语言模型

1.语言模型

语言模型就是这样的东西：给定一个一句话前面的部分，预测接下来最有可能的一个词是什么。
假设：一个句子里的单词基于之前出现的单词
目标：预测下一个单词出现的概率。
用**公式**表示：一个句子的概率是，给定前面的单词的条件下，每个单词的概率的乘积
P(w1,w2,w3)=p(w3|w1,w2)p(w1,w2)=p(w3|w1,w2)p(w2|w1)p(w1)
语言模型是对一种语言的特征进行建模，它有很多很多用处，最重要的有以下两种：
打分机制：选择最高概率的单词作为预测值，比如在语音识别中，往往是若干个可能的候选词，这时候就需要语言模型来从这些候选词中选择一个最可能的。
产生文本：并用此接着预测下一个单词，直到产生完整的一句话，由此产生特定风格的文本。有一些研究者就是把莎士比亚的小说，输出具有莎士比亚风格的句子；或者用 linux code 作为训练集，输出具有这种语法的代码。

2.N-Gram model

该模型基于这样一种**假设**，第 N 个词的出现只与前面 N-1 个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的乘积。这些概率可以通过直接从语料中统计 N 个词同时出现的次数得到。常用的是二元的 Bi-Gram 和三元的 Tri-Gram 而且效果还不错。

我 昨天 上学 迟到 了 ，老师 批评 了 ____。

如果用 2-Gram 进行建模，那么电脑在预测的时候，只会看到前面的『了』，然后，电脑会在语料库中，搜索『了』后面最可能的一个词。不管最后电脑选的是不是『我』，我们都知道这个模型是不靠谱的，因为『了』前面说了那么一大堆实际上是没有用到的。如果是 3-Gram 模型呢，会搜索『批评了』后面最可能的词，感觉上比 2-Gram 靠谱了不少，但还是远远不够的。**因为这句话最关键的信息『我』，远在 9 个词之前！**

你可能会想，可以提升继续提升 N 的值呀，比如 4-Gram、5-Gram.....。实际上，这个想法是没有实用性的。**因为我们想处理任意长度的句子，N 设为多少都不合适**；另外，模型的大小和 N 的关系是指数级的，4-Gram 模型就会**占用海量的存储空间**。

正因为 N-Gram 太简单粗暴了，所以 RNN 正是登场。

RNN

1. 普通神经网络

到底什么是 RNN，为什么这种形式的神经网络被称为 RNN 呢？与普通神经网络有什么不同呢？

现在我们拿两者的结构进行比较。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的，所以这种普通的神经网络对于很多问题却无能无力，比如处理序列型数据（即输入的样本是有顺序的）。因为顺序型数据的预测往往跟之前的输入有关，而普通神经网络的隐藏层又无法相互传递信息。所以普通神经网络对序列数据的预测效果普遍不好。

2. Why it is called RNN

而 RNN 对于处理序列数据有天然的优势。隐藏层之间的节点是顺序连接的，因此隐藏层 St 的输入不仅包括输入层的输出 Xt 还包括上一时刻隐藏层的输出 St-1。这个结构使得 RNN 可以对前面的信息进行记忆并应用于当前输出的计算中。因此，一个序列当前的输出 Ot 与前面的输出 Ot-1 也有关。

它之所以称为循环神经网路，是因为 RNN 在每步骤 t 做的工作是一样的，每一步每一层都共享参数 U,V,W，只不过输入不同而已。在传统的神经网络中，这些参数都是不同的。

