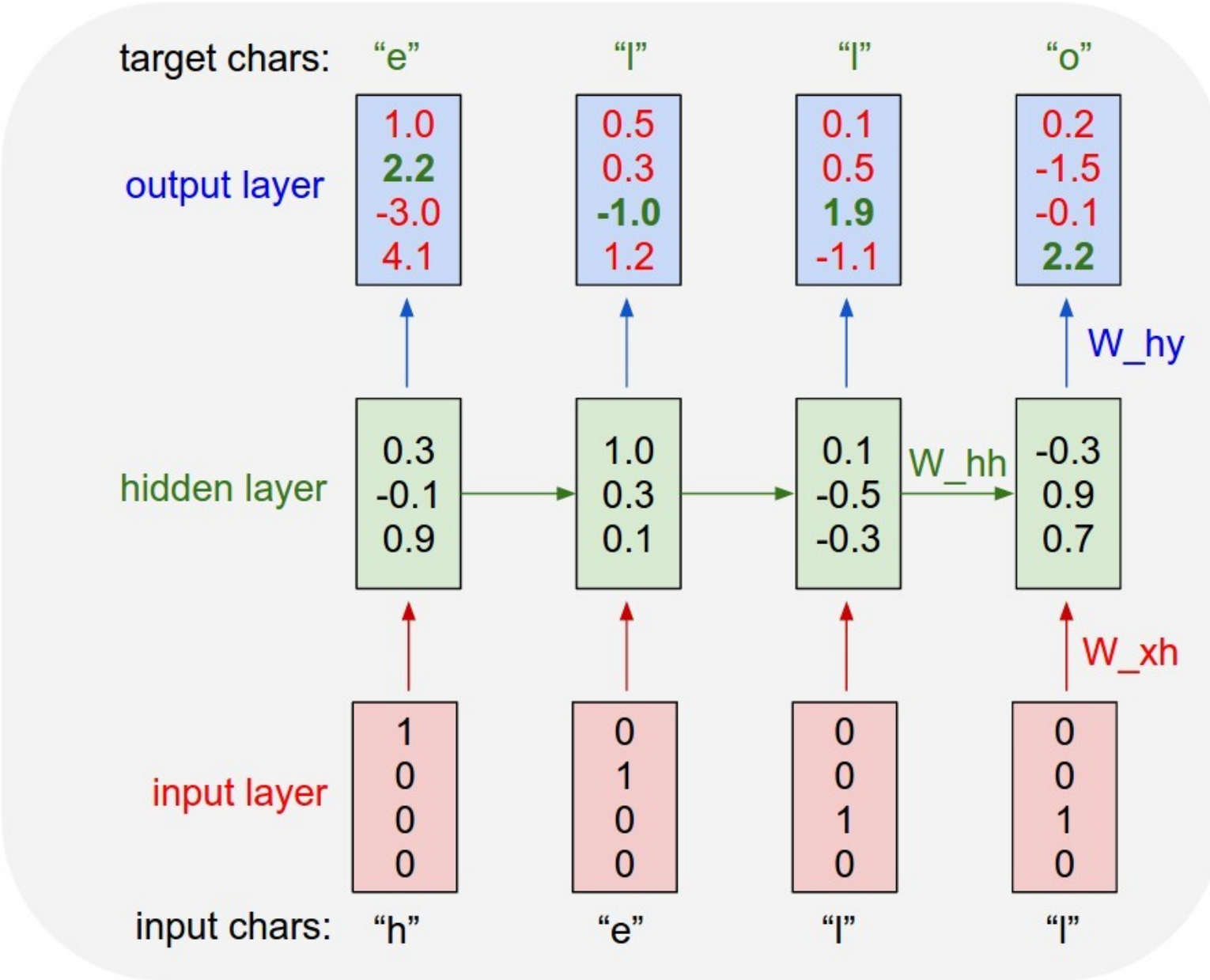


《安娜卡列尼娜》文本生成——利用TensorFlow构建LSTM模型

天雨粟

模型师傅 / 果粉

前言

最近看完了LSTM的一些外文资料，主要参考了Colah的blog以及Andrej Karpathy blog的一些关于RNN和LSTM的材料，准备动手去实现一个LSTM模型。代码的基础框架来自于Udacity上深度学习纳米学位的课程（付费课程）的一个demo，我刚开始看代码的时候真的是一头雾水，很多东西没有理解，后来反复查阅资料，并我重新对代码进行了学习和修改，对步骤进行了进一步的剖析，下面将一步步用TensorFlow来构建LSTM模型进行文本学习并试图去生成新的文本。本篇文章比较适合新手去操作，LSTM层采用的是BasicLSTMCell。

关于RNN与LSTM模型本文不做介绍，详情去查阅资料过着去看上面的blog链接，讲的很清楚啦。这篇文章主要是偏向实战，来自己动手构建LSTM模型。

数据集来自于外文版《安娜卡列妮娜》书籍的文本文档（本文后面会提供整个project的git链接）。

工具介绍

语言：Python 3
包：TensorFlow及其它数据处理包（见代码中）
编辑器：jupyter notebook
线上GPU：floyd

正文部分

正文部分主要包括以下四个部分：

- **数据预处理：**加载数据、转换数据、分割数据mini-batch

- **模型构建：**输入层，LSTM层，输出层，训练误差，loss，optimizer

- **模型训练：**设置模型参数对模型进行训练

- **生成新文本：**训练新的文本

