长短期记忆神经网络基础学习笔记

> RNN是一类用于处理序列数据的神经网络。就像卷积网络是专门用于处理网格化数据X（如一个图像）的神经网络，RNN网络是专门用于处理序列x(1), ⋯,x(τ)的神经网络。正如卷积网络可以很容易地扩展到具有很大宽度和高度的图像，以及处理大小可变的图像，循环网络可以扩展到更长的序列（比不基于序列的特化网络长得多）。大多数循环网络也能处理可变长度的序列。

一、RNN

> 首先要分清楚一个概念，RNN是两种神经网络的缩写，一种是递归神经网络（RecursiveNeural Network），一种是循环神经网络（Recurrent Neural Network），虽然这两个概念有千丝万缕的联系，但这里主要讨论的是第二种，也就是循环神经网络及其变种。

循环神经网络是指一个随着时间的推移，重复发生的结构。例如，如果你有一个序列X = ['H','E', 'L', 'L']，该序列被送到一个神经元，而这个神经元的输出连接到它的输入上。在步骤0的这个时刻，字母“H”是作为输入传入的，在步骤2时，字母“E”被作为输入传入。随着时间的推移展开这个网络将变成如下图所示的网络结构：

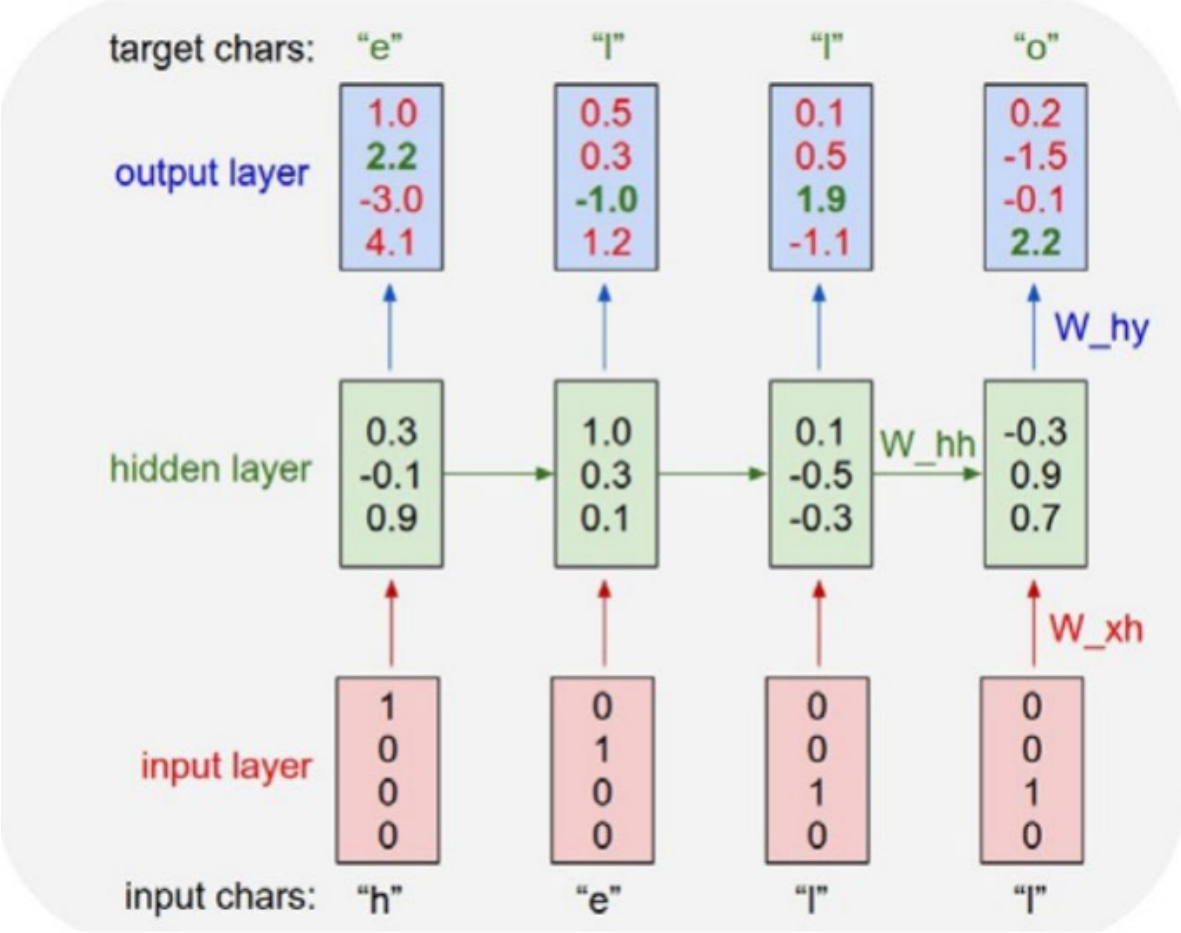


图4-27 RNN单元的展开

递归神经网络仅仅是广义化的循环神经网络。**循环神经网络在一个序列的长度上的权重是共享的（并且维度保持不变）**。因为，当遇到一个训练时间和测试时间长度不同的序列时，是不能处理位置独立权重的。递归网络的权重（与维数保持恒定）出于同样的原因在每一个节点被共享。

