基于RNN的古诗生成器

在了解RNN和LSTM相关知识后，基于RNN实现了一个古诗生成器，可实现自动作诗和藏头诗生成。

1. 数据预处理

训练集初始数据来源于/origin-data/poetry.txt文件，存放四万多首古诗，源码见process-data.py。

处理步骤如下：

- 从文件中读取唐诗，逐行处理，去空转码等，将诗词内容存放到列表中
- 获取唐诗内容中所有的字符，统计词频并降序排列
- 生成单词到id的映射，将诗词内容转换为向量形式写入文件/processed-data/poetry.txt

2. 模型编写

该部分包括两个模型的编写，分别用于训练和验证（生成古诗体），模型大体相似，当进行验证时，验证模型读取训练模型的参数进行覆盖，源码见rnn-models.py。

训练模型：

- 初始化数据、标签、LSTM层dropout保留率、RNN深度等
- 构建RNN网络结构如下：

```
def cell(self):
    """
    rnn网络结构
    """
    lstm_cell = [

 tf.nn.rnn_cell.DropoutWrapper(tf.nn.rnn_cell.BasicLSTMCell(HIDDEN_SIZE), \
        output_keep_prob=self.rnn_keep) for _ in range(NUM_LAYERS)]
    cell = tf.nn.rnn_cell.MultiRNNCell(lstm_cell)
    return cell
```

- 定义前向传播、反向传播过程及损失函数

验证模型：

- 初始化输入、embedding层dropout保留率、LSTM层dropout保留率等
- 构建RNN网络结构，和训练模型相同
- 定义前向传播过程函数
- 使用softmax计算概率

3. 组织数据集

为方便训练使用，定义方法read-data( )从processed-data中读取数据，构建数据集，每个batch大小设置为64，返回训练数据和训练标签。定义方法next-batch( )获取下一个batch。源码见dataset.py。

4. 模型训练

循环训练指定个数batch，训练过程中显示当前步数以及loss值，并定期保存当前模型及对应的checkpoint于ckpt文件夹，相关代码如下。源码见train.py。

```
with tf.Session() as sess:
      # 初始化
    sess.run(tf.global_variables_initializer())
    for step in range(TRAIN_TIMES):
        # 获取训练batch
        x, y = data.next_batch()
        # 计算loss
        loss, _ = sess.run([model.loss, model.optimize],
                           {model.data: x, model.labels: y,
                            model.emb_keep: setting.EMB_KEEP,
                            model.rnn_keep: setting.RNN_KEEP})
        if step % SHOW_STEP == 0:
            print('step {}, loss is {}'.format(step, loss))
        # 保存模型
        if step % SAVE_STEP == 0:
            saver.save(sess, setting.CKPT_PATH,
global_step=model.global_step)
```

5. 模型验证

- **自动作诗：**

1. 首先，读取训练模型保存的参数，覆盖验证模型的参数
2. 将开始符号's'作为输入传给模型，模型将输出下一个字符为此表中各词的概率，以及rnn传递的state
3. 根据2中输出的概率，使用AOC概率选择法，随机出下一个字

4. 将随机出来的字作为输入，前一次输出的state作为本次输入的state传给模型，模型将输入下一个字符为此表中各词的概率，以及rnn传递的state
5. 重复3,4步骤，直到随机出结束符'e',生成结束。过程中生成的所有字符，构成本次生成的古体诗('s'和'e'不计)

   代码如下：

```
def generate_poem():
    """
    随机生成一首诗歌
    :return:
    """
    with tf.Session() as sess:
        # 加载最新的模型
        ckpt = tf.train.get_checkpoint_state('ckpt')
        saver.restore(sess, ckpt.model_checkpoint_path)
        # 预测第一个词
        rnn_state = sess.run(model.cell.zero_state(1, tf.float32))
        x = np.array([[word2id_dict['s']]], np.int32)
        prob, rnn_state = sess.run([model.prob, model.last_state],
                                   {model.data: x, model.init_state:
rnn_state,
                                    model.emb_keep: 1.0,
                                    model.rnn_keep: 1.0})
        word = generate_word(prob)
        poem = ''
        # 循环操作，直到预测出结束符号‘e’
        while word != 'e':
            poem += word
            x = np.array([[word2id_dict[word]]])
            prob, rnn_state = sess.run([model.prob,
model.last_state],
                                       {model.data: x,
model.init_state: rnn_state,
                                        model.emb_keep: 1.0,
                                        model.rnn_keep: 1.0})
            word = generate_word(prob)
        # 打印生成的诗歌
        print(poem)
```