主题：整个文本将基于《安娜卡列妮娜》这本书的英文文本作为LSTM模型的训练数据，输入为单个字符，通过学习整个英文文档的字符（包括字母和标点符号等）来进行文本生成。在开始建模之前，我们首先要明确我们的输入和输出。即输入是字符，输出是预测出的新字符。

一. 数据预处理

在开始模型之前，我们首先要导入需要的包：

```
import time
import numpy as np
import tensorflow as tf
```

这一部分主要包括了数据的转换与mini-batch的分割步骤。

首先我们来进行数据的加载与编码转换。由于我们是基于**字符**（字母和标点符号等单个字符串，以下统称为**字符**）进行模型构建，也就是说我们的输入和输出都是字符。举个栗子，假如我们有一个

单词“hello”，我们想要基于这个单词构建LSTM，那么希望的到的结果是，输入“h”，预测下一个字母为“e”；输入“e”时，预测下一个字母为“l”，等等。

因此我们的输入便是一个个字母，下面我们将文章进行转换。

```
# 加载数据
with open('anna.txt', 'r') as f:
    text = f.read()

# 构建字符集合
vocab = set(text)
# 字符-数字映射字典
vocab_to_int = {c: i for i, c in enumerate(vocab)}
# 数字-字符映射字典
int_to_vocab = dict(enumerate(vocab))

# 对文本进行转码
encoded = np.array([vocab_to_int[c] for c in text], dtype=np.int32)
```