RNN 展开后可以看成一个深层的神经网络，比如一个句子有 5 个单词，每个单词是输入，那么 RNN 就可以展开成 5 层的神经网络。

3. RNN vs NN vs N-Gram

（1）通过顺序连接隐藏层，记住以前输入的信息

（2）**通过循环的方式，处理任意长度的句子。**因为神经网络的输入层单元个数是固定的，因此必须用**循环或者递归的方式来处理长度可变的输入**。循环神经网络实现了前者，**通过将长度不定的输入分割为等长度的小块，然后再依次的输入到网络中，从而实现了神经网络对变长输入的处理**。一个典型的例子是，当我们处理一句话的时候，我们可以把一句话看作是词组成的序列，然后，每次向循环神经网络输入一个词，如此循环直至整句话输入完毕，循环神经网络将产生对应的输出。如此，我们就能处理任意长度的句子了。

4. RNN 能干嘛

语言模型生成文本（给你一个单词序列，每次根据前面的单词预测下一个单词的可能性，挑选可能性最大的单词从而生成文本）

机器翻译（与语言模型关键的区别在于，需要将源语言语句序列输入后，才进行输出，因为要听完所有单词才知道一句话的意思）

语音识别（输入声波，预测一系列的语音碎片和它们的概率）

图像描述（CNN+RNN=为图像打标签）

5. 公式

RNN 的公式很简单，参数很少

$$s_t = f(Ux_t + Ws_{t-1})$$

$$o_t = \mathrm{softmax}(Vs_t)$$

• Xt 是第 t 步的输入。它一般是以 one-hot vector 的形式存在，至于什么是 one-hot 向量后面会讲到

• St 为隐藏层的第 t 步的状态，它是网络的记忆单元，根据之前的隐藏层和当前的输入层计算而来。

• F 函数是一个非线性激活函数，一般是 tanh 或 relu。在计算 s0 时，即第一个单词的隐藏层状态，需要用到 s-1，但是其并不存在，在实现中一般置为 0 向量；

• Ot 是第 t 步的输出，如果要在一个句子中预测下一个单词，**那 Ot 就是一个词汇表中的概率向量。**

6. RNN 的 BPTT（多练）

求一般的神经网络的参数是用梯度下降法+向后传播法。

（手写普通神经网络的 FP 过程，BP 算法）（sigmoid 与 tanh 的函数形式及其导数形式）

直觉：tanh 的导数小于 1，而且 tanh 的导数的两端值都趋于 0，因此矩阵（$\frac{\partial s_t}{\partial s_k}$）中的元素值很小，再经过多次矩阵相乘，梯度就会以指数级的速度趋于 0（sigmoid 的导数小于 1/4，**用 sigmoid 就更容易出现梯度消失的问题**）

如果 W 中的值太大，那矩阵的值也跟着变大，多次矩阵相乘后就爆炸了。

在这里就不给大家列出具体的推导过程了，只是想给大家一个直观的理解，梯度消失和爆炸是怎么出现的。对推导过程感兴趣的同学可以参考这篇文章。**（打开该网页）**

7. RNN 的梯度消失/梯度爆炸问题

梯度爆炸并不是一个严重的问题，因为它很容易被发现，一旦出现梯度爆炸，程序会崩溃，得到一堆的 NA，而且很容易解决，设立一个上限，一旦达到这个阈值就强行把梯度消减下来

梯度消失就比较麻烦，不容易被发现也不容易被解决。幸运的是，还是有些解决办法的。比如合理地初始化 W，一般流行的做法是，W 的元素随机赋值在 0-1 之间，方差很小。
还有用 RELU 函数。因为 RELU 函数的导数不是 0 就是 1，实际运算程序时很少出现 relu 函数中的值小于 0 的情况，因此多个矩阵相乘时不会越乘越小。

8. 解决办法

合理初始化权重：

Relu 函数：（Relu 的函数形式及其导数形式）

前两种方法，都是让矩阵（$\frac{\partial s_i}{\partial s_{i-1}}$）的值尽可能贴近于 1，**终究还是难以抵挡指数函数的威力**。有一种更流行的方法，LSTM 及其变体，彻底解决梯度消失问题

