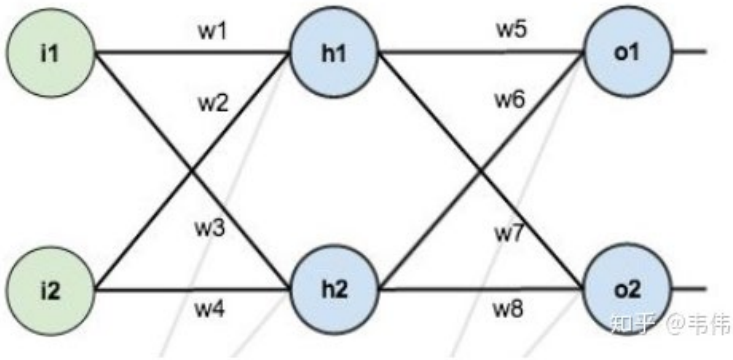
从反向传播推导到梯度消失 and 爆炸的原因及解决方案（从 DNN 到 RNN，内附详细反向传播公式推导）

（https://zhuanlan.zhihu.com/p/76772734）

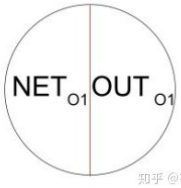
一、 概述

想要真正了解梯度爆炸和消失问题，必须手推反向传播，了解反向传播里梯度更新到底是怎么样更新的，所有问题都需要用数学表达式来说明，经过手推之后，便可分析出是什么原因导致的。本人就是在手推之后，才真正了解了这个问题发生的本质，所以本文以手推反向传播开始。

二、 手推反向传播



以上图为例开始推起来，先说明几点，i1，i2 是输入节点，h1，h2 为隐藏层节点，o1，o2 为输出层节点，除了输入层，其他两层的节点结构为下图所示：



举例说明，NET_{o1} 为输出层的输入，也就是隐藏层的输出经过线性变换后的值，OUT_{o1} 为

经过激活函数 sigmoid 后的值；同理 NET_{h1} 为隐藏层的输入，也就是输入层经过线性变换后的值，OUT_{h1} 为经过激活函数 sigmoid 的值。只有这两层有激活函数，输入层没有。关于 sigmoid 函数和 sigmoid 函数的导数如下所示：

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\begin{aligned}\sigma'(z) &= \left(\frac{1}{1+e^{-z}}\right)' \\ &= \frac{e^{-z}}{(1+e^{-z})^2} \\ &= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\ &= \frac{\sigma(z)}{(1+e^{-z})^2} \\ &= \sigma(z)(1-\sigma(z))\end{aligned}$$

定义一下损失函数，这里的损失函数是均方误差函数，即：

$$Loss_{total} = \sum \frac{1}{2}(\text{target - output})^2$$

具体到上图，就是：

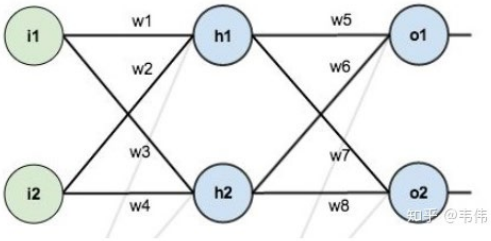
$$Loss_{total} = \frac{1}{2}(\text{target1 - out\_o1})^2 + \frac{1}{2}(\text{target2 - out\_o2})^2$$

到这里，所有前提就交代清楚了，前向传播就不推了，默认大家都会，下面推反向传播。

第一个反向传播（热身）：

先来一个简单的热热身，求一下损失函数对 W5 的偏导，即：

$$\frac{\partial Loss_{total}}{\partial w_5}$$



首先根据链式求导法则写出对 W5 求偏导的总公式，再把图拿下来对照（如上），可以看出，需要计算三部分的求导，下面就一步一步来：

$$\text{总公式：}\ \frac{\partial Loss_{total}}{\partial w_5} = \frac{\partial Loss_{total}}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial net_{o1}} \frac{\partial net_{o1}}{\partial w_5}$$