上面的代码主要完成了下面三个任务：

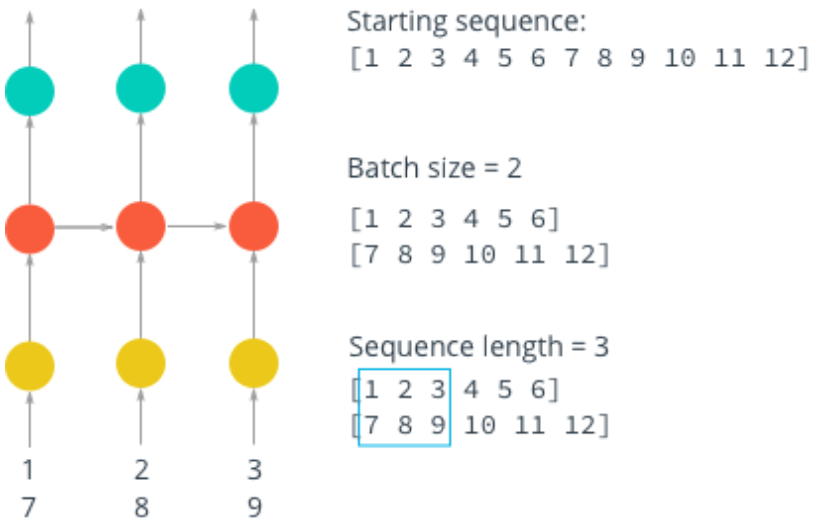
- 得到了文章中所有的字符集合 vocab

- 得到一个字符-数字的映射 vocab_to_int

- 得到一个数字-字符的映射 int_to_vocab

- 对原文进行转码后的列表 encoded

完成了前面的数据预处理操作，接下来就是要划分我们的数据集，在这里我们使用mini-batch来进行模型训练，那么我们要如何划分数据集呢？在进行mini-batch划分之前，我们先来了解几个概念。



假如我们目前手里有一个序列1-12，我们接下来以这个序列为例来说明划分mini-batch中的几个概念。首先我们回顾一下，在DNN和CNN中，我们都会将数据分batch输入给神经网络，加入我们有100个样本，如果设置我们的batch_size=10，那么意味着每次我们都会向神经网络输入10个样本

进行训练调整参数。同样的，在LSTM中，batch_size意味着每次向网络输入多少个样本，在上图中，当我们设置batch_size=2时，我们会将整个序列划分为6个batch，每个batch中有两个数字。

然而由于RNN中存在着“记忆”，也就是循环。事实上一个循环神经网络能够被看做是多个相同神经网络的叠加，在这个系统中，每一个网络都会传递信息给下一个。上面的图中，我们可以看到整个RNN网络由三个相同的神经网络单元叠加起来的序列。那么在这里就有了第二个概念sequence_length（也叫steps），中文叫序列长度。上图中序列长度是3，可以看到将三个字符作为了一个序列。

有了上面两个概念，我们来规范一下后面的定义。我们定义一个batch中的序列个数为N（即batch_size），定义单个序列长度为M（也就是我们的num_steps）。那么实际上我们每个batch是一个 $N \times M$ 的数组，**相当于我们的每个batch中有 $N \times M$ 个字符。**在上图中，当我们设置N=2， M=3时，我们可以得到每个batch的大小为2 x 3 = 6个字符，整个序列可以被分割成12 / 6 = 2个batch。

基于上面的分析，我们下面来进行mini-batch的分割：

```
def get_batches(arr, n_seqs, n_steps):
    """
    对已有的数组进行mini-batch分割

    arr: 待分割的数组
    n_seqs: 一个batch中的序列个数
    n_steps: 单个序列长度
    """
    batch_size = n_seqs * n_steps
    n_batches = int(len(arr) / batch_size)

    # 这个地方我们仅保留完成的batch(full batch)，也就是说对于不能整除的部分进行舍弃
    arr = arr[: batch_size * n_batches]

    # 重塑
    arr = arr.reshape((n_seqs, -1))

    for n in range(0, arr.shape[1], n_steps):
        x = arr[: n:n+n_steps]
        # 注意targets相比于x会向后错位一个字符
        y = np.zeros_like(x)
        y[:, :-1], y[:, -1] = x[:, 1:], y[:, 0]
        yield x, y
```

上面的代码定义了一个generator，调用函数会返回一个generator对象，我们可以获取一个batch。

经过上面的步骤，我们已经完成了对数据集的预处理。下一步我们开始构建模型。

二. 模型构建

模型构建部分主要包括了输入层，LSTM层，输出层，loss，optimizer等部分的构建，我们将一块一块来进行实现。

1.输入层

在数据预处理阶段，我们定义了mini-batch的分割函数，输入层的size取决于我们设置batch的size（$n\_seqs \times n\_steps$），下面我们首先构建输入层。

```
def build_inputs(num_seqs, num_steps):
    '''
    构建输入层

    num_seqs: 每个batch中的序列个数
    num_steps: 每个序列包含的字符数
    '''
    inputs = tf.placeholder(tf.int32, shape=(num_seqs, num_steps), name='inputs')
    targets = tf.placeholder(tf.int32, shape=(num_seqs, num_steps), name='targets')

    # 加入keep_prob
    keep_prob = tf.placeholder(tf.float32, name='keep_prob')

    return inputs, targets, keep_prob
```

