TensorFlow中RNN实现的正确打开方式

何之源
深度学习（Deep Learning） 话题的优秀回答者

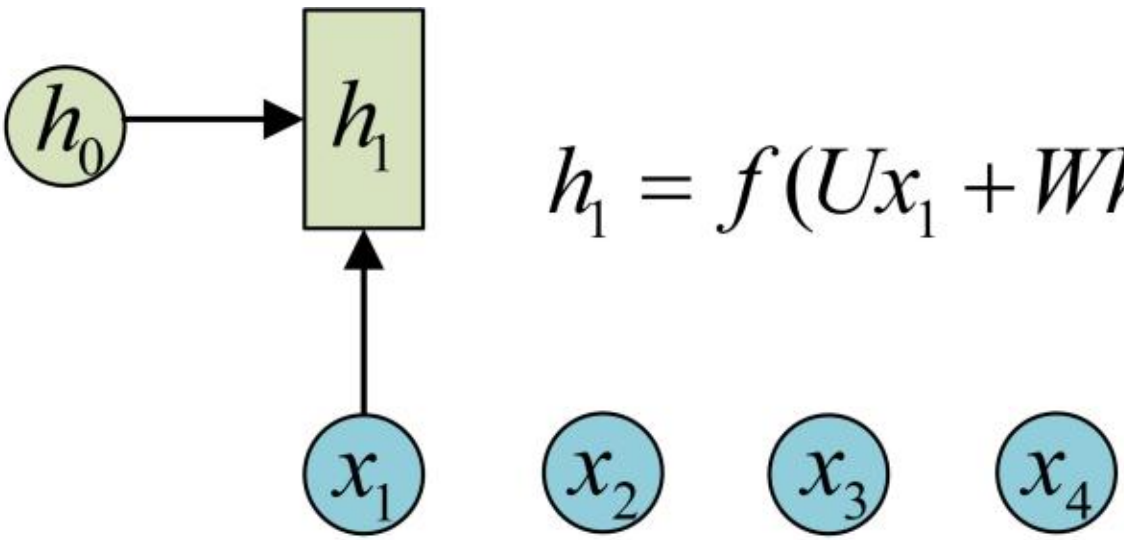
上周写了一篇文章介绍了一下RNN的几种结构，今天就来聊一聊如何在TensorFlow中实现这些结构，这篇文章的主要内容为：

一个完整的、循序渐进的学习TensorFlow中RNN实现的方法。这个学习路径的曲线较为平缓，应该可以减少不少学习精力，帮助大家少走弯路。
一些可能会踩的坑
TensorFlow源码分析
一个Char RNN实现示例，可以用来写诗，生成歌词，甚至可以用来写网络小说！（项目地址：hzy46/Char-RNN-TensorFlow）

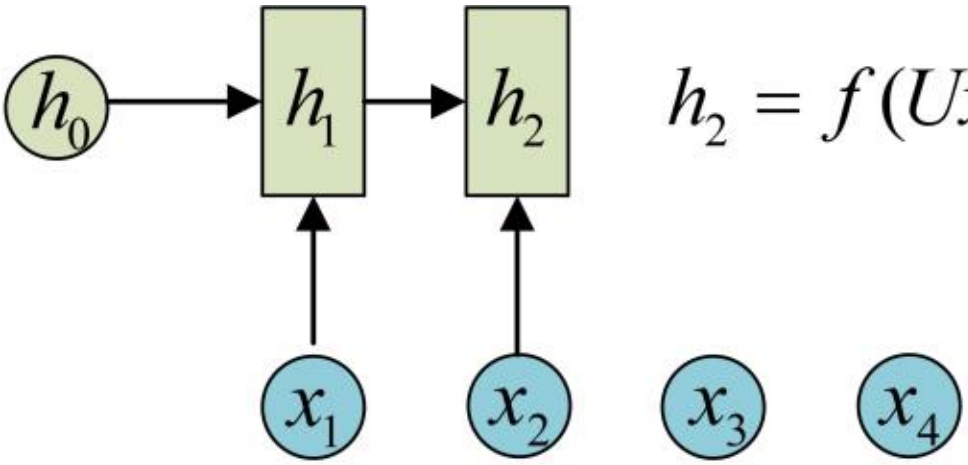
一、学习单步的RNN：RNNCell

如果要学习TensorFlow中的RNN，第一站应该就是去了解“RNNCell”，它是TensorFlow中实现RNN的基本单元，每个RNNCell都有一个call方法，使用方式是：(output, next_state) = call(input, state)。

