用 RNN 网络模型写唐诗

Linyang He

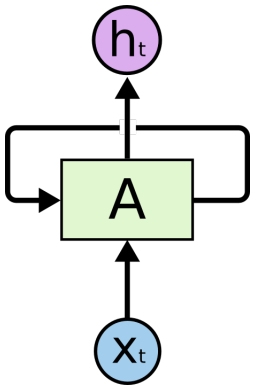
2018 年 11 月 17 日

本文以写唐诗这个情景为例，重点关注了 RNN 模型（以及 LSTM,GRU 模型）在时间序列问题上的应用。本文先会讲述各个 RNN 模型，再描述了唐诗生成的过程。接着展示了以“日、红、山、夜、湖、海、月”等字为首字的诗歌。最后是实验的总结，并提出了一些自己的思考。

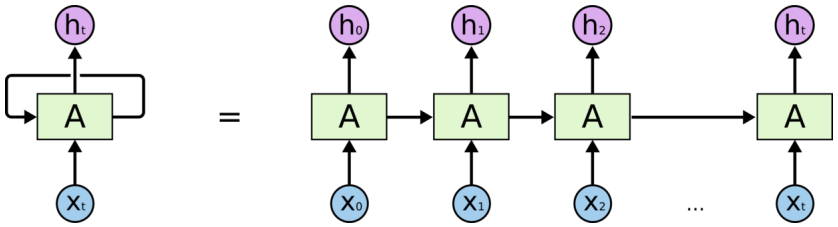
1 Background

1.1 RNN

当我们在理解一句话意思时，孤立的理解这句话的每个词是不够的，我们需要处理这些词连接起来的整个序列。为了解决一些这样类似的问题，能够更好的处理序列的信息，RNN 就诞生了。递归神经网络（Recurrent Neural Networks，RNN）对于时序信息的处理有着天然的优异性。RNN 的特点在于，会把上个时序的 hidden_state 信息作为这个时序的输入，这样，就实现了不同时序上的信息的沟通。下图是一个单独的 RNN Unit 示意图。



可以发现，隐藏层中间有个 w 在循环的输入输出，这便是 hidden_state。如果展开来看，可以得到如下图像，这也是平时我们常见的 RNN 图像。



具体而言，假设在时刻 t 时，网络的输入为 x_t，隐层状态（即隐层神经元活性值）为 h_t 不仅和当前时刻的输入 x_t 相关，也和上一个时刻的隐层状态 h_{t-1} 相关。$f(\cdot)$ 是非线性激活函数，通常为 logistic 函数或 tanh 函数，U 为状态-状态权重矩阵，W 为状态-输入权重矩阵, 那么，RNN 的运算逻辑通常可以表示成：

$$h_t = f(Wx_t + Uh_{t-1})$$

. 总之，链式的特征揭示了 RNN 本质上是与序列和列表相关的。他们是对于这类数据的最自然的神经网络架构。

1.2 LSTM

RNN 的关键点之一就是他们可以用来连接先前的信息到当前的任务上，例如使用过去的视频段来推测对当前段的理解。如果 RNN 可以做到这个，他们就变得非常有用。但是真的可以么？答案是，还有很多依赖因素。有时候，我们仅仅需要知道先前的信息来执行当前的任务。例如，我们有一个语言模型用来基于先前的词来预测下一个词。如果我们试着预测“the clouds are in the sky”最后的词，我们并不需要任何其他的上下文——因此下一个词很显然就应该是 sky。在这样的场景中，相关的信息和预测的词位置之间的间隔是非常小的，RNN 可以学会使用先前的信息。

但是同样会有一些更加复杂的场景。假设我们试着去预测“I grew up in France... I speak fluent French”最后的词。当前的信息建议下一个词可能是一种语言的名字，但是如果我们需要弄清楚是什么语言，我们是需要先前提到的离当前位置很远的 France 的上下文的。这说明相关信息和当前预测位置之间的间隔就肯定变得相当的大。不幸的是，在这个间隔不断增大时，RNN 会丧失学习到连接如此远的信息的能力。在理论上，RNN 绝对可

以处理这样的长期依赖问题。人们可以仔细挑选参数来解决这类问题中的最初级形式，但在实践中，RNN 肯定不能够成功学习到这些知识。Bengio, et al. (1994) 等人对该问题进行了深入的研究，他们发现一些使训练 RNN 变得非常困难的相当根本的原因。然而，幸运的是，LSTM 并没有这个问题。