同样的，输出层的 $shape = N \times M$ （因为输入一个字符，同样会输出一个字符）。除了输入输出外，我们还定义了keep_prob参数用来在后面控制dropout的保留结点数。关于dropout正则化请参考链接。

2.LSTM层

LSTM层是整个神经网络的关键部分。TensorFlow中，tf.contrib.rnn模块中有BasicLSTMCell和LSTMCell两个包，它们的区别在于：

> BasicLSTMCell does not allow cell clipping, a projection layer, and does not use peep-hole connections: it is the basic baseline.（来自TensorFlow官网）

在这里我们仅使用基本模块BasicLSTMCell。

```python
def build_lstm(lstm_size, num_layers, batch_size, keep_prob):
    """
    构建lstm层

    lstm_size: lstm cell中隐层结点数目
    num_layers: lstm层的数目
    batch_size: num_seqs x num_steps
    keep_prob
    """
    # 构建一个基本lstm单元
    lstm = tf.nn.run_cell.BasicLSTMCell(lstm_size)

    # 添加dropout
    drop = tf.nn.rnn_cell.DropoutWrapper(lstm, output_keep_prob=keep_prob)

    # 堆叠多个LSTM单元
    cell = tf.nn.rnn_cell.MultiRNNCell([drop for _ in range(num_layers)])
    initial_state = cell.zero_state(batch_size, tf.float32)

    return cell, initial_state
```

上面的代码中，我并没有使用tf.contrib.rnn模块，是因为我在使用远程floyd的GPU运行代码时候告诉我找不到这个模块，可以用tf.nn.run_cell.BasicLSTMCell替代。构建好LSTM cell后，为了防止过拟合，在它的隐层添加了dropout正则。

后面的MultiRNNCell实现了对基本LSTM cell的顺序堆叠，它接收的是cell对象组成的list。最后initial_state定义了初始cell state。

3.输出层

到目前为止，我们的输入和LSTM层都已经构建完毕。接下来就要构造我们的输出层，输出层采用softmax，它与LSTM进行全连接。对于每一个字符来说，它经过LSTM后的输出大小是 $1 \times L$ （L为LSTM cell隐层的结点数量），我们上面也分析过输入一个N x M的batch，我们从LSTM层得到的输出为 $N \times M \times L$ ，要将这个输出与softmax全连接层建立连接，就需要对LSTM的输出进行重塑，变成 $(N * M) \times L$ 的一个2D的tensor。softmax层的结点数应该是vocab的大小（我们要计算概率分布）。因此整个LSTM层到softmax层的大小为 $L \times vocab\_size$ 。

```python
def build_output(lstm_output, in_size, out_size):
    """
    构建输出层

    lstm_output: LSTM层的输出结果（是一个三维数组）
    in_size: LSTM层重塑后的size
    out_size: softmax层的size
    """
    # 将lstm的输出按照列concate，例如[[1,2,3],[7,8,9]],
    # tf.concat的结果是[1,2,3,7,8,9]
    seq_output = tf.concat(lstm_output, 1)
    # reshape
    x = tf.reshape(seq_output, [-1, in_size])

    # 连接LSTM输入到softmax layer
    with tf.variable_scope('softmax'):
        softmax_w = tf.Variable(tf.truncated_normal([in_size, out_size], stddev=0.1))
        softmax_b = tf.Variable(tf.zeros(out_size))

    # 计算logits
    logits = tf.matmul(x, softmax_w) + softmax_b

    # softmax层返回概率分布
    out = tf.nn.softmax(logits, name='predictions')

    return out, logits
```