借助图片来说可能更容易理解。假设我们有一个初始状态h0，还有输入x1，调用call(x1, h0)后就可以得到(output1, h1)：



再调用一次call(x2, h1)就可以得到(output2, h2)：



也就是说，**每调用一次RNNCell的call方法，就相当于在时间上“推进了一步”，这就是RNNCell的基本功能。**

在代码实现上，RNNCell只是一个抽象类，我们用的时候都是用的它的两个子类BasicRNNCell和BasicLSTMCell。顾名思义，前者是RNN的基础类，后者是LSTM的基础类。这里推荐大家阅读其源码实现，一开始并不需要全部看一遍，只需要看下RNNCell、BasicRNNCell、BasicLSTMCell这三个类的注释部分，应该就可以理解它们的功能了。

除了call方法外，对于RNNCell，还有两个类属性比较重要：

state_size
output_size

前者是隐层的大小，后者是输出的大小。比如我们通常是将一个batch送入模型计算，设输入数据的形状为(batch_size, input_size)，那么计算时得到的隐层状态就是(batch_size, state_size)，输出就是(batch_size, output_size)。

可以用下面的代码验证一下（注意，以下代码都基于TensorFlow最新的1.2版本）：

```
import tensorflow as tf
import numpy as np

cell = tf.nn.rnn_cell.BasicRNNCell(num_units=128) # state_size = 128
print(cell.state_size) # 128

inputs = tf.placeholder(np.float32, shape=(32, 100)) # 32 是 batch_size
h0 = cell.zero_state(32, np.float32) # 通过zero_state得到一个全0的初始状态，形状为(b
output, h1 = cell.call(inputs, h0) #调用call函数

print(h1.shape) # (32, 128)
```

对于BasicLSTMCell，情况有些许不同，因为LSTM可以看做有两个隐状态h和c，对应的隐层就是一个Tuple，每个都是(batch_size, state_size)的形状：

```
import tensorflow as tf
import numpy as np
lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units=128)
inputs = tf.placeholder(np.float32, shape=(32, 100)) # 32 是 batch_size
h0 = lstm_cell.zero_state(32, np.float32) # 通过zero_state得到一个全0的初始状态
output, h1 = lstm_cell.call(inputs, h0)

print(h1.h)  # shape=(32, 128)
print(h1.c)  # shape=(32, 128)
```

二、学习如何一次执行多步：tf.nn.dynamic_rnn

基础的RNNCell有一个很明显的问题：对于单个的RNNCell，我们使用它的call函数进行运算时，只是在序列时间上前进了一步。比如使用x1、h0得到h1，通过x2、h1得到h2等。**这样的h话，如果我们的序列长度为10，就要调用10次call函数，比较麻烦。对此，TensorFlow提供了一个tf.nn.dynamic_rnn函数，使用该函数就相当于调用了n次call函数。**即通过{h0,x1, x2, ..., xn}直接得{h1,h2...,hn}。

具体来说，设我们输入数据的格式为(batch_size, time_steps, input_size)，其中time_steps表示序列本身的长度，如在Char RNN中，长度为10的句子对应的time_steps就等于10。最后的input_size就表示输入数据单个序列单个时间维度上固有的长度。另外我们已经定义好了一个RNNCell，调用该RNNCell的call函数time_steps次，对应的代码就是：

```
# inputs: shape = (batch_size, time_steps, input_size)
# cell: RNNCell
# initial_state: shape = (batch_size, cell.state_size)。初始状态。一般可以取零矩阵
outputs, state = tf.nn.dynamic_rnn(cell, inputs, initial_state=initial_state)
```