这意味着，所有的W_xh权重是相等的（共享），以及W_hh权重也是相等的。它是单个的神经元，并且能够及时展开。

递归神经网络看起来如下图所示：

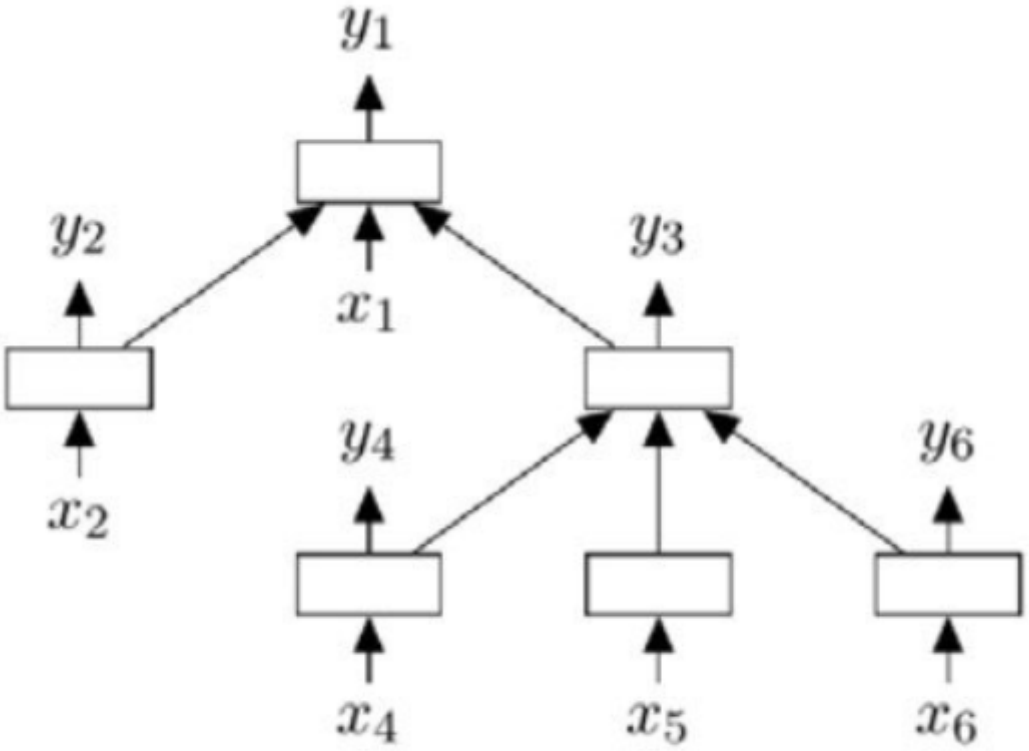


图4-28 递归神经网络示意图

为什么被称为递归神经网络呢？因为每个父节点的子节点和另外一个节点类似。

> 到底使用哪种神经网络取决于实际情况。在Karpathy的博客中可以看到，他的项目是生成一个个字符，而处理中不需要分层，对这种情况来说循环神经网络是不错的选择。
>
> 如果你想生成一个解析树，用递归神经网络会更好些，因为它有助于创造更好的分层表示。

1. 什么是循环神经网络?

正式进入循环神经网络之前，来想一下我们的思考步骤，或者叫思考时序。

我们不会每一秒钟都从头开始思考。比如，当你看一本书时，会根据以往学习的知识理解每一个词。你会从上下文中产生联想，帮助你更好地理解这篇文章。当你冒出来一个想法或问题后，会通过读本书来归纳总结，试着印证你的想法或者回答你的问题。

人类的这一大特点，无法在传统的神经网络中找到类似的，这也是一般神经网络的一个缺点。例如，假设你要将电影中每个时刻发生的事按时间归类，传统的神经网络目前还无法做到，因为这需要使用之前在电影中出现的事件推理出后面发生的事情，而循环神经网络可以解决这一问题。它们可以在网络中循环，并能够维持信息，如下图所示：

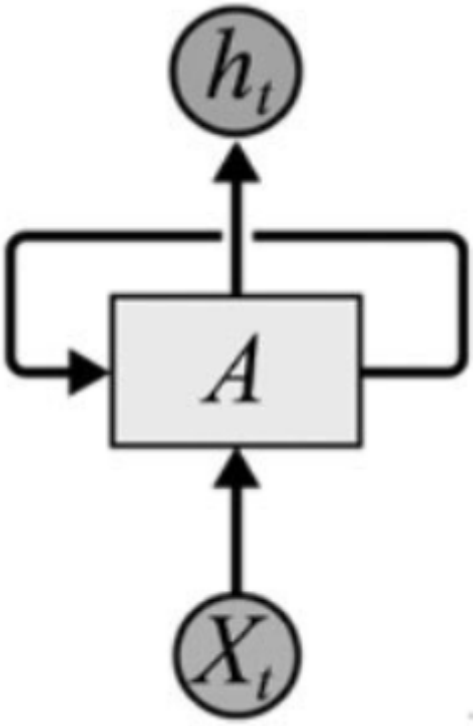


图4-29 一个循环神经单元

1）循环神经网络有回路

在上图中，神经网络的单元A，它的输入值是xt，输出值是ht。信息通过回路从网络的目前状态传递到下一个状态。同一个单元不停地处理不同的输入值，而这些值是自己产生的。大家觉得循环神经网络的回路很神秘吗？如果你深入思考会发现，它们与正常的神经网络没什么不同。反复出现的神经网络可以被认为是在同一个网络中的多个副本，每个都传递消息给继承者，也就是下个时态的神经元。