将数据重塑后，我们对LSTM层和softmax层进行连接。并计算logits和softmax后的概率分布。

4.训练误差计算

至此我们已经完成了整个网络的构建，接下来要定义train loss和optimizer。我们知道从sotfmax层输出的是概率分布，因此我们要对targets进行one-hot编码。我们采用softmax_cross_entropy_with_logits交叉熵来计算loss。

```python
def build_loss(logits, targets, lstm_size, num_classes):
    """
    根据logits和targets计算损失

    logits: 全连接层输出的结果(没有经过softmax)
    targets: 目标字符
    lstm_size: LSTM cell隐层结点的数量
    num_classes: vocab_size
    """
    # 对targets进行编码
    y_one_hot = tf.one_hot(targets, num_classes)
    y_reshaped = tf.reshape(y_one_hot, logits.get_shape())

    # softmax cross entropy between logits and labels
    loss = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y_reshaped)
    loss = tf.reduce_mean(loss)

    return loss
```

5.Optimizer

我们知道RNN会遇到**梯度爆炸（gradients exploding）**和**梯度弥散（gradients disappearing)**的问题。LSTM解决了梯度弥散的问题，但是gradients仍然可能会爆炸，因此我们采用gradient clippling的方式来防止梯度爆炸。即通过设置一个阈值，当gradients超过这个阈值时，就将它重置为阈值大小，这就保证了梯度不会变得很大。

```
def build_optimizer(loss, learning_rate, grad_clip):
    '''
    构造Optimizer

    loss: 损失
    learning_rate: 学习率

    '''

    # 使用clipping gradients
    tvars = tf.trainable_variables()
    grads, _ = tf.clip_by_global_norm(tf.gradients(loss, tvars), grad_clip)
    train_op = tf.train.AdamOptimizer(learning_rate)
    optimizer = train_op.apply_gradients(zip(grads, tvars))

    return optimizer
```

tf.clip_by_global_norm会返回clip以后的gradients以及global_norm。整个学习过程采用AdamOptimizer

6.模型组合

经过上面五个步骤，我们完成了所有的模块设置。下面我们来将这些部分组合起来，构建一个类。

```
class CharRNN:

    def __init__(self, num_classes, batch_size=64, num_steps=50,
                       lstm_size=128, num_layers=2, learning_rate=0.001,
                       grad_clip=5, sampling=False):

        # 如果sampling是True，则采用SGD
        if sampling == True:
            batch_size, num_steps = 1, 1
        else:
            batch_size, num_steps = batch_size, num_steps

        tf.reset_default_graph()

        # 输入层
        self.inputs, self.targets, self.keep_prob = build_inputs(batch_size, num_steps)

        # LSTM层
        cell, self.initial_state = build_lstm(lstm_size, num_layers, batch_size, self.keep_prob)

        # 对输入进行one-hot编码
        x_one_hot = tf.one_hot(self.inputs, num_classes)

        # 运行RNN
        outputs, state = tf.nn.dynamic_rnn(cell, x_one_hot, initial_state=self.initial_state)
        self.final_state = state

        # 预测结果
        self.prediction, self.logits = build_output(outputs, lstm_size, num_classes)

        # Loss 和 optimizer (with gradient clipping)
        self.loss = build_loss(self.logits, self.targets, lstm_size, num_classes)
        self.optimizer = build_optimizer(self.loss, learning_rate, grad_clip)
```

我们使用tf.nn.dynamic_run来运行RNN序列。

三. 模型训练

在模型训练之前，我们首先初始化一些参数，我们的参数主要有：

batch_size: 单个batch中序列的个数
num_steps: 单个序列中字符数目
lstm_size: 隐层结点个数
num_layers: LSTM层个数
learning_rate: 学习率
keep_prob: 训练时dropout层中保留结点比例

```
batch_size = 100          # Sequences per batch
num_steps = 100           # Number of sequence steps per batch
lstm_size = 512           # Size of hidden layers in LSTMs
num_layers = 2            # Number of LSTM layers
learning_rate = 0.001     # Learning rate
keep_prob = 0.5           # Dropout keep probability
```