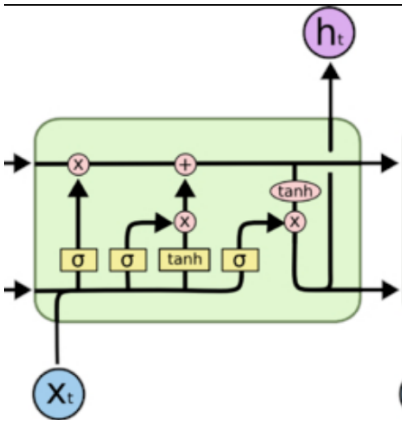
LSTM 是一种 RNN 特殊的类型，可以学习长期依赖信息。LSTM 通过刻意的设计来避免长期依赖问题。不同于普通 RNN 的单一神经网络层，LSTM unit 中有许多其他门控（包括遗忘门、记忆门等），以一种非常特殊的方式进行交互。这四个门控是：

1. Input gate (current cell matters)
2. Forget (gate 0, forget past)
3. Output (how much cell is exposed)
4. New memory cell

对应的四个门控的输出是：

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right) \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right) \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right) \\
\tilde{c}_t &= \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)
\end{aligned}
$$

此时，LSTM Unit 的示意图为下图，从左至右分别为 Input gate, forget, new memory cell, 以及 output。



因此，最后的两个输出分别为：

1. Final memory cell（位于上方的）
2. Final hidden state （位于下方和 h_t).

对应的公式分别为：

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tanh(c_t)$$

1.3 GRU

门控循环单元（Gated Recurrent Unit，GRU）网络是一种比 LSTM 网络更加简单的循环神经网络。与 LSTM 类似，都是在 RNN 上增加了不同的门控产生的高级的 RNN。在 LSTM 网络中，输入门和遗忘门是互补关系，用两个门比较冗余。GRU 将输入门与和遗忘门合并成一个门：更新门。同时，GRU 也不引入额外的记忆单元，直接在当前状态 h_t 和历史状态 h_{t-1} 之间引入线性依赖关系。GRU 有三个门：

1. Update gate
2. Reset Gate
3. New memory content.

三个门控分别对应的输出是：

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

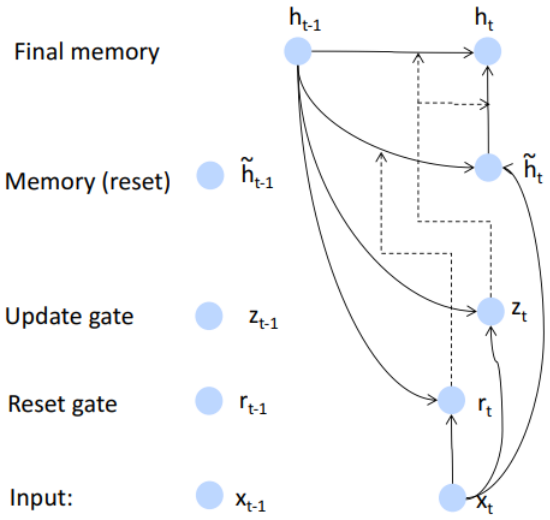
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$

这样，我们得到新的 h_t 是：

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

GRU 的图示：



2 Experiment

2.1 Language Model

语言模型是生成唐诗的重要背景知识。实际上，唐诗能写成，本质上就是利用 RNN 网络训练了一个基于唐诗语料库的语言模型。如果把语料库换成其他的歌词，也就可以写成其他风格的诗作了。这里简要介绍一下语言模型。统计语言模型是一个单词序列上的概率分布，对于一个给定长度为 m 的序列，它可以为整个序列产生一个概率 $P(w_1, w_2, ..., w_m)$ 。理论上我们需要根据一句话中所有的历史词汇来计算当前词汇的概率，但是这样计算太过复杂。通常，我们根据马尔可夫性假设当前词的概率与前面的 n 个词有关系。于是，语言模型：

$$P(w_1, w_2, ..., w_m) = \prod_{i=1}^{m} P(w_i|w_1, ..., w_{i-1}) = \prod_{i=1}^{m} P(w_i|w_{i-(n-1),...,w_{i-1}})$$