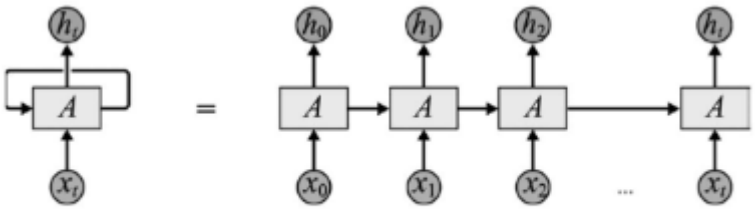


图4-30 单元展开循环

2）已展开的递归神经网络

这个链式结构揭示了与循环神经网络密切相关的序列。它们是让神经网络能使用这些数据的一种自然结构。这么好的东西肯定会被用在各个方面。

在过去的几年里，RNN已经成功地应用在各种问题上，并取得令人难以置信的成功。例如，语音识别、语言建模、翻译、字幕、图像.....不胜枚举。这个结果的关键是使用"LSTMs"（长短期记忆网络，Long Short Term MemoryNetworks），它是一种特殊的循环神经网络的变种，对于许多任务来说，这种方法比标准的RNN好得多。

3）长时间依赖的问题

RNN的诉求之一是，它也许能将以前的信息连接到当前任务。例如，我们有时需要使用前一个视频帧理解当前帧的内容。如果RNN能做到这一点，它们会非常有用，但它们能做到吗？视情况而定。

有时候，我们只需要看最近的信息来执行现在的任务。举个例子，考虑一个语言模型试图预测基于当前的下一个词。如果我们试图预测"天空中有 "这句话的最后一个字，那么我们不需要任何进一步的语境就可以判断下一个字是云或鸟。**在这种情况下，如果相关的信息（这里指的是"天空中有"）和我们需要填词的位置之间的差距较小，那么RNN就能学会利用过去的信息。**

但有时，我们需要更多的上下文。试着预测"我在中国长大……（省略20个字），我讲一口流利的__。"的最后一个词。最近的信息表明，下一个字可能是语言的名字，但如果我们想要缩小语言名字的范围，则需要这个词的上下文。我们发现，**有时相关上下文信息和我们需要得到的词的这个位置相差很大。不幸的是，这种距离的增长将使RNN无法学习到这些信息。**

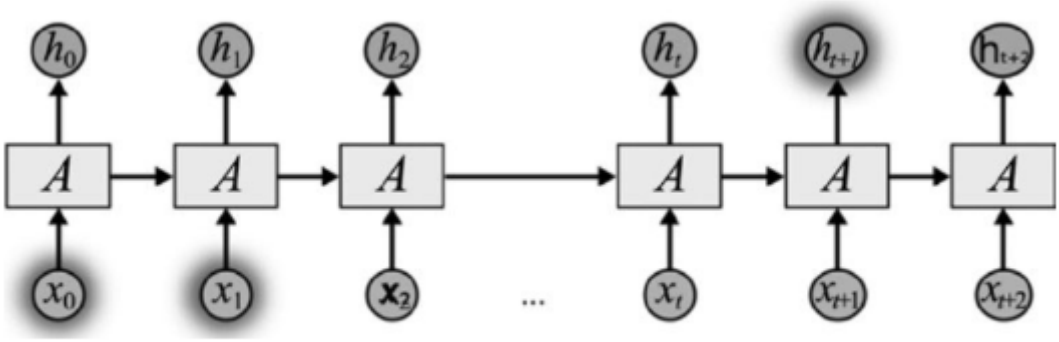


图4-32 距离较长，无法利用上下文知识

我们仔细看上图,**ht+1需要x0x1位置的信息，但由于距离较长，x0x1信息无法传导过来。**从模型结构上看，RNN完全能够处理这样的"长期依赖"问题。理论上讲，人们可以仔细挑选参数，为它们解决这种小问题。但在实践中，我们发现RNN似乎并不能处理好这些问题。Hochreiter和Bengio深入探讨了这一问题，发现了普通的RNN结构无法处理好这种问题。