$$\text{第一步：}\ \frac{\partial Loss_{total}}{\partial out_{o1}} = \frac{\partial \frac{1}{2}(target_1 - out_{o1})^2 + \frac{1}{2}(target_2 - out_{o2})^2}{\partial out_{o1}} = out_{o1} - target_1$$

$$\text{第二步：}\ \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial \frac{1}{1+e^{-net_{o1}}}}{\partial net_{o1}} = \sigma(net_{o1})(1 - \sigma(net_{o1}))$$

$$\text{第三步：}\ \frac{\partial net_{o1}}{\partial w_5} = \frac{\partial out_{h1} w_5 + out_{h2} w_6}{\partial w_5} = out_{h_1}$$

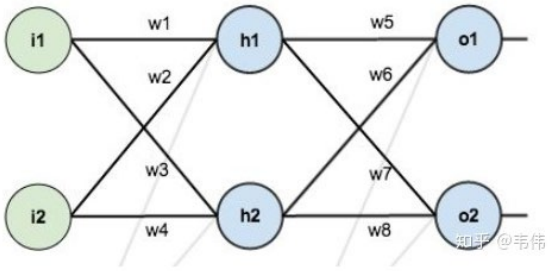
综上三个步骤，得到总公式：

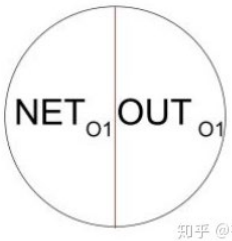
$$\text{总公式：}\ \frac{\partial Loss_{total}}{\partial w_5} = (out_{o1} - target_1) \cdot (\sigma(net_{o1})(1 - \sigma(net_{o1}))) \cdot out_{h_1}$$

第二个反向传播：

接下来，要求损失函数对 w1 的偏导，即：

$$\frac{\partial Loss_{total}}{\partial w_1}$$





还是把图摆在这，方便看，先写出总公式，对 w1 求导有个地方要注意，w1 的影响不仅来自 o1 还来自 o2，从图上可以一目了然，所以总公式为：

$$\text{总公式：}\ \frac{\partial Loss_{total}}{\partial w_1} = \frac{\partial Loss_{total}}{\partial out_{o1}}\frac{\partial out_{o1}}{\partial net_{o1}}\frac{\partial net_{o1}}{\partial out_{h1}}\frac{\partial out_{h1}}{\partial net_{h1}}\frac{\partial net_{h1}}{\partial w_1} + \frac{\partial Loss_{total}}{\partial out_{o2}}\frac{\partial out_{o2}}{\partial net_{o2}}\frac{\partial net_{o2}}{\partial out_{h1}}\frac{\partial out_{h1}}{\partial net_{h1}}\frac{\partial net_{h1}}{\partial w_1}$$

所以总共分为左右两个式子，分别又对应 5 个步骤，详细写一下左边，右边同理：

$$\text{第一步：}\ \frac{\partial Loss_{total}}{\partial out_{o1}} = out_{o1} - target_1$$

$$\text{第二步：}\ \frac{\partial out_{o1}}{\partial net_{o1}} = \sigma(net_{o1})(1-\sigma(net_{o1}))$$

$$\text{第三步：}\ \frac{\partial net_{o1}}{\partial out_{h1}} = \frac{\partial out_{h1}w_5 + out_{h2}w_6}{\partial out_{h1}} = w_5$$

$$\text{第四步：}\ \frac{\partial out_{h1}}{\partial net_{h1}} = \sigma(net_{h1})(1-\sigma(net_{h1}))$$

$$\text{第二步：}\ \frac{\partial net_{h1}}{\partial w_1} = \frac{\partial i_1 w_1 + i_2 w_2}{\partial w_1} = i_1$$

右边也是同理，就不详细写了，写一下总的公式：

$$\begin{aligned}\frac{\partial Loss_{total}}{\partial w_1} &= ((out_{o1} - target_1)\cdot(\sigma(net_{o1})(1-\sigma(net_{o1})))\cdot w_5\cdot(\sigma(net_{h1})(1-\sigma(net_{h1})))\cdot i_1) \\ &+ ((out_{o2} - target_2)\cdot(\sigma(net_{o2})(1-\sigma(net_{o2})))\cdot w_7\cdot(\sigma(net_{h1})(1-\sigma(net_{h1})))\cdot i_1)\end{aligned}$$