可以看到，我们实际中在应用统计语言模型的时候，只能根据前 n 个词汇来计算，过长的依赖就不能得到解决。而这也正是 RNN 神经语言模型能够解决的问题。只需要把上面提到的各种 RNN 模型的输入变成语料库中的 lexicon 序列，训练这个模型便可以得到神经语言模型了。

2.2 Peom Generation

本文使用的是 tensorflow 版本的代码，这里分析一下各个函数的作用，以此来说明唐诗生成的过程。

2.2.1 数据预处理部分

经过对函数 process_poems 的分析，我们可以得处三个返回值 poems_vector, word_int_map, words 分别为：存着诗（字符串）的 vector 索引集；每个字和索引的字典；诗歌语料库中出现的所有的字。

2.2.2 rnn_lstm model

这部分代码定义了我们的 RNN 模型，我们使用的是 MultiRNNCell，表示我们的模型是多层 RNN 堆叠而成的。同时，我们利用的是 dynamic_rnn 得到输出，说明有多个时序被同时计算，这样提升了模型的训练效率。

2.2.3 训练模型部分

这部分代码是清晰易懂的。唯一个值得关注的点是该程序将参数保存 checkpoint 中。如果训练被中断了，下次训练时可以直接从中断部分继续实验，极大地提升了效率。

2.2.4 生成诗歌部分

2.2.3 在代码上是训练模型，而生成诗歌则是验证模型。此外，要注意我们需要导入已经训练好的模型。注意如果需要训练好的参数能够成功导入，那么就要让网络模型等和训练的时候的一致。而具体到诗歌生成，原理其实很简单，此时，实际上利用的就是语言模型的思想，给出一个 begin_word，利用神经语言模型，预测出最可能的下一个 word，以此类推直到遇到了 end_token 为止，而这些预测出来的 word 组合起来也就是所谓的唐诗了。

2.2.5 主函数

将上述的代码组合起来，按照预处理、训练、验证的流程进行实现。

3 Result

本次实验中，我们训练了三个模型，情况如下，其中 trraning_loss 是最后一次迭代的值。

模型	learning_rate	epochs	training_loss	RNN
A	0.01	50	4.11	LSTM
B	0.001	112	3.73	LSTM
C	0.001	32	3.52	LSTM

生成的诗歌分别为：

A

```
[INFO] write tang poem
INFO:tensorflow:Restor
####################
日日临江上，东风吹玉尘。
风光摇碧落，风动玉楼风。
风起风吹落，风吹玉树枝。
风吹春色动，风起柳条新。
春色春风起，春风落日斜。
春风吹不发，春色满春风。
####################
INFO:tensorflow:Restor
####################
红叶萧萧下，青山白日斜。
风光生绿树，水色动清漪。
风起风吹落，风吹水色寒。
风光摇落日，风起水声清。
风起风吹湿，风吹落叶轻。
何当一杯酒，何况一相思。
####################
INFO:tensorflow:Restor
####################
山上山中见，山川不可寻。
山河入山寺，山色入云间。
石壁松阴下，松声白石圆。
松萝生古岸，石径入松萝。
松柏无人迹，松声入石房。
何当见山水，不见白云中。
####################
####################
夜月明皎皎，清风满夕阳。
风光摇落日，月照月光明。
月色凝清露，风吹露滴霜。
风光凝翠色，风动玉壶寒。
日暮风吹起，风吹月色寒。
何当一番泪，不见一枝枝。
####################
INFO:tensorflow:Restor
####################
湖上春风起，春风满夕阳。
春风吹落日，春色落花枝。
春色春风起，春风落日斜。
春风吹柳色，春色落花枝。
春色春风起，春风蝶舞腰。
春风吹不发，春色满春风。
####################
INFO:tensorflow:Restor
####################
海上山中见，青山路渐除。
风光生远岫，云影入山川。
水色浮云外，山寒水色清。
风光生远岫，风雨入山声。
日暮无人见，人间不可寻。
何当见君子，不见白头翁。
####################
INFO:tensorflow:Restor
####################
月照秋风起，风光澹澹摇。
风光摇碧落，月照玉山颓。
风起风吹落，风吹玉树新。
风光摇碧落，风动玉壶清。
日暮风光动，风吹玉树枝。
何当一杯酒，何处更相逢。
####################
```