值得庆幸的是，LSTMs没有这个问题。

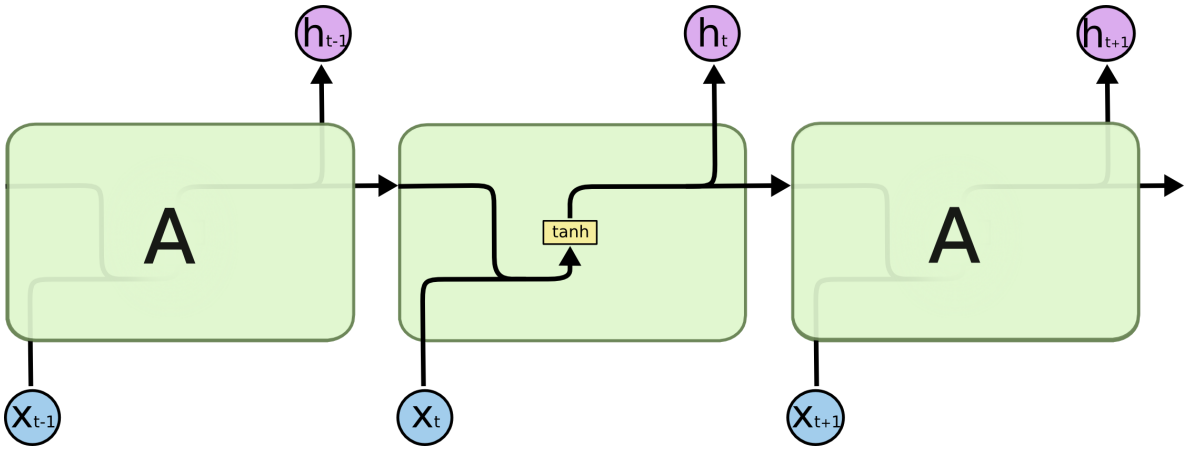
二、LSTM

> 长短时记忆神经网络（Long Short-Term Memory,LSTM）最早由Hochreiter&Schmidhuber于1997年提出，能够有效解决信息的长期依赖，避免梯度消失或爆炸。事实上，长短时记忆神经网络的设计就是专门用于解决长期依赖问题的。与传统RNN相比，它在结构上的独特之处是它精巧的设计了循环体结构。

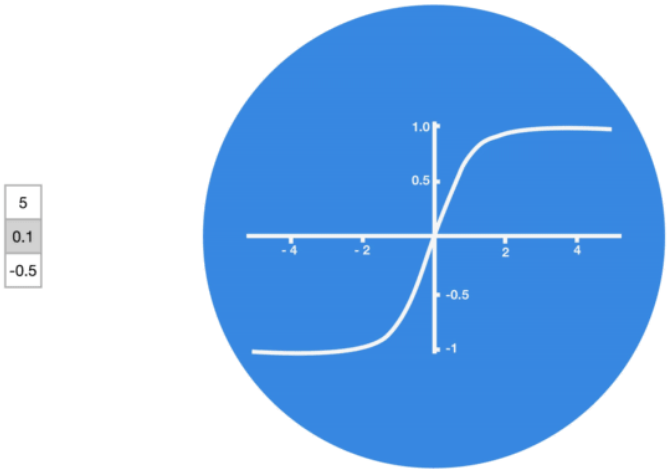
1.RNN与LSTM区别

LSTM可以学习只保留相关信息来进行预测，并忘记不相关的数据。记住长期的信息在实践中是LSTM的默认行为，而非需要付出很大代价才能获得的能力！

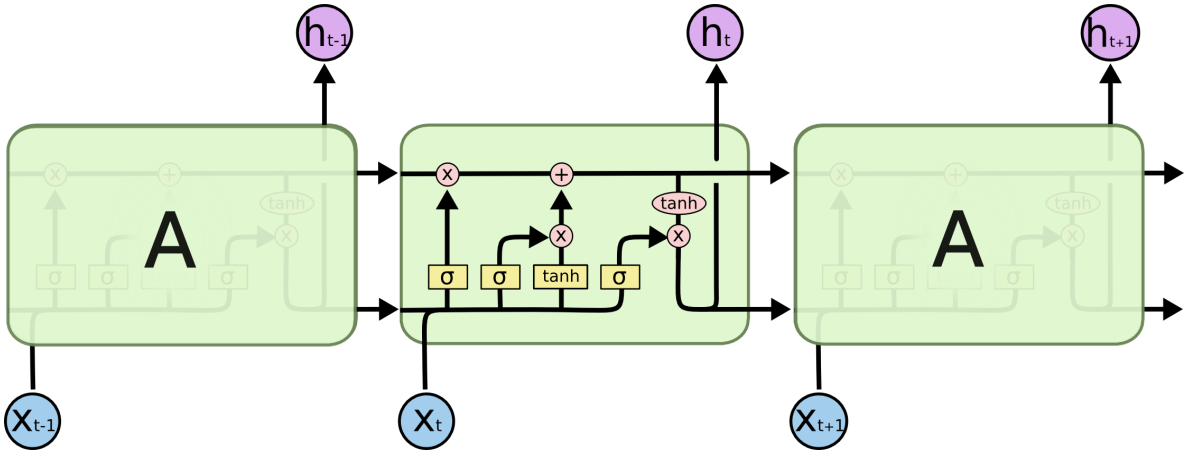
所有RNN都具有一种重复神经网络模块的链式的形式。在标准的RNN中，这个重复的模块只有一个非常简单的结构，例如一个tanh层。



激活函数 Tanh 作用在于帮助调节流经网络的值，使得数值始终限制在 -1 和 1 之间。



LSTM同样是这样的结构，但是重复的模块拥有一个不同的结构。具体来说，RNN是重复单一的神经网络层，LSTM中的重复模块则包含四个交互的层，三个Sigmoid 和一个tanh层，并以一种非常特殊的方式进行交互。

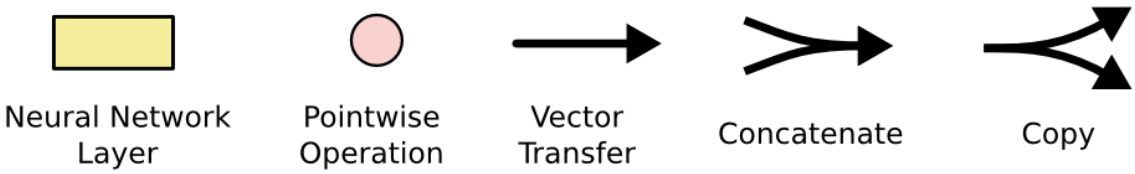


上图中，**σ表示的Sigmoid 激活函数与 tanh 函数类似，不同之处在于 sigmoid 是把值压缩到0~1 之间而不是 -1~1 之间。这样的设置有助于更新或忘记信息**：