这个公式只是对如此简单的一个网络结构的一个节点的偏导为了后面描述方便，把上面的

公式化简一下：

$out_{o1} - target_1$ 记为 C_{o1} ，

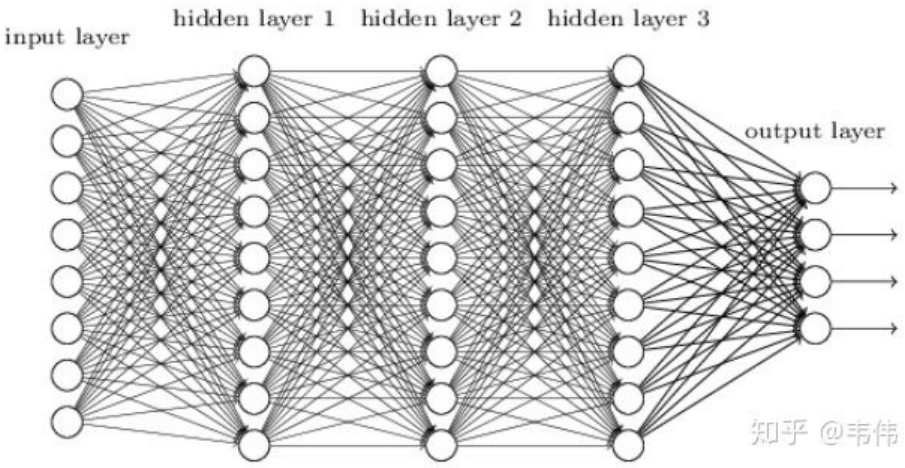
$\sigma(net_{o1})(1-\sigma(net_{o1}))$ 记为 $\sigma(net_{o1})'$ ，

$$\frac{\partial Loss_{total}}{\partial w_1} = C_{o1} \cdot \sigma(net_{o1})' \cdot w_5 \cdot \sigma(net_{h1})' \cdot i_1 + C_{o2} \cdot \sigma(net_{o2})' \cdot w_7 \cdot \sigma(net_{h1})' \cdot i_1$$

三、 梯度消失，爆炸产生原因

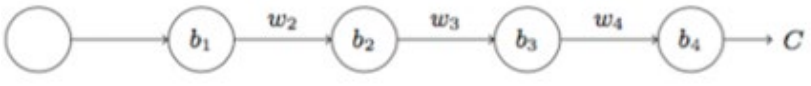
从上式其实已经能看出来，求和操作其实不影响，主要是是看乘法操作就可以说明问题，可以看出，损失函数对 w1 的偏导，与 Co ，权重 w，sigmoid 的导数有关，明明还有输入 i 为什么不提？因为如果是多层神经网络的中间某层的某个节点，那么就没有输入什么事了。所以产生影响的就是刚刚提的三个因素。

再详细点描述，如图，多层神经网络：



参考：https://zhuanlan.zhihu.com/p/25631496

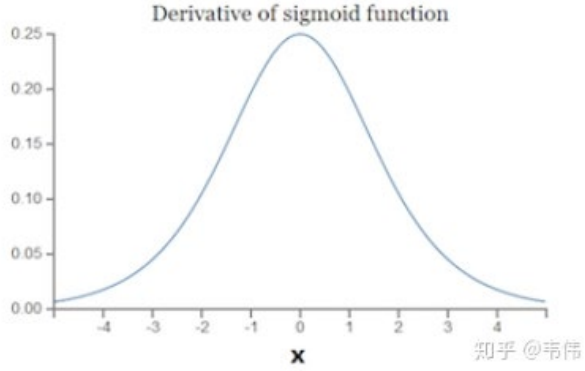
假设每一层只有一个神经元，其中 σ 为 sigmoid 函数，如图：



则反向传播的求导公式如下：

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$
$$= C_{y4}\sigma'(z_4)\, w_4\sigma'(z_3)\, w_3\sigma'(z_2)\, w_2\sigma'(z_1)$$