B

```
[INFO] write tang poem
INFO:tensorflow:Restor
####################
日日春光满，春风满地清。
不知何处处，不似旧时时。
####################
INFO:tensorflow:Restor
####################
红烛开新阁，春风满绿苔。
云门春色上，花落一枝香。
风卷云初起，烟生夜影微。
风吹云影外，山色入云端。
####################
INFO:tensorflow:Restor
####################
山中有高树，不见白云中。
山色连云外，山禽入暮山。
山僧来远夜，山色近人稀。
山色连山远，山深雨未收。
归来不可见，日暮一声风。
何处无人见，何人更有情。
####################
INFO:tensorflow:Restor
####################
夜久清风起，清风日渐曛。
云山无限意，不见一枝香。
远岫连云远，秋声入梦深。
云生云气湿，山色水连寒。
野鸟惊秋夜，山寒水色清。
相逢不可见，何处有残阳。
####################
INFO:tensorflow:Restor
####################
####################
湖上春光满，春风一曲歌。
不知何处水，不似上阳台。
日暮风吹远，春风入楚春。
春风吹玉砌，红树映花枝。
欲待春光里，还随白露姿。
不知何处去，不敢怨春风。
####################
INFO:tensorflow:Restor
####################
海上山河水，山川见故乡。
山川无限路，一望一时清。
山色连山远，山川入梦迟。
山光连水色，山色入空林。
此地无人迹，何人得此身。
此时终不得，不是在家心。
####################
INFO:tensorflow:Restor
####################
月明初照日，日暮暮云深。
山色连山远，山阴入梦深。
云生云外树，山色入空林。
日暮风吹远，秋风吹雨声。
风吹山上月，月下夜相思。
莫问东风景，何人更有期。
####################
```

C

```
[INFO] write tang poem
INFO:tensorflow:Restor
####################
日暮花如雪，春风入夜深。
####################
INFO:tensorflow:Restor
####################
红花开日日，春色正纷纷。
####################
INFO:tensorflow:Restor
####################
山中无人识，山水独无处。
####################
INFO:tensorflow:Restor
####################
夜夜月明月，夜深夜夜啼。
####################
INFO:tensorflow:Restor
####################
湖上春风起，春风吹雨声。
####################
INFO:tensorflow:Restor
####################
海上山头白，春风吹雨声。
####################
INFO:tensorflow:Restor
####################
月明千里月，风月夜深夜。
####################
```

结果分析：

1. 我们发现在同样取 learning_rate 为 0.001 的时候，epoch 多了之后，反而 training_loss 变高，这说明本身这个模型可能设计的就不太好，导致不能完全收敛到全局最优解。

2. 总体上而言，当 training_loss 变小之后，生成的诗歌的句子的长度也减小了。这说明神经语言模型得到了一定的提升，语言模型会较早的发现 end_token，而不会陷入生成很多很长且没有多少道理的诗句。

这些诗句中，还是有一些比较惊艳的，这里摘录出来，以飨读者：

- 何当见君子，不见白头翁。
- 日暮花如雪，春风入夜深。
- 云门春色上，花落一枝香。
- 莫问东风景，何人更有期。
- 海上山头白，春风吹雨声。
- 归来不可见，日暮一声风。
- 欲待春光里，还随白露姿。
- 相逢不可见，何处有残阳。

4 Others

对唐诗生成这个任务的一些思考：

1. 本文对字的嵌入是将所有的字编号来了个索引，但是字与字之前在语义上的相似性，并没有通过这个编号本身表现出来。如果换作是 Chinese Character 的 embedding 向量，效果会不会更好一些。利用 word2vec 的原理，只不过这里的 word 是每一个字。然后还是用这个古诗的语料库进行训练。

2. 本文的方法基本指利用了语言模型，我们知道古诗词有严格的韵脚、对仗等限制，未来也许可以添加这些条件，让诗歌看起来更工整。

3. 一般而言，生成任务都有一些 evaluation 来评测我们的模型是好是坏，但是在诗歌生成的案例中，似乎并没有这样的一个评判方法。可否考虑大量的人工评分数据作为 evaluation 值得思考。艺术作品的好与坏，事实上，是很难用机器去定量评价的。艺术本就闪耀着人性的光辉，人性的东西应该回归由人性自身去评价，