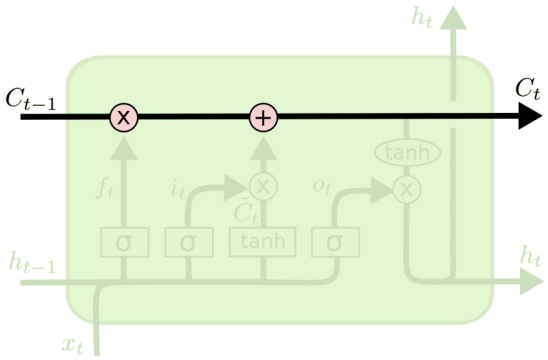
- 因为任何数乘以 0 都得 0，这部分信息就会剔除掉；
- 同样的，任何数乘以 1 都得到它本身，这部分信息就会完美地保存下来。

相当于要么是1则记住，要么是0则忘掉，所以原则就是：**因记忆能力有限，记住重要的，忘记无关紧要的。**

此外，对于图中使用的各种元素的图标中，每一条黑线传输着一整个向量，从一个节点的输出到其他节点的输入。粉色的圈代表pointwise的操作，诸如向量的和，而黄色的矩阵就是学习到的神经网络层。合在一起的线表示向量的连接，分开的线表示内容被复制，然后分发到不同的位置。

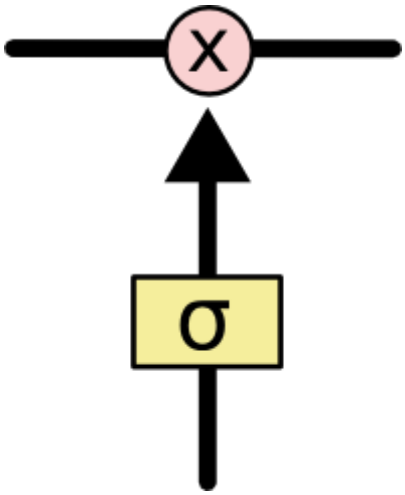


LSTM的关键就是细胞状态，水平线在图上方贯穿运行。细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。

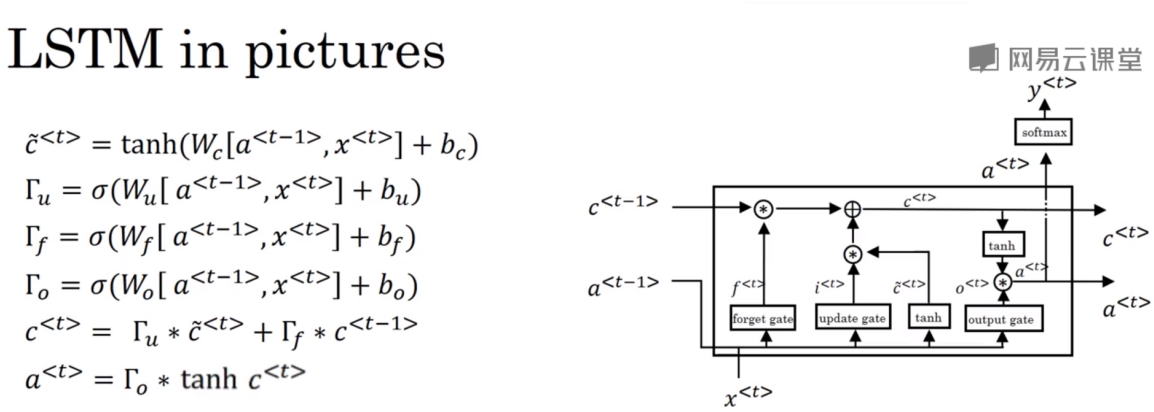


LSTM有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个sigmoid神经网络层和一个pointwise乘法的非线性操作。

如此，0代表“不许任何量通过”，1就指“允许任意量通过”！从而使得网络就能了解哪些数据是需要遗忘，哪些数据是需要保存。



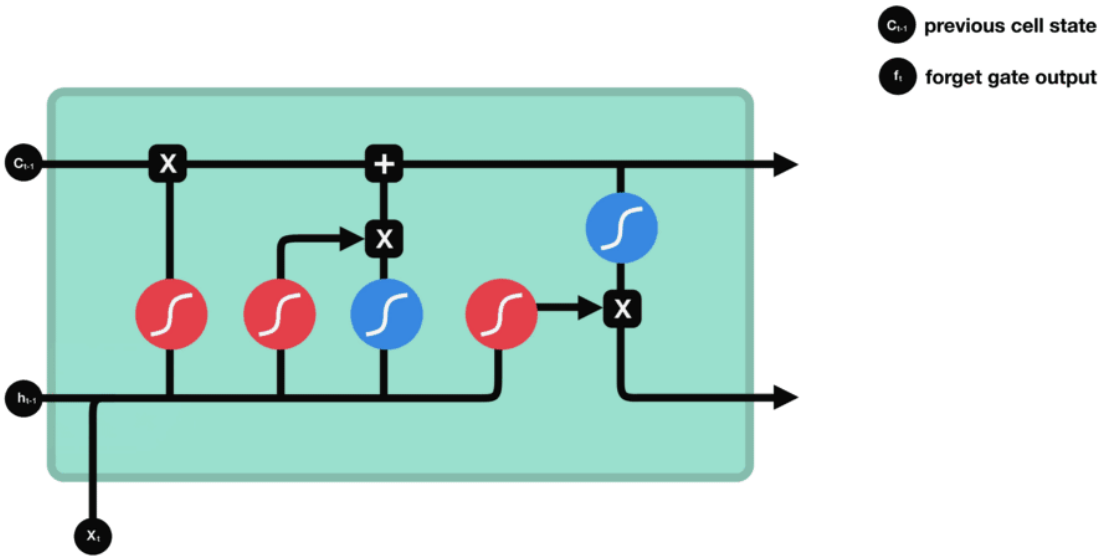
LSTM拥有三种类型的门结构：遗忘门/忘记门(forget gate)、输入门(update gate)和输出门(output gate)，来保护和控制细胞状态。在介绍这三个门之前，我们先看一下吴恩达教授关于LSTM的介绍图：