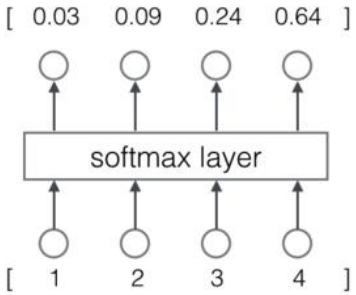
9. 基于 RNN 的语言模型的编程实例（多练）

不知道前面的理论部分是不是让感觉特别抽象，为了让大家更好地理解 RNN，下面我给大家简单地讲讲编程实例把，让你们直观地感受下样本和预测值到底是怎么样的。

在这里就不给大家讲代码了，有兴趣的可以自己看看这个网页，有很详细的代码。

为什么要用 one-hot vector：为了让语言模型能够被神经网络处理，我们必须把词表达为向量的形式

为什么用 softmax：把输出转化为概率：每一项为取值为 0-1 之间的正数；所有项的总和是 1。



$$
\begin{aligned}
y_1 &= \frac{e^{x_1}}{\sum_k e^{x_k}} \\
&= \frac{e^1}{e^1 + e^2 + e^3 + e^4} \\
&= 0.03
\end{aligned}
$$

为什么用交叉熵作为损失函数：SDD 计算量大，而且不容易求最小值

$$L(y, o) = -\frac{1}{N}\sum_{n \in N} y_n log o_n$$

在上式中，N是训练样本的个数，向量y_n是样本的标记，向量o_n是网络的输出。标记y_n是一个one-hot向量，例如$y_1 = [1, 0, 0, 0]$，如果网络的输出$o = [0.03, 0.09, 0.24, 0.64]$，那么，交叉熵误差是（假设只有一个训练样本，即N=1）：

$$
\begin{aligned}
L &= -\frac{1}{N}\sum_{n \in N} y_n log o_n && (86) \\
&= -y_1 log o_1 && (87) \\
&= -(1 * log 0.03 + 0 * log 0.09 + 0 * log 0.24 + 0 * log 0.64) && (88) \\
&= 3.51 && (89)
\end{aligned}
$$

suppose the softmax output is (0.87 0.01 0.12) as above and the training value is (1 0 0). The SSD： (1 – 0.87)^2 + (0 – 0.05)^2 + (0 – 1.00)^2.

Cross entropy loss: (1 * Ln(0.87)) + (0 * Ln(0.01)) + (0 * Ln(1.00))= -0.06 + 0 + 0 = -0.06

10. RNN 的缺陷：梯度消失导致长期依赖性问题

如果相关联的的信息只相隔一两个单词，那 RNN 完全可以应用之前的单词去预测下一个单词，比如 the cloud are in the sky，预测最后一个 sky。

正因为 RNN 有梯度消失的问题，因此它很难记住较早之前学的东西。如果文本中两个相关联的词相距太远，RNN 就不能把信息链接起来了。这个缺陷就叫做长期依赖性问题（long term dependency problem）**总之 RNN 在理论上可以记住之前的所有信息，理论上很完美，但实际上用在程序中往往效果不好。**

11.一个生动的例子

我今天要做红烧排骨，首先要准备排骨，然后…，最后美味的一道菜就出锅了。现在请 RNN 来分析，我今天做的到底是什么菜呢. RNN 可能会给出“辣子鸡”这个答案. 由于判断失误, RNN 就要开始学习 这个长序列 X 和 ‘红烧排骨’ 的关系 ，而 RNN 需要的关键信息 ”红烧排骨”却出现在句子开头,回答错了，就要根据误差进行学习，优化参数，但是向后传播时, 每传一层就要乘以权重 W 和 tanh 的导数(tanh 的导数最大值为 1), 如果 W 太小 W<1, 乘到最后误差乘成 0 了。如果 W>1，乘到最后就变成无穷大的数了。