这是我自己设置的一些参数，具体一些调参经验可以参考Andrej Karpathy的git上的建议。

参数设置完毕后，离运行整个LSTM就差一步啦，下面我们来运行整个模型。

```
epochs = 20
# 每n轮进行一次变量保存
save_every_n = 200

model = CharRNN(len(vocab), batch_size=batch_size, num_steps=num_steps,
                lstm_size=lstm_size, num_layers=num_layers,
                learning_rate=learning_rate)

saver = tf.train.Saver(max_to_keep=100)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    counter = 0
    for e in range(epochs):
        # Train network
        new_state = sess.run(model.initial_state)
        loss = 0
        for x, y in get_batches(encoded, batch_size, num_steps):
            counter += 1
            start = time.time()
            feed = {model.inputs: x,
                    model.targets: y,
                    model.keep_prob: keep_prob,
                    model.initial_state: new_state}
            batch_loss, new_state, _ = sess.run([model.loss,
                                                 model.final_state,
                                                 model.optimizer],
                                                 feed_dict=feed)

            end = time.time()
            # control the print lines
            if counter % 500 == 0:
                print('轮数: {}/{}... '.format(e+1, epochs),
                      '训练步数: {}... '.format(counter),
                      '训练误差: {:.4f}... '.format(batch_loss),
                      '{:.4f} sec/batch'.format((end-start)))

            if (counter % save_every_n == 0):
                saver.save(sess, "checkpoints/i{}_l{}.ckpt".format(counter, lstm_size))

    saver.save(sess, "checkpoints/i{}_l{}.ckpt".format(counter, lstm_size))
```

我这里设置的迭代次数为20次，并且在代码运行中我们设置了结点的保存，设置了每运行200次进行一次变量保存，这样的好处是有利于我们后面去直观地观察在整个训练过程中文本生成的结果是如何一步步“进化”的。

四.文本生成

经过漫长的模型训练，我们得到了一系列训练过程中保存下来的参数，可以利用这些参数来进行文本生成啦。当我们输入一个字符时，它会预测下一个，我们再将这个新的字符输入模型，就可以一直不断地生成字符，从而形成文本。

为了减少噪音，每次的预测值我会选择最可能的前5个进行随机选择，比如输入h，预测结果概率最大的前五个为 $[o, e, i, u, b]$ ，我们将随机从这五个中挑选一个作为新的字符，让过程加入随机因素会减少一些噪音的生成。

代码封装了两个函数来做文本生成，具体请参看文章尾部的git链接中的源码。

训练步数：200

当训练步数为200的时候，LSTM生成的文本大概长下面这个样子：

```
Farsd othas sead ansisg he therinn wen an time he has tas hers wand so the her ansong astens
 hit
ang arideret to hess the
cand the wan ou he and waste sad that had whe hh simorsersond torith we sotered.

"Whot simte wet ant ans wit hise afde sit hat her touten th merint he tor he chins.

"ho se ar her ore tir that hes hor
she there tothe werilth se ses ororing, thas te that ant he ser ot hit he corlithit."
"Sele, Ir anthe wast to han sist hos the coseling thet tir sassesesin thon the tan sheres an
 tor the ceras tat aless the
hans, wins of the hite ho tar so at ane an to tont his the whers and
ther herasdith shere th th woul ter hor
the he wothe hires te sel hin hind and the he sotor has has andeses onet toren his of athand
 this thar sost the soud
ale ar on hot orin at the her ses on the sesed tor het
an ase thing, at hins hat had hhe was shares thin and his ofe the hor th the sised ther
alind tor here tor ther thar

one tanes, she thes sotithe with toused tat her aret al hans
on he he sor
```