我们先看一下 sigmoid 函数的导数的样子：



发现 sigmoid 函数求导后最大也只能是 0.25。

再来看 W，一般我们初始化权重参数 W 时，通常都小于 1，用的最多的应该是 0，1 正态分布。所以 $|\sigma'(z)\, w| \leq 0.25$ ，多个小于 1 的数连乘之后，那将会越来越小，导致靠近输入层的层的权重的偏导几乎为 0，也就是说几乎不更新，这就是梯度消失的根本原因。

再来看看梯度爆炸的原因，也就是说如果 $|\sigma'(z)\, w| \geq 1$ 时，连乘下来就会导致梯度过大，导致梯度更新幅度特别大，可能会溢出，导致模型无法收敛。sigmoid 的函数是不可能大于 1 了，上图看的很清楚，那只能是 w 了。但梯度爆炸的情况一般不会发生，对于 sigmoid 函来说， $\sigma(z)'$ 的大小也与 w 有关，因为 $z = wx + b$ ，除非该层的输入值在一直一个比较小的范围内。其实梯度爆炸和梯度消失问题都是因为网络太深，网络权值更新不稳定造成的，本质上是因为梯度反向传播中的连乘效应。 所以，总结一下，为什么会发生梯度爆炸和消失：本质上是因为神经网络的更新方法，梯度消失是因为反向传播过程中对梯度的求解会产生 sigmoid 导数和参数的连乘，sigmoid 导数的最大值为 0.25，权重一般初始都在 0，1 之间，乘积小于 1，多层的话就会有多个小于 1 的值连乘，导致靠近输入层的梯度几乎为 0，得不到更新。梯度爆炸是也是同样的原因，只是如果初始权重大于 1，或者更大一些，多个大于 1 的值连乘，将会很大或溢出，导致梯度更新过大，模型无法收敛。

四. 梯度消失，爆炸解决方案：

参考：https://zhuanlan.zhihu.com/p/33006526

解决方案一（预训练加微调）：

此方法来自 Hinton 在 2006 年发表的一篇论文，Hinton 为了解决梯度的问题，提出采取无监督逐层训练方法，其基本思想是每次训练一层隐节点，训练时将上一层隐节点的输出作为输入，而本层隐节点的输出作为下一层隐节点的输入，此过程就是逐层“预训练”（pre-training）；在预训练完成后，再对整个网络进行“微调”（fine-tunning）。Hinton 在训练深度信念网络（Deep Belief Networks 中，使用了这个方法，在各层预训练完成后，再利用 BP 算法对整个网络进行训练。此思想相当于是先寻找局部最优，然后整合起来寻找全局最优，此方法有一定的好处，但是目前应用的不是很多了。

解决方案二（梯度剪切、正则）：

梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。正则化是通过对网络权重做正则限制过拟合，仔细看正则项在损失函数的形式：