所以这个时候专门拯救 RNN 梯度消失的 LSTM 就粉墨登场了。

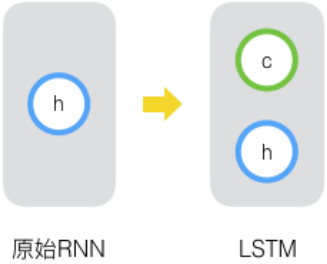
LSTM （长短期记忆模型）

1.LSTM 的设计理念

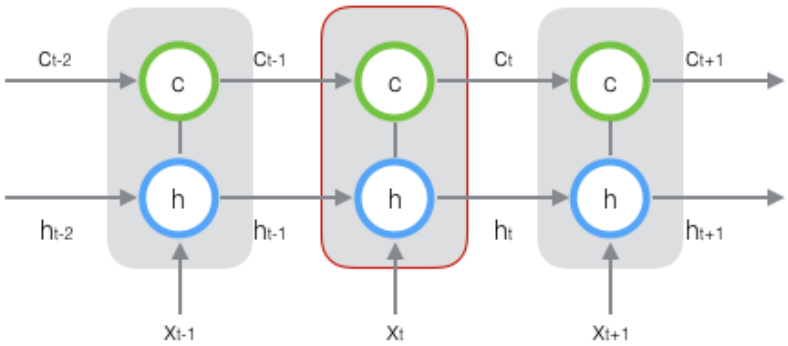
LSTM 是 RNN 的一种形式，它们只是在计算隐藏层的方式不同，但也是输入 St-1 和 Xt 得到 St

LSTM 是为避免 RNN 的长期依赖性问题专门设计出来的，记住长远信息是它的长项。

（1）**原始 RNN 的隐藏层只有一个状态，即 h，它对于短期的输入非常敏感。那么，假如我们再增加一个状态，即 c，让它来保存长期的状态**，那么问题不就解决了么？

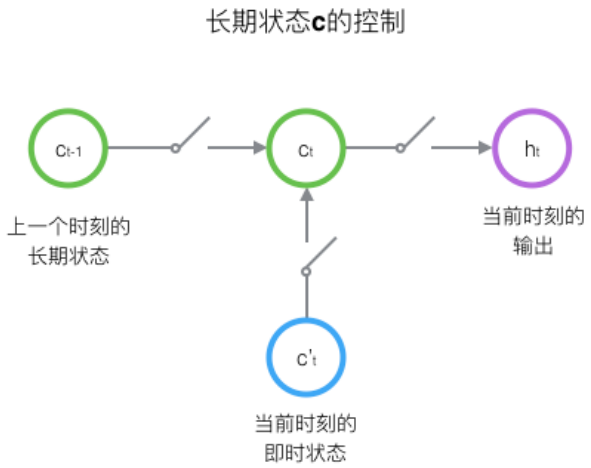


新增加的状态 c，称为单元状态(cell state)。我们把上图按照时间维度展开：



在 t 时刻，**LSTM 的输入有三个**：当前时刻网络的输入值 Xt、上一时刻 LSTM 的输出值 ht-1、以及上一时刻的单元状态 Ct-1；**LSTM 的输出有两个**：当前时刻 LSTM 输出值 ht、和当前时刻的单元状态 Ct。注意 X,C,h 都是向量。

（2）LSTM 的关键，就是怎样控制长期状态 c。在这里，LSTM 的思路是使用三个控制开关。第一个开关，负责控制保存长期状态 Ct-1；第二个开关，负责控制把即时状态输入到长期状态 Ct；第三个开关，负责控制根据长期状态 Ct 输出当前的 LSTM 的隐藏层 ht。三个开关的作用如下图所示：



LSTM 引入的这个细胞状态概念，可以看成是**单元的内部记忆。可以把 cell state 看成是计算 ht 的中间产物，但更好的是理解成 LSTM 中储存内部长期记忆的部件。**

