
#模型训练
model = RNNModel(BATCH_SIZE, HIDDEN_SIZE, HIDDEN_LAYERS,
VOCAB_SIZE,TIME_STEPS)
# 保存模型
saver = tf.train.Saver()
with tf.Session() as sess:
     #记录日志
     writer = tf.summary.FileWriter('logs/tensorboard',
     tf.get_default_graph())
     sess.run(tf.global_variables_initializer())
     #针对每个周期开启遍历
     for k in range(EPOCHS):
         state = sess.run(model.initial_state)
         train_data = data_generator(numdata, BATCH_SIZE,
     TIME_STEPS)
         total_loss = 0.
         for i in range(BATCH_NUMS):
           #添加数据读取的错误处理
           try:
                xs, ys = next(train_data)
                feed = {model.inputs: xs, model.targets: ys,
     model.keepprb: 0.8, model.initial_state: state}
                costs, state, _ = sess.run([model.cost,
     model.final_state, model.opt], feed_dict=feed)
                total_loss += costs
                if (i + 1) % 50 == 0:
                   print('epochs:', k + 1, 'iter:', i + 1,
     'cost:', total_loss / i + 1)
          except StopIteration:
                break
         saver.save(sess, './checkpoints/lstm.ckpt')
```

3.6 模型测试

将测试文字放入项目目录下 test_text.txt 文件中（内容可自行替换）。进一步，从本地目录读取刚训练好的模型参数，在测试段落基础上续写 100 字。

```
with open('./test_text.txt', 'r', encoding='utf-8') as f:
          test_string = f.readlines()
          print(f)
          T = []
          test_process_string =load_trainset(test_string)
          print(test_process_string)
          for word in test_process_string:
```

```
                tmp = word2id[word]
                T.append(tmp)
            print(T)
            print(np.array([T]))

            tf.reset_default_graph()
            evalmodel = RNNModel(1, HIDDEN_SIZE, HIDDEN_LAYERS,
VOCAB_SIZE, learning_rate)
            # 加载模型
            saver = tf.train.Saver()
            with tf.Session() as sess:
                saver.restore(sess, './checkpoints/lstm.ckpt')
                new_state = sess.run(evalmodel.initial_state)
                x = np.array([T])
                samples = []
                for i in range(100):
                    feed = {evalmodel.inputs: x, evalmodel.keepprb:
1., evalmodel.initial_state: new_state}
                    c, new_state = sess.run([evalmodel.predict,
evalmodel.final_state], feed_dict=feed)
                    for j in range(len(T)):
                        x[0][j] = c[0]
                        samples.append(c[0])
                print(x)
                print(len(c))
                print(samples)
                print('test:', ''.join([id2word[index] for index
in samples]))
```

四、实验结果与分析

本次实验，是将天龙八部里的一句话输入训练好的模型对内容进行续写，输入的原文为：

数十名帮众从四面八方围将上来，手中各持一捆药草，点燃了火，浓烟直冒。段誉刚从地下爬起，突然一阵头晕，又即摔倒，迷迷糊糊之中只见钟灵的身子不住摇晃，跟着也即跌倒。两名帮众奔上来想揪住钟灵，闪电貂护主，跳过去在俩人身上各咬了一口。众人大骇倒退，四下里团团围住，叫嚷吆喝，却无从下手。司空玄叫道：“东方烧雄黄，南方烧麝香，西方北方人人散开。”诸帮众应命烧起麝香、雄黄。神农帮无药不备，药物更是无一

而非上等精品，这麝香、雄黄质纯性强，一经烧起，登时发出气味辛辣的浓烟，顺着东南风向钟灵吹去。不料闪电貂却不怕药气，仍是矫夭灵活，霎时间又咬倒了五名帮众。

司空玄眉头一皱，计上心来，叫道：“铲泥掩盖，将女娃娃连毒貂一起活埋了。

输出的结果是：

丁计是的在哭衣，误了匆匆在手掌豪你人截下的江肚肠，汉江南。迸厚人的侠到生死符。蒙珍救你，岩石法儿不至于害我呜呜咽咽，钟灵断肠散尊山惨象沉大块。

模型输出的结构能够答题看出武侠小说的轮廓，但是整体逻辑不知所云。在后续的优化工作上：模型的模型的优化可以从网络结构和学习率优化上进行提升。然而我的电脑能力有限，我选择的网络结构较为简单，可以尝试更深的结构，更多的隐藏层节点数。就能得到更好的结果。

五、遇到的问题

因为我是 Mac M1 芯片，在配置 tensorflow 环境的时候遇到很多问题：需要安装 Xcode Command Line Tools，以及 arm 架构对应的 miniforge 版本版本——通过创建 conda 虚拟环境，才可以安装以及使用 Apple-TensorFlow2.4。