2.LSTM详细介绍

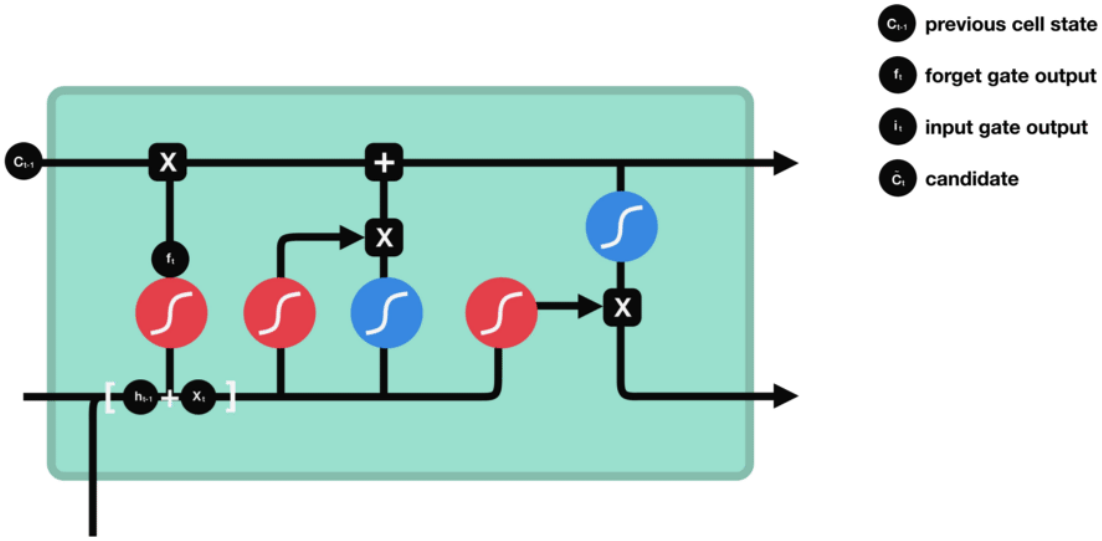
1）遗忘门(forget gate)

LSTM中的第一步是决定我们会从细胞状态中丢弃什么信息。这个决定通过一个称为“遗忘门”的结构完成。该遗忘门会读取上一个输出和当前输入，做一个Sigmoid 的非线性映射，然后输出一个向量（该向量每一个维度的值都在0到1之间，1表示完全保留，0表示完全舍弃，相当于记住了重要的，忘记了无关紧要的），最后与细胞状态相乘。过程如下图所示：



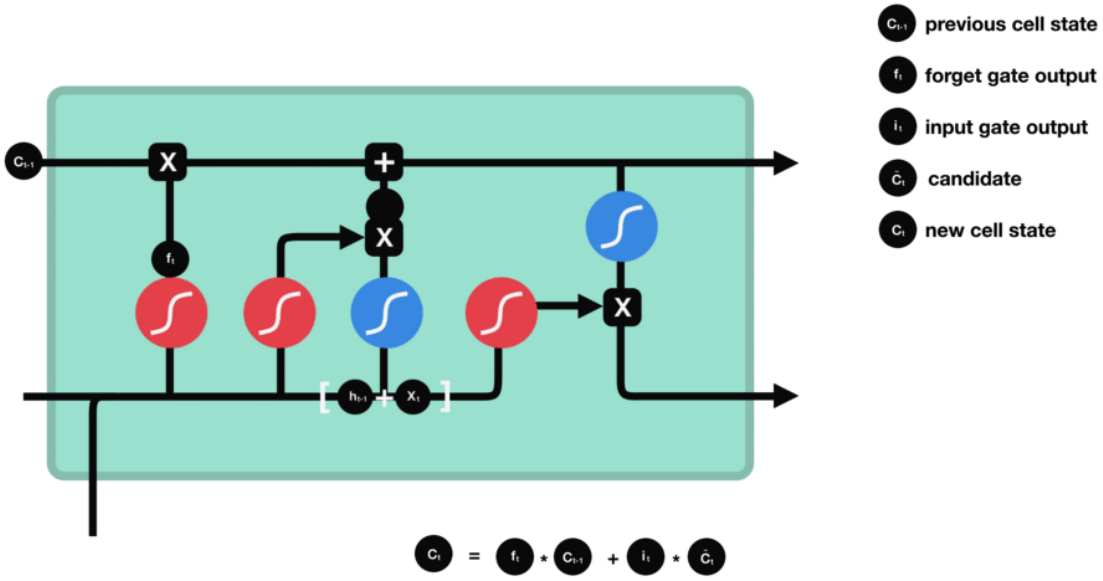
2）输入门(update gate)