2. LSTM 的结构

RNN 计算隐藏层 St 的过程很简单，就是一个 tanh 函数，但 LSTM 就复杂多了，但不要担心，不要害怕，我下面会从理念到细节，一步步给大家介绍这个结构

（1） Ct 细胞状态（当前单元的内部记忆模块）

LSTM 的关键就是细胞状态，表现为在图上方贯穿运行的水平线。

细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。

其中 c_{t-1}是额外制造出来、可线性自连接的单元

（2） 门

门是一种让信息选择式通过的方法。三种门的的方程形式相同，但参数矩阵不同

门的乘法加法都是点乘运算，不是矩阵运算

加法门可以认为是两组信息的叠加，乘法门可以看做一组控制信号对另一组信息的控制。(限制某些时刻输入的表达。限制某些时刻输出的表达。限制某一时刻状态到下一时刻状态的表达。)

gates 只是起到限制信息的量的作用。因为 gates 起到的是过滤器作用，所以所用的激活函数是 sigmoid 而不是 tanh。

因为门的输出是 0 到 1 之间的实数向量，那么，当门输出为 0 时，任何向量与之相乘都会得到 0 向量，这就相当于啥都不能通过；输出为 1 时，任何向量与之相乘都不会有任何改变，这就相当于啥都可以通过。因为（也就是 sigmoid 函数）的值域是(0,1)，所以门的状态都是**半开半闭**的，即选择性让信息通过。

（3）遗忘门（作用在 Ct-1 上）

决定要扔掉什么信息，由 sigmoid 层实现。输入 ht-1 和 xt，输出 0-1 之间的数。1 表示全部保留，0 表示全部忘记。

为什么还要遗忘信息呢？

a 举个例子，如果细胞状态记住了一个主语的性别，那下次遇到这个主语时它就知道性别是什么了，但换了一个新的主语，它就要忘记旧主语的性别。

b 再举个例子，如果当前的输入是个句号，直觉上那么句号之后跟句号之前的上下文依赖性较低，这个时候我们可以适当忘记之前的一些东西，比如说忘记 8 成，好让后面更好地学习。

（4）输入门（作用在 $\tilde{C}_t$ 上）

要储存什么信息。it 决定要更新什么，tanh 层产生候选值 $\tilde{C}_t$

我们把旧状态 Ct 与 f_t 相乘，丢弃掉我们确定需要丢弃的信息。接着加上 i_t*C_t。这就是新的候选值，（比如说，我要把新主语的性别加进去）

举个例子：在预测下一个单词的问题中那么 LSTM 可以通过输入门关闭输入阀（it 值为 0），不让那些单词影响 Cell 的值，从而隔得远的信息的关系用一次偏导就可以描述，而没有什么中间状态，不需要偏导的偏导的偏导，**从而解决梯度消失的问题**。

(5) 更新细胞状态

我们就把 LSTM 关于当前的记忆和长期的记忆组合在一起，形成了新的单元状态。**由于遗忘门的控制，它可以保存很久很久之前的信息，由于输入门的控制，它又可以避免当前无关紧要的内容进入记忆。**

（6）输出门

输出由细胞状态决定，但要加个过滤器，决定从细胞状态输出什么作为隐藏层 ht。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。

如此大费周章的最终是同普通 RNN 一样要计算当前隐藏状态。当前隐藏状态 h_t 是从 c_t 计算得来的，因为 c_t 是以线性的方式自我更新的，所以先将其加入带有非线性功能的 tanh(c_t)。 随后再靠输出门 o_t 的过滤来得到当前隐藏状态 h_t。

3. RNN vs LSTM

（1）LSTM 是 RNN 的一种特殊形式：如果 输入门 i_t 全为 1，遗忘门 f_t 全为 0（全忘了），输出所有 o_t=1（记得多少输出多少）那就差不多是一个标准的 RNN，不过最后多了个 tanh

（2）RNN 与 LSTM 不同的是 LSTM 靠 3 个 gates 将信息的积累建立在线性自连接的 memory cell 之上，并靠其作为中间物来计算当前 h_t 。