$$Loss = (y - W^T x)^2 + \alpha ||W||^2$$

其中，α 是指正则项系数，因此，如果发生梯度爆炸，权值的范数就会变的非常大，通过正则化项，可以部分限制梯度爆炸的发生。

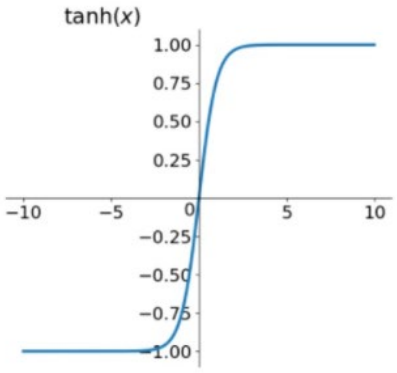
注：事实上，在深度神经网络中，往往是梯度消失出现的更多一些

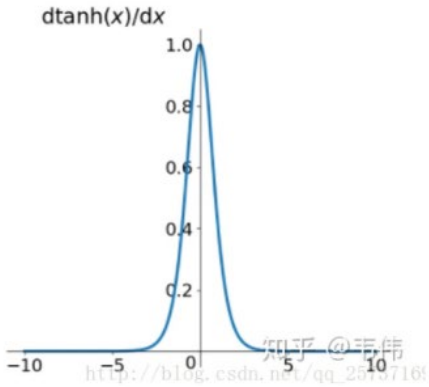
解决方案三（改变激活函数）：

首先说明一点，tanh 激活函数不能有效的改善这个问题，先来看 tanh 的形式：

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

再来看 tanh 的导数图像：



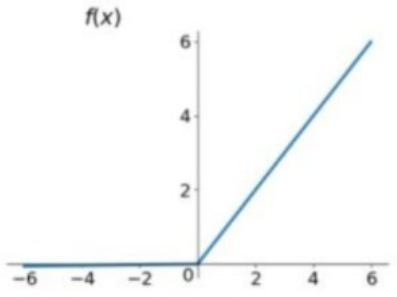


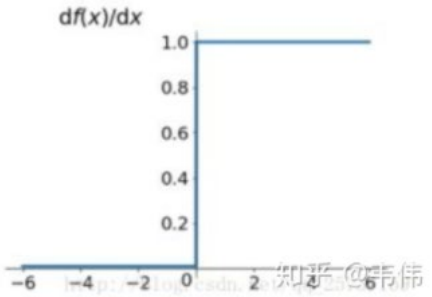
发现虽然比 sigmoid 的好一点，sigmoid 的最大值小于 0.25，tanh 的最大值小于 1，但仍是小于 1 的，所以并不能解决这个问题。

Relu:

思想也很简单，如果激活函数的导数为 1，那么就不存在梯度消失爆炸的问题了，每层的网络都可以得到相同的更新速度，relu 就这样应运而生。先看一下 relu 的数学表达式：

$$\mathrm{Re\,lu(x)} = \max(\mathrm{x}, 0) = \left\{ \begin{matrix} 0, x < 0 \\ x, x > 0 \end{matrix} \right\}$$