此时，得到的outputs就是time_steps步里所有的输出。它的形状为(batch_size, time_steps, cell.output_size)。state是最后一步的隐状态，它的形状为(batch_size, cell.state_size)。

此处建议大家阅读tf.nn.dynamic_rnn的文档做进一步了解。

三、学习如何堆叠RNNCell：MultiRNNCell

很多时候，单层RNN的能力有限，我们需要多层的RNN。将x输入第一层RNN的后得到隐层状态h，这个隐层状态就相当于第二层RNN的输入，第二层RNN的隐层状态又相当于第三层RNN的输

入，以此类推。在TensorFlow中，可以使用tf.nn.rnn_cell.MultiRNNCell函数对RNNCell进行堆叠，相应的示例程序如下：

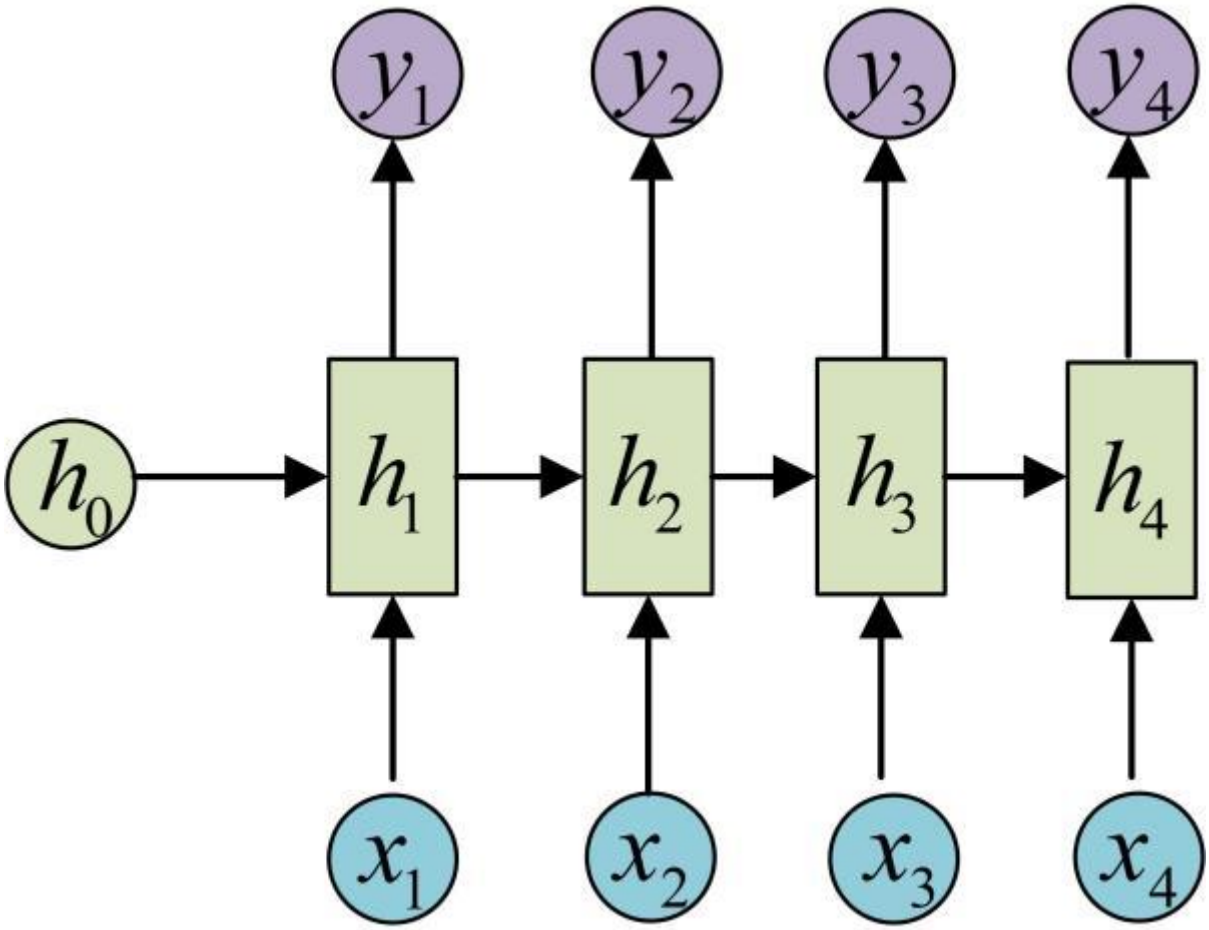
```
import tensorflow as tf
import numpy as np

# 每调用一次这个函数就返回一个BasicRNNCell
def get_a_cell():
    return tf.nn.rnn_cell.BasicRNNCell(num_units=128)
# 用tf.nn.rnn_cell MultiRNNCell创建3层RNN
cell = tf.nn.rnn_cell.MultiRNNCell([get_a_cell() for _ in range(3)]) # 3层RNN
# 得到的cell实际也是RNNCell的子类
# 它的state_size是(128, 128, 128)
# (128, 128, 128)并不是128x128x128的意思
# 而是表示共有3个隐层状态，每个隐层状态的大小为128
print(cell.state_size) # (128, 128, 128)
# 使用对应的call函数
inputs = tf.placeholder(np.float32, shape=(32, 100)) # 32 是 batch_size
h0 = cell.zero_state(32, np.float32) # 通过zero_state得到一个全0的初始状态
output, h1 = cell.call(inputs, h0)
print(h1) # tuple中含有3个32x128的向量
```