- **藏头诗生成：**

藏头诗生成与随机生成古体诗类似，不同的是在开始和预测生成标号时传入句头字符。

代码如下：

```python
def generate_acrostic(head):
    """
    生成藏头诗
    :param head:每行的第一个字组成的字符串
    """
    with tf.Session() as sess:
        # 加载最新的模型
        ckpt = tf.train.get_checkpoint_state('ckpt')
        saver.restore(sess, ckpt.model_checkpoint_path)
        # 进行预测
        rnn_state = sess.run(model.cell.zero_state(1, tf.float32))
        poem = ''
        cnt = 1
        # 一句句生成诗歌
        for x in head:
            word = x
            while word != ', ' and word != '。':
                poem += word
                x = np.array([[word2id_dict[word]]])
                prob, rnn_state = sess.run([model.prob,
model.last_state],
                                           {model.data: x,
model.init_state: rnn_state,
                                            model.emb_keep: 1.0,
                                            model.rnn_keep: 1.0})
                word = generate_word(prob)
                if len(poem) > 25:
                    print('bad.')
                    break
            # 根据单双句添加标点符号
            if cnt & 1:
                poem += ', '
            else:
                poem += '。'
            cnt += 1
        # 打印生成的诗歌
        print(poem)
        return poem
```

- **词生成：**

  选择概率最高的前100个词，并用AOC概率选择法选取最终结果，代码如下：

```
def generate_word(prob):
    """
    :param prob: 概率向量
    :return: 生成的词
    """
    prob = sorted(prob, reverse=True)[:100]
    index = np.searchsorted(np.cumsum(prob), np.random.rand(1) *
np.sum(prob))
    return id2word_dict[int(index)]
```

6. 配置文件

一些相关参数值及映射函数等不做赘述，源码见setting.py及maps.py

7. 实验环境

Tool	Version
Python	3.6.7 64-bit
TensorFlow	1.12.0
IDEL	Visual Studio Code 1.30.0

Platform/Hardware	Info
MacBook Pro	macOS Mojava
CPU	2.9 GHz Intel Core i5
GPU	Intel Iris Graphics 550 1536 MB

8. 结果

- **运行**

```
>_ python3.6 valid.py
Number：生成诗的数目
Head：藏头诗首字组成的字符串
```

- **随机作诗**

截取较好五组诗句如下：

> 行座清成浪，花频旧起还。还人欲送面，山计易傍吟。
>
> 酒当含旆千管夫，泪此平清保满天。不问不情空可梦，经来无坐客无来。
>
> 矫城枕气移，向舍凤沼烟。茶旗飘层钿，云丛退返间。青庭昏重意，阵印茉金光。后增变远实，伊疲任守钟。宣风已后遂，知禄雪仙筵。
>
> 伊时半据可竟枢，夜朵生元非羽城。声上眼他有欲客，恐怀尘更指三沦。
>
> 夏里上无更似南，消风日政水无同。尽心宿作桐萦起，荏复无留隔海回。一绕团纹终里起，功闲偏倚折深台。聊愁槿跸万垂侣，全是丹收必松休。

- **藏头诗**

输入“清华”，截取较好五组诗句如下：

> 清蜜森却坤沟朱芙危露，华河空寒吹袖拊飞神见。
>
> 清陷旻彩窈流冷仪，华脸场江萝紫点杨。
>
> 清场被乌徊药早苔半，华花远炉窗惊霄石明。
>
> 清适见夜红行屋，华相闲楼廊幽石。
>
> 清中水春低落磬云眠，华兵桐塔花翠岸鸣大羽。

9. 分析

通过对RNN及LSTM的学习，完成了此次作业，相较于成熟的自动作诗案例，还有很多不足，主要原因可能在于训练集太小，训练模型也有很多欠缺。

相对藏头诗，自动作诗更为完善，但仅限于格式正确，语句可读性很差，感觉在训练模型的时候结合语义，会有很大的优化，联系之前的HMM模型，工作可行且工作量不大，但模型训练时间应该会翻倍。与此同时。我观测到藏头诗的输出有很多的问题，譬如"清华"，在观测到间隔符'，'需要生成后，模型经常只输出'华'字即停止，或是生成间隔符后即停止，推测可能模型过于简单或者训练效果不好，可以选择更大的数据集进行训练对比结果。

通过计算语言学的学习，进一步了解了NLP，熟悉了数据科学分析的常用方法，受益颇多。