从上图中，我们可以很容易看出，relu 函数的导数在正数部分是恒等于 1 的，因此在深层网络中使用 relu 激活函数就不会导致梯度消失和爆炸的问题。

relu 的主要贡献在于：

- 解决了梯度消失、爆炸的问题
- 计算方便，计算速度快
- 加速了网络的训练

同时也存在一些缺点：

- 由于负数部分恒为 0，会导致一些神经元无法激活（可通过设置小学习率部分解决）

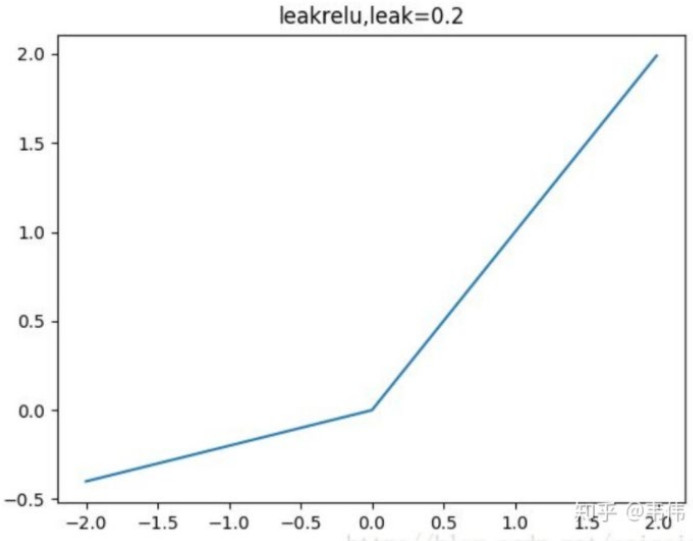
- 输出不是以 0 为中心的

Leakrelu：

leakrelu 就是为了解决 relu 的 0 区间带来的影响，其数学表达为：

$$leakrelu = f(x) = \begin{cases} x, & x > 0 \\ x * k, & x \leq 0 \end{cases}$$

其中 k 是 leak 系数，一般选择 0.1 或者 0.2，或者通过学习而来解决死神经元的问题。



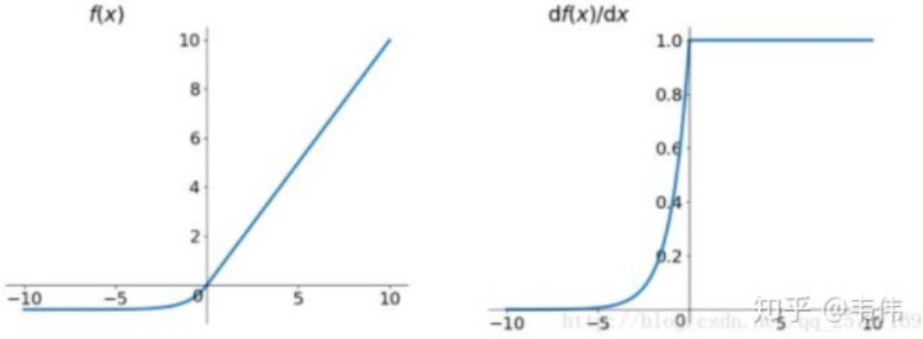
leakrelu 解决了 0 区间带来的影响，而且包含了 relu 的所有优点。

Elu：

elu 激活函数也是为了解决 relu 的 0 区间带来的影响，其数学表达为：

$$\begin{cases} x, & \text{if } x > 0 \\ \alpha\left(e^{x} - 1\right), & \text{otherwise} \end{cases}$$

其函数及其导数数学形式为：



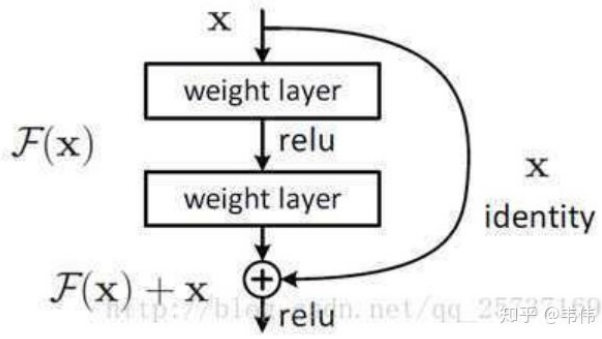
但是 elu 相对于 leakrelu 来说，计算要更耗时间一些，因为有 e。

解决方案四（batchnorm）：

Batchnorm 是深度学习发展以来提出的最重要的成果之一了，目前已经被广泛的应用到了各大网络中，具有加速网络收敛速度，提升训练稳定性的效果，Batchnorm 本质上是解决反向传播过程中的梯度问题。batchnorm 全名是 batch normalization，简称 BN，即批规范化，通过规范化操作将输出信号 x 规范化到均值为 0，方差为 1 保证网络的稳定性。

具体的 batchnorm 原理非常复杂，在这里不做详细展开，此部分大概讲一下 batchnorm 解决梯度的问题上。batchnorm 就是通过对每一层的输出做 scale 和 shift 的方法，通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到正太分布，即严重偏离的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，使得让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度。

解决方案五（残差结构）：



如图，把输入加入到某层中，这样求导时，总会有个 1 在，这样就不会梯度消失了。

$$\frac{\partial \text{loss}}{\partial x_l} = \frac{\partial \text{loss}}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_l} = \frac{\partial \text{loss}}{\partial x_L} \cdot \left(1 + \frac{\partial}{\partial x_L} \sum_{i=l}^{L-1} F(x_i, W_i)\right)$$

式子的第一个因子表示的损失函数到达的梯度，小括号中的 1 表明短路机制可以无损地传播梯度，而另外一项残差梯度则需要经过带有 weights 的层，梯度不是直接传递过来的。残差梯度不会那么巧全为-1，而且就算其比较小，有 1 的存在也不会导致梯度消失。所以残差学习会更容易。