4. 为什么 LSTM 能解决 RNN 梯度消失的问题

（1）在 RNN 中，当前状态值 S（t）= tanh(x(t) U + W S(t-1))，正如上面所述在利用梯度下降算法链式求导时是连乘的形式，若其中只要有一个是接近零的，那么总体值就容易为 0，导致梯度消失，不能解决长时依赖问题。
而 LSTM 更新状态值：是相加的形式，所以不容易出现状态值逐渐接近 0 的情况。

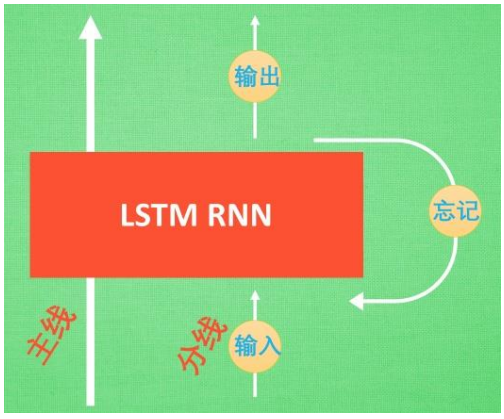
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

（2）忘记中间无关的信息,一次偏导就可以
（3）对 ct 求导是 ft，ft 中元素控制在 1 上下就行

5. 对 LSTM 的生动理解

LSTM 就是为了解决没有办法回忆起久远记忆这个问题而诞生的. LSTM 和普通 RNN 相比，多出了三个控制器. (输入控制，输出控制，忘记控制). 现在, LSTM RNN 内部的情况是这样.

他多了一个 控制全局的记忆，我们用粗线代替. 为了方便理解，我们把粗线想象成电影或游戏当中的 主线剧情. 而原本的 RNN 体系就是 分线剧情. 三个控制器都是在原始的 RNN 体系上，我们先看 输入方面 ，如果此时的分线剧情对于剧终结果十分重要，输入控制就会将这个分线剧情按重要程度 写入主线剧情 进行分析. 再看 忘记方面，如果此时的分线剧情更改了我们对之前剧情的想法，那么忘记控制就会将之前的某些主线剧情忘记，按比例替换成现在的新剧情. 所以 主线剧情的更新就取决于输入 和忘记 控制. 最后的输出方面，输出控制会基于目前的主线剧情和分线剧情判断要输出的到底是什么.基于这些控制机制, LSTM 就像延缓记忆衰退的良药，可以带来更好的结果.



总结成一句话就是：LSTM 的核心是由当前时刻的输入 Xt、上一时刻的输出 ht-1、上一

时刻的记忆 Ct-1 共同决策的，并且产生一个新的输出 ht，同时又更新了内部的记忆 Ct。

LSTM 的变体：GRU（Gated Recurrent Unit）

存在着相当多的变体，最流行的一种就是 GRU
Reset gate 决定如何合并新输入和之前的记忆
Update gate 决定要保留之前多少信息
如果 reset gate 是 1，update gate 是 0，就是一个标准的 RNN

1. 特点

只有两个门
不用计算 ct
输出 ht 时不用用第二次非线性激活函数
GRU 只用了两个 gates，将 LSTM 中的输入门和遗忘门合并成了更新门。并且并不把线性自更新建立在额外的 memory cell 上，而是直接线性累积建立在隐藏状态上，并靠 gates 来调控。

2. GRU 和 LSTM 的比较：

哪个更好还没有定论，两者效果相似。但 GRU 的参数少，训练速度一般更快

我知道这一个多小时没有办法讲其中的详细的数学推导过程和编程思路，我的初衷就是想让大家对 RNN 和 LSTM 有一个直观的理解，所以我的重点是在介绍 RNN 和 LSTM 的设计理念，为什么要这样设计，起到的作用是什么，那你们之后自学的时候效率就会提高很多了。谢谢大家！谢谢老师！