LSTM中的下一步就是确定什么样的新信息被存放在细胞状态中。这里包含两个部分：
第一，sigmoid层称"输入门层"决定什么值我们将要更新；
第二，一个tanh层创建一个新的候选值向量$\tilde{C}_t$，会被加入到状态中。
这两个信息来产生对状态的更新。过程如下图所示（*红圈表示Sigmoid 激活函数，篮圈表示tanh 函数*）：



3）细胞状态：

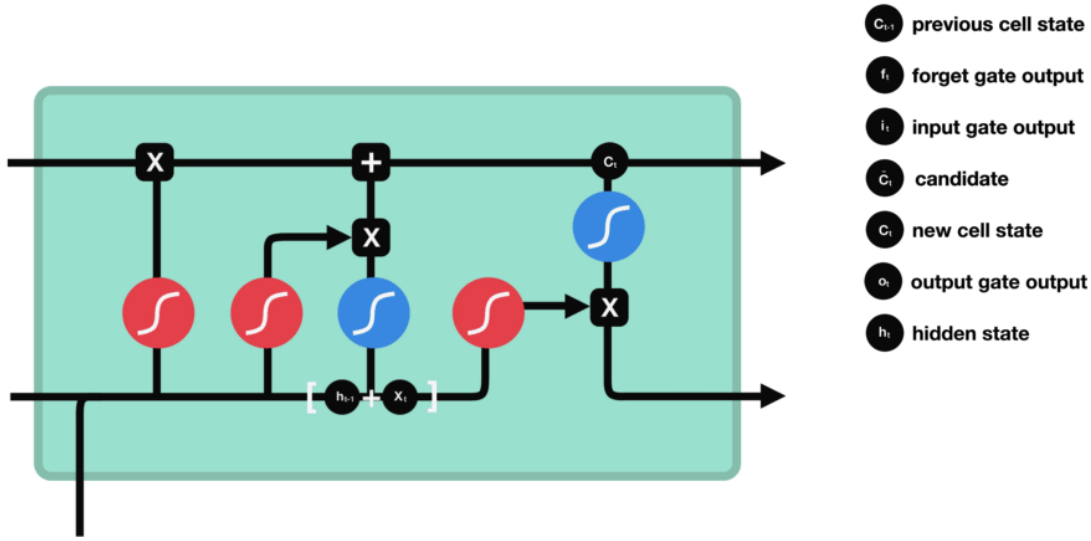
现在是更新旧细胞状态的时间了，C_{t-1}更新为C_t。前面的步骤已经决定了将会做什么，我们现在就是实际去完成。我们把旧状态与相乘，丢弃掉我们确定需要丢弃的信息。接着加上。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。
在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的性别信息并添加新的信息的地方，类似更新细胞状态。过程如下图所示：



4）输出门

最终，我们需要确定输出什么值。这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。

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。过程如下图所示：



小结：LSTM的循环体结构如下图所示：

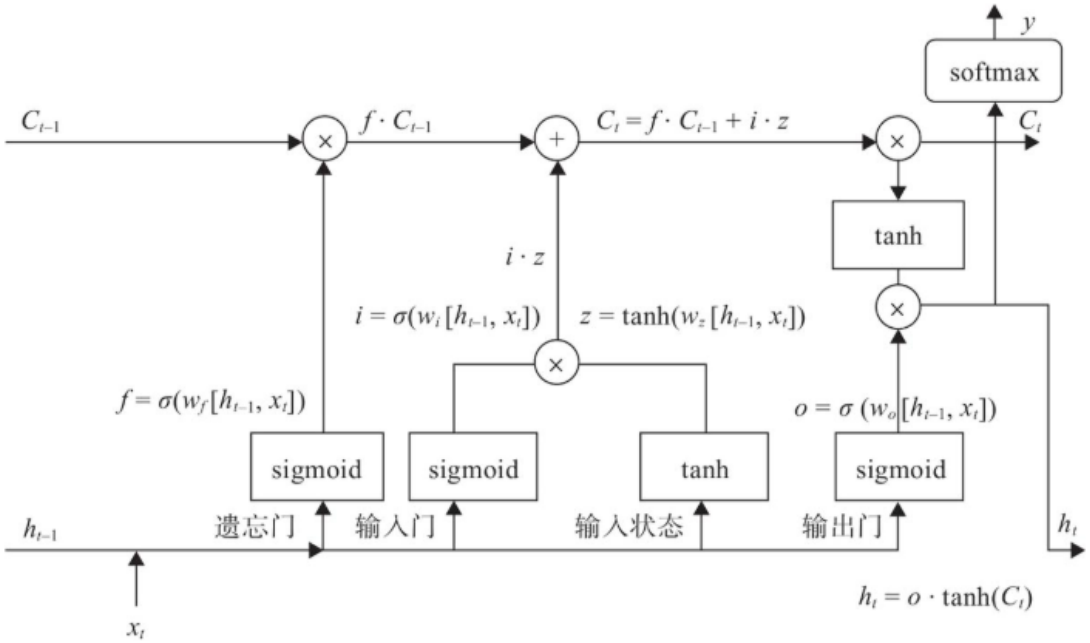
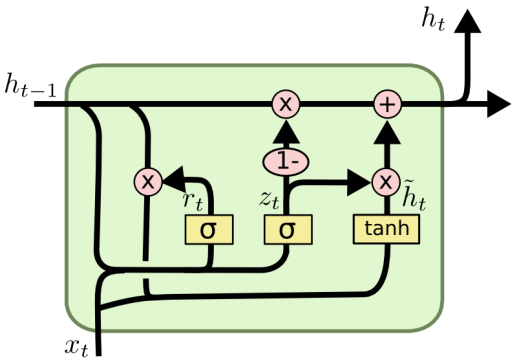