看起来像是字符的随机组合，但是可以看到有一些单词例如hat，her等已经出现，并且生成了成对的引号。

训练步数：1000

```
Farrer, as his positions
the child of happeness and
she wither she wan sainted at the petsaot, that his expression of his had, but he wanted to a
n the drien that whome he haved to the pertion with the count and that the perpleation of the
 mungrait of the cander were sate along the same at
her all this he heard the
stallers, and that she saw she same and shade all with him he had not said to stelp only with
 the sorm a misters
one when he could not he
stought."

"Yes, and
you sand to ser anster and som that that?
Ind to be a candies, and shilly see that with the down of the semilial all or
taking,
and thought he said staid at the clate offen as had almoment to be anouted happy... The prest
ing he deen the conder and had
not so mo that he can't an in a such walked in it a pruce to him.

"I say that have been the seming of the posstor," talking and her thought one
standing his explaning of a that this something worls. So he was stept, and that that when sh
e had been that he wonder to be the ser
```

当训练步数到达1000的时候，已经开始有简单的句子出现，并且单词看起来似乎不是那么乱了。

训练步数：2000

```
Fardievna, and had as he children his
hand.

"And his bight me that's she do it's being became string."

"Oh, that, you know where your home is never mented and to do?"

"I could have seen her house and a minute and the socies, that's not as
it's nothing to see in that arest...." And the count of the
sers of tense.

"Well that is she something was that's, the same impersaite with you."

"I'll delighted her?" he added.

"Yes, I can't speak of him only a place of the made in the
raise.

The compitions was in she said if, and still a group of the prince. "That's the higher of
the pashed or to her. What as seemed is in thes though, in a cerrain with
him. His fingers of the middare of the mare they and she had
spenk a land."

"Alexey Alexandrovitch who wrote a going, in her mother, in the
prace was to ble and they always begin to talk and say to
him, and would be the secarisial missed to breather. I'm said: "It's
nothing as a look to hall that I shall go and will see that sick of
him the heart
```

当训练步数达到2000的时候，单词和句子看起来已经有所规范。

训练步数：3960

```
Ther woold have saw that it was a second of a little.

"The princess went a presence out of the music. Then he'd to be dropped. And
I'm not saigh."

"Well, and he was not a man. That's which I have been troubled to supposing
her transformed, and his woman, that here is not a shartes of hesitation."

"Well, that you will not see you all to the delight of all he he didn't
believe it."

"What do you think? I shall see her, and he has not asked it in, and I could
call me a little bride."

"What a man of the same in that, I'll go to him," said Alexey
Alexandrovitch, smiling to himself, "they will be angry, to say. The
contrary is an inside. It's nine thinging to him in argive, but I
don't want to tell you, and I didn't come," he said, and he found to
get the presence of side, he was not too stopped in her eyes.

An halves, both, though these pares, he heard the seat of the charmage, watching
the pather, and he was seen in the poor the things of a peculiar love,
his feech one sigher she had not cared to see her fine, but talking of
the steps to the still sense of himself to say. But he deding that
how should be about the country with him, to be darted; and she was not
so and he had a game and sort of letter on his feilling on their
most fingress as all thit in the classes of the social thoughts, and that
he was set first in the corner of the crops to them, and an instance he
had a good hair so something brightly tore, and he was sitting to his
hands, and that she were silent the comforted peasant. They were
taken a confusion, but he was since he would have say how it was natural
to bad a beginning about him. The solies of the same simply. But the
carder he was assamed.

"There's a time was so talking at this feeling in him. Would not be a
charme of my pace and will see. When they were to go and the points were
surposed to how I have as a sort of sort for how I can't submerter, and I was
that all at the simplest of it too any sort of most, as
tried to be to see your children.
```

当训练结束时（本文仅训练了3960步），生成的文本已经有小部分可以读的比较通顺了，而且很少有单词拼写的错误。

五.总结

整个文章通过构建LSTM模型完成了对《安娜卡列宁娜》文本的学习并且基于学习成果生成了新的文本。

通过观察上面的生成文本，我们可以看出随着训练步数的增加，模型的训练误差在持续减少。本文仅设置了20次迭代，尝试更大次数的迭代可能会取得更好的效果。

个人觉得LSTM对于文本的学习能力还是很强，后面可能将针对中文文本构造一些学习模型，应该会更有意思！

我对RNN也是在不断地探索与学习中，文中不免会有一些错误和谬误，恳请各位指正，非常感谢！