通过MultiRNNCell得到的cell并不是什么新鲜事物，它实际也是RNNCell的子类，因此也有call方法、state_size和output_size属性。同样可以通过tf.nn.dynamic_rnn来一次运行多步。

此处建议阅读MutiRNNCell源码中的注释进一步了解其功能。

四、可能遇到的坑1：Output说明

在经典RNN结构中有这样的图：



在上面的代码中，我们好像有意忽略了调用call或dynamic_rnn函数后得到的output的介绍。将上图与TensorFlow的BasicRNNCell对照来看。h就对应了BasicRNNCell的state_size。那么，y是不是就对应了BasicRNNCell的output_size呢？**答案是否定的**。

找到源码中BasicRNNCell的call函数实现：

```
def call(self, inputs, state):
    """Most basic RNN: output = new_state = act(W * input + U * state + B)."""
    output = self._activation(_linear([inputs, state], self._num_units, True))
    return output, output
```

这句“return output, output”说明在BasicRNNCell中，output其实和隐状态的值是一样的。因此，我们还需要额外对输出定义新的变换，才能得到图中真正的输出y。由于output和隐状态是一回事，所以在BasicRNNCell中，state_size永远等于output_size。TensorFlow是出于尽量精简的目的来定义BasicRNNCell的，所以省略了输出参数，我们这里一定要弄清楚它和图中原始RNN定义的联系与区别。

再来看一下BasicLSTMCell的call函数定义（函数的最后几行）：

```
new_c = (
    c * sigmoid(f + self._forget_bias) + sigmoid(i) * self._activation(j))
new_h = self._activation(new_c) * sigmoid(o)

if self._state_is_tuple:
  new_state = LSTMStateTuple(new_c, new_h)
else:
  new_state = array_ops.concat([new_c, new_h], 1)
return new_h, new_state
```

我们只需要关注self._state_is_tuple == True的情况，因为self._state_is_tuple == False的情况将在未来被弃用。返回的隐状态是new_c和new_h的组合，而output就是单独的new_h。如果我们处理的是分类问题，那么我们还需要对new_h添加单独的Softmax层才能得到最后的分类概率输出。

还是建议大家亲自看一下源码实现来搞明白其中的细节。

五、可能遇到的坑2：因版本原因引起的错误

在前面我们讲到堆叠RNN时，使用的代码是：

```
# 每调用一次这个函数就返回一个BasicRNNCell
def get_a_cell():
    return tf.nn.rnn_cell.BasicRNNCell(num_units=128)
# 用tf.nn.rnn_cell MultiRNNCell创建3层RNN
cell = tf.nn.rnn_cell.MultiRNNCell([get_a_cell() for _ in range(3)]) # 3层RNN
```