图7-8 LSTM架构图

3.LSTM变体

一个改动较大的变体是 Gated Recurrent Unit (GRU)，这是由 Cho, et al. (2014) 提出。它将忘记门和输入门合成了一个单一的 更新门。同样还混合了细胞状态和隐藏状态，和其他一些改动。**最终的模型比标准的 LSTM 模型要简单，也是非常流行的变体**。其他的变体限于篇幅就不再介绍了。



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

> 比较推荐的博客：https://blog.csdn.net/v_JULY_v/article/details/89894058
>
> https://www.julyedu.com/question/big/kp_id/26/ques_id/1851
>
> 吴恩达教授网易云课堂关于深度学习的课程：https://mooc.study.163.com/smartSpec/detail/1001319001.htm

三、LSTM的pytorch实现

1.相关函数介绍

class torch.nn.LSTM(args,*kwargs)

```
lstm = nn.LSTM(input_size=10, hidden_size=20,num_layers= 2, bidirectional=False)
```

- 参数说明：
  - input_size: 输入的特征维度
  - output_size: 输出的特征维度

    - num_layers: 层数（注意与时序展开区分）
    - bias 隐层状态是否带bias，默认为`True`
    - bidirectional: 如果为`True`，为双向LSTM。默认为`False`
- LSTM的输入：input,(h0,c0)
    - input(seq_len,batch,input_size): 包含输入特征的`tensor`,注意输入是`tensor`。
    - h0(num_layers · num_directions,batch,hidden_size): 保存初始化隐藏层状态的`tensor`
    - c0(num_layers · num_directions,batch,hidden_size): 保存初始化细胞状态的`tensor`
- LSTM的输出： output,(hn,cn)
    - output(seq_len, batch, hidden_size * num_directions): 保存`RNN`最后一层输出的`tensor`
    - hn(num_layers * num_directions,batch,hidden_size): 保存`RNN`最后一个时间步隐藏状态的`tensor`
    - cn(num_layers * num_directions,batch,hidden_size): 保存`RNN`最后一个时间步细胞状态的`tensor`

class torch.nn.Linear()

```
class torch.nn.Linear(in_features,out_features,bias = True)
```

- 作用：对输入数据做线性变换。y=Ax+by=Ax+b
- 参数：
    - in_features：每个输入样本的大小
    - out_features: 每个输出样本的大小
    - bias: 默认值为True。是否学习偏置。
- 形状：
    - 输入： (N,in_features)
    - 输出： (N,out_features)
- 变量：
    - weights: 可学习的权重，形状为(in_features,out_features)
    - bias: 可学习的偏置，形状为(out_features)

2.简单的小例子

```
# _*_ coding:utf-8 _*_
# 开发人员:未央
# 开发时间:2020/4/21 10:31
# 文件名:lstm_test1.py
# 开发工具:PyCharm
# https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html#lstm-s-in-pytorch
# Author: Robert Guthrie

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)


lstm = nn.LSTM(3, 3)  # Input dim is 3, output dim is 3
inputs = [torch.randn(1, 3) for _ in range(5)]  # make a sequence of length 5
```

```
# initialize the hidden state.
hidden = (torch.randn(1, 1, 3),
          torch.randn(1, 1, 3))
for i in inputs:
    # Step through the sequence one element at a time.
    # after each step, hidden contains the hidden state.
    out, hidden = lstm(i.view(1, 1, -1), hidden)

# alternatively, we can do the entire sequence all at once.
# the first value returned by LSTM is all of the hidden states throughout
# the sequence. the second is just the most recent hidden state
# (compare the last slice of "out" with "hidden" below, they are the same)
# The reason for this is that:
# "out" will give you access to all hidden states in the sequence
# "hidden" will allow you to continue the sequence and backpropagate,
# by passing it as an argument  to the lstm at a later time
# Add the extra 2nd dimension
inputs = torch.cat(inputs).view(len(inputs), 1, -1)
hidden = (torch.randn(1, 1, 3), torch.randn(1, 1, 3))  # clean out hidden state
out, hidden = lstm(inputs, hidden)
print(out)
print("*"*20)
print(hidden)
```

运行后输出如下：

```
tensor([[[-0.0187,  0.1713, -0.2944]],
        [[-0.3521,  0.1026, -0.2971]],
        [[-0.3191,  0.0781, -0.1957]],
        [[-0.1634,  0.0941, -0.1637]],
        [[-0.3368,  0.0959, -0.0538]]], grad_fn=<StackBackward>)
********************
(tensor([[[-0.3368,  0.0959, -0.0538]]], grad_fn=<StackBackward>),
tensor([[[-0.9825,  0.4715, -0.0633]]], grad_fn=<StackBackward>))
```

Write by sheen