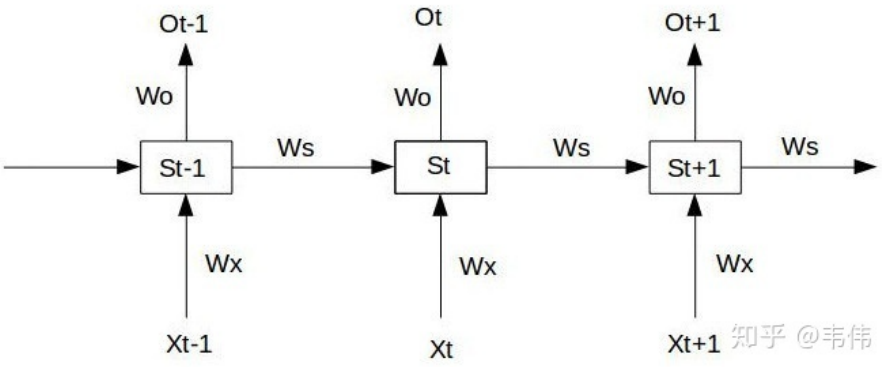
注：上面的推导并不是严格的证，只为帮助理解。

解决方案六（LSTM）：

在介绍这个方案之前，有必要来推导一下 RNN 的反向传播，因为关于梯度消失的含义它跟 DNN 不一样！

参考 https://zhuanlan.zhihu.com/p/28687529

RNN 结构如图：



假设我们的时间序列只有三段，神经元没有激活函数，则 RNN 最简单的前向传播过程如下：

$$S_1 = W_x X_1 + W_s S_0 + b_1 \; O_1 = W_o S_1 + b_2$$

$$S_2 = W_x X_2 + W_s S_1 + b_1 \; O_2 = W_o S_2 + b_2$$

$$S_3 = W_x X_3 + W_s S_2 + b_1 \; O_3 = W_o S_3 + b_2$$

假设在 t=3 时刻，损失函数为：

$$L_3 = \frac{1}{2}(Y_3 - O_3)^2$$

则对于一次训练任务的损失函数为：

$$L = \sum_{t=0}^{T} L_t$$

即每一时刻损失值的累加。

现在假设我们我们的时间序列只有三段，t1，t2，t3。

我们只对 t3 时刻的 W_x、W_s、W_0 求偏导（其他时刻类似）：

$$\frac{\partial L_3}{\partial W_0} = \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial W_o}$$

$$\frac{\partial L_3}{\partial W_x} = \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial W_x} + \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial S_2}\frac{\partial S_2}{\partial W_x} + \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial S_2}\frac{\partial S_2}{\partial S_1}\frac{\partial S_1}{\partial W_x}$$

$$\frac{\partial L_3}{\partial W_s} = \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial W_s} + \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial S_2}\frac{\partial S_2}{\partial W_s} + \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial S_2}\frac{\partial S_2}{\partial S_1}\frac{\partial S_1}{\partial W_s}$$

可以看出对于 W0 求偏导并没有长期依赖，但是对于 Wx 和 Ws 求偏导，会随着时间序列产生长期依赖。因为 St 随着时间序列向前传播，而 St 又是 Wx 和 Ws 的函数。

根据上述求偏导的过程，我们可以得出任意时刻对 Wx 和 Ws 求偏导的公式：

$$\frac{\partial L_t}{\partial W_x} = \sum_{k=0}^{t}\frac{\partial L_t}{\partial O_t}\frac{\partial O_t}{\partial S_t}\left(\prod_{j=k+1}^{t}\frac{\partial S_j}{\partial S_{j-1}}\right)\frac{\partial S_k}{\partial W_x}$$

任意时刻对 Ws 求偏导的公式同上。如果加上激活函数：

$$S_j = tanh(W_x X_j + W_s S_{j-1} + b_1)$$

$$\prod_{j=k+1}^{t}\frac{\partial S_j}{\partial S_{j-1}} = \prod_{j=k+1}^{t} tanh' W_s$$

激活函数 tanh 和它的导数图像在上面已经说过了，所以原因在这就不赘述了，还是一样的，激活函数导数小于 1。现在来解释一下，为什么说 RNN 和 DNN 的梯度消失问题含义不一样？