这个代码在TensorFlow 1.2中是可以正确使用的。但在之前的版本中（以及网上很多相关教程），实现方式是这样的：

```
one_cell =  tf.nn.rnn_cell.BasicRNNCell(num_units=128)
cell = tf.nn.rnn_cell.MultiRNNCell([one_cell] * 3) # 3层RNN
```

如果在TensorFlow 1.2中还按照原来的方式定义，就会引起错误！

六、一个练手项目：Char RNN

上面的内容实际上就是TensorFlow中实现RNN的基本知识了。**这个时候，建议大家用一个项目来练习巩固一下。此处特别推荐Char RNN项目，这个项目对应的是经典的RNN结构，实现它使用的**

TensorFlow函数就是上面说到的几个，项目本身又比较有趣，可以用来做文本生成，平常大家看到的用深度学习来写诗写歌词的基本用的就是它了。

Char RNN的实现已经有很多了，可以自己去Github上面找，我这里也做了一个实现，供大家参考。项目地址为：hzy46/Char-RNN-TensorFlow。代码的部分实现来自于这篇专栏，在此感谢

@天雨粟 。

我主要向代码中添加了embedding层，以支持中文，另外重新整理了代码结构，将API改成了最新的TensorFlow 1.2版本。

可以用这个项目来写诗（以下诗句都是自动生成的）：

```
何人无不见，此地自何如。
一夜山边去，江山一夜归。
山风春草色，秋水夜声深。
何事同相见，应知旧子人。
何当不相见，何处见江边。
一叶生云里，春风出竹堂。
何时有相访，不得在君心。
```

还可以生成代码：

```
static int page_cpus(struct flags *str)
{
        int rc;
        struct rq *do_init;
};

/*
 * Core_trace_periods the time in is is that supsed,
 */
#endif

/*
 * Intendifint to state anded.
 */
int print_init(struct priority *rt)
{       /* Comment sighind if see task so and the sections */
        console(string, &can);
}
```

此外生成英文更不是问题（使用莎士比亚的文本训练）：

```
LAUNCE:
The formity so mistalied on his, thou hast she was
to her hears, what we shall be that say a soun man
Would the lord and all a fouls and too, the say,
That we destent and here with my peace.

PALINA:
Why, are the must thou art breath or thy saming,
I have sate it him with too to have me of
I the camples.
```

最后，如果你脑洞够大，还可以来做一些更有意思的事情，比如我用了著名的网络小说《斗破苍穹》训练了一个RNN模型，可以生成下面的文本：

```
闻言，萧炎一怔，旋即目光转向一旁的那名灰袍青年，然后目光在那位老者身上扫过，那里，一个巨大的石

“这是一位斗尊阶别，不过不管你，也不可能会出手，那些家伙，可以为了这里，这里也是能够有着一些异常

“这里的人，也是能够与魂殿强者抗衡。”

萧炎眼眸中也是掠过一抹惊骇，旋即一笑，旋即一声冷喝，身后那些魂殿殿主便是对于萧炎，一道冷喝的身

“嗤！”
```

还是挺好玩的吧，另外还尝试了生成日文等等。

七、学习完整版的LSTMCell

上面只说了基础版的BasicRNNCell和BasicLSTMCell。TensorFlow中还有一个“完全体”的LSTM：LSTMCell。这个完整版的LSTM可以定义peephole，添加输出的投影层，以及给LSTM的遗忘单元设置bias等，可以参考其源码了解使用方法。

八、学习最新的Seq2Seq API

Google在TensorFlow的1.2版本（1.3.0的rc版已经出了，貌似正式版也要出了，更新真是快）中更新了Seq2Seq API，使用这个API我们可以不用手动地去定义Seq2Seq模型中的Encoder和Decoder。此外它还和1.2版本中的新数据读入方式Datasets兼容。可以阅读此处的文档学习它的使用方法。

九、总结

最后简单地总结一下，这篇文章提供了一个学习TensorFlow RNN实现的详细路径，其中包括了学习顺序、可能会踩的坑、源码分析以及一个示例项目hzy46/Char-RNN-TensorFlow，希望能对大家有所帮助。