先来说 DNN 中的反向传播：在上文的 DNN 反向传播中，我推导了两个权重的梯度，第一个梯度是直接连接着输出层的梯度，求解起来并没有梯度消失或爆炸的问题，因为它没有连乘，只需要计算一步。第二个梯度出现了连乘，也就是说越靠近输入层的权重，梯度消失或爆炸的问题越严重，可能就会消失会爆炸。一句话总结一下，DNN 中各个权重的梯度是独立的，该消失的就会消失，不会消失的就不会消失。

再来说 RNN：RNN 的特殊性在于，它的权重是共享的。抛开 Wo 不谈，因为它在某时刻的梯度不会出现问题（某时刻并不依赖于前面的时刻），但是 Ws 和 Wx 就不一样了，每一时刻都由前面所有时刻共同决定，是一个相加的过程，这样的话就有个问题，当距离长了，计算最前面的导数时，最前面的导数就会消失或爆炸，但当前时刻整体的梯度并不会消失，因

为它是求和的过程，当下的梯度总会在，只是前面的梯度没了，但是更新时，由于权值共享，所以整体的梯度还是会更新，通常人们所说的梯度消失就是指的这个，指的是当下梯度更新时，用不到前面的信息了，因为距离长了，前面的梯度就会消失，也就是没有前面的信息了，但要知道，整体的梯度并不会消失，因为当下的梯度还在，并没有消失。一句话概括：RNN 的梯度不会消失，RNN 的梯度消失指的是当下梯度用不到前面的梯度了，但 DNN 靠近输入的权重的梯度是真的会消失。

说完了 RNN 的反向传播及梯度消失的含义，终于该说为什么 LSTM 可以解决这个问题了，这里默认大家都懂 LSTM 的结构，对结构不做过多的描述。

参考：https://www.zhihu.com/question/34878706

1. LSTM 中梯度的传播有很多条路径，

$$c_{t-1} \rightarrow c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

这条路径上只有逐元素相乘和相加的操作，梯度流最稳定；但是其他路径例如

$$c_{t-1} \rightarrow h_{t-1} \rightarrow i_t \rightarrow c_t$$

梯度流与普通 RNN 类似，照样会发生相同的权重矩阵反复连乘。

2. LSTM 刚提出时没有遗忘门，这时候在 $c_{t-1} \rightarrow c_t$ 直接相连的短路路径上，的梯度畅通无阻，不会消失。类似于 ResNet 中的残差连接。

3. 但是在其他路径上，LSTM 的梯度流和普通 RNN 没有太大区别，依然会爆炸或者消失。由于总的远距离梯度 = 各条路径的远距离梯度之和，即便其他远距离路径梯度消失了，只要保证有一条远距离路径（就是上面说的那条高速公路）梯度不消失，总的远距离梯度就不会消失（正常梯度 + 消失梯度 = 正常梯度）。因此 LSTM 通过改善一条路径上的梯度问题拯救了总体的远距离梯度。

4. 同样，因为总的远距离梯度 = 各条路径的远距离梯度之和，高速公路上梯度流比较稳定，但其他路径上梯度有可能爆炸，此时总的远距离梯度 = 正常梯度 + 爆炸梯度 = 爆炸梯度，因此 LSTM 仍然有可能发生梯度爆炸。不过，由于 LSTM 的其他路径非常崎岖，和普通 RNN 相比多经过了很多次激活函数（导数都小于 1），因此 LSTM 发生梯度爆炸的频率要低得多。实践中梯度爆炸一般通过梯度裁剪来解决。

5. 对于现在常用的带遗忘门的 LSTM 来说，4 中的分析依然成立，而 3 分为两种情况：其一是遗忘门接近 1（例如模型初始化时会把 forget bias 设置成较大的正数，让遗忘门饱和），这时候远距离梯度不消失；其二是遗忘门接近 0，但这时模型是故意阻断梯度流的，

这不是 bug 而是 feature（例如情感分析任务中有一条样本 “A，但是 B”，模型读到“但是”后选择把遗忘门设置成 0，遗忘掉内容 A，这是合理的）。当然，常常也存在 f 介于 [0, 1] 之间的情况，在这种情况下只能说 LSTM 改善（而非解决）了梯度消失的状况。