RNN 之 BPTT 理论推导

一、RNN (循环神经网络)

RNN 是一种具有长时记忆能力的神经网络模型，被广泛用于序列标注问题。序列标注问题中，模型的输入是一段时间序列，记为

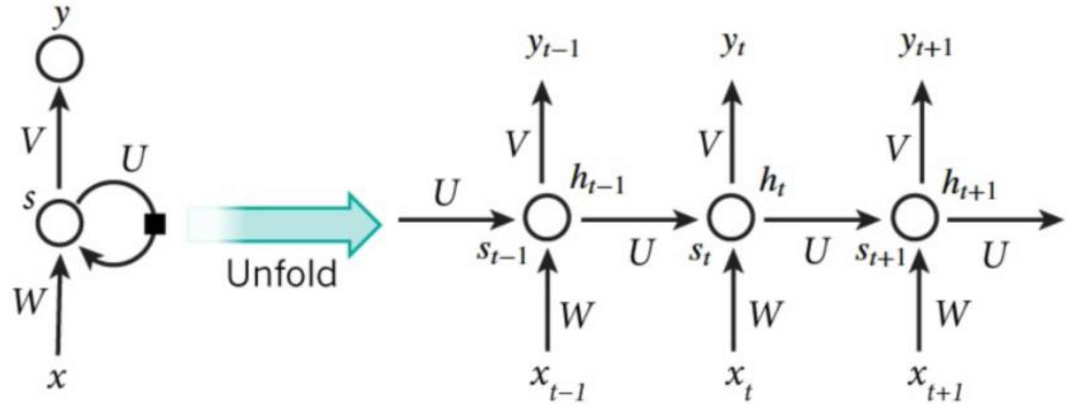
$$\bar{x} = \{x_1, x_2, ..., x_T\}$$

我们的目标是为输入序列的每个元素打上标签集合中的对应标签，记为

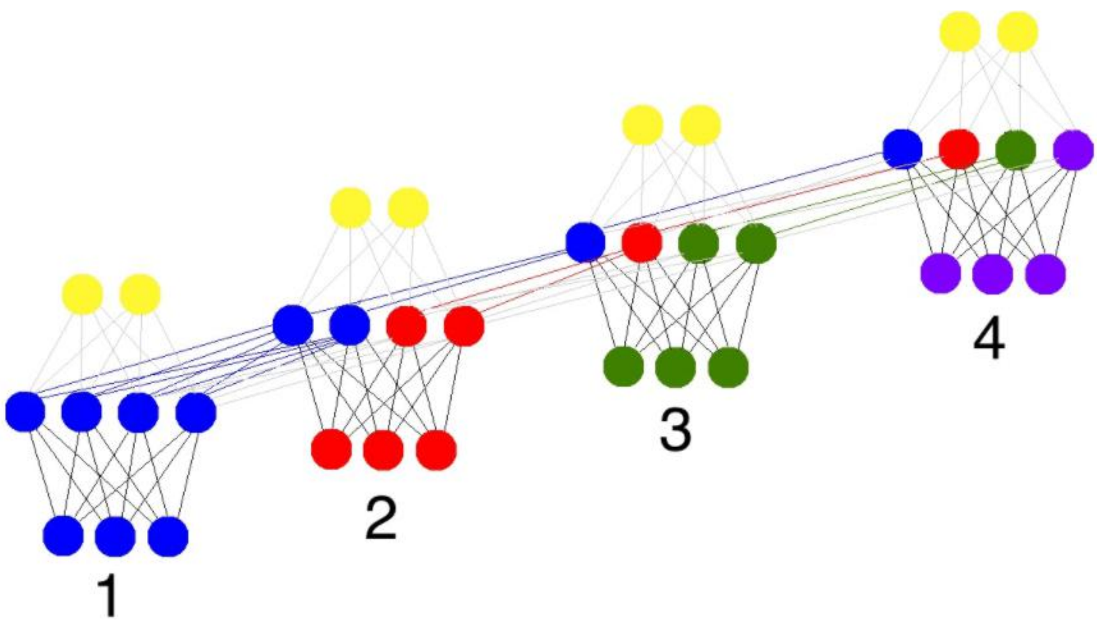
$$\bar{y} = \{y_1, y_2, ..., y_T\}$$

二、RNN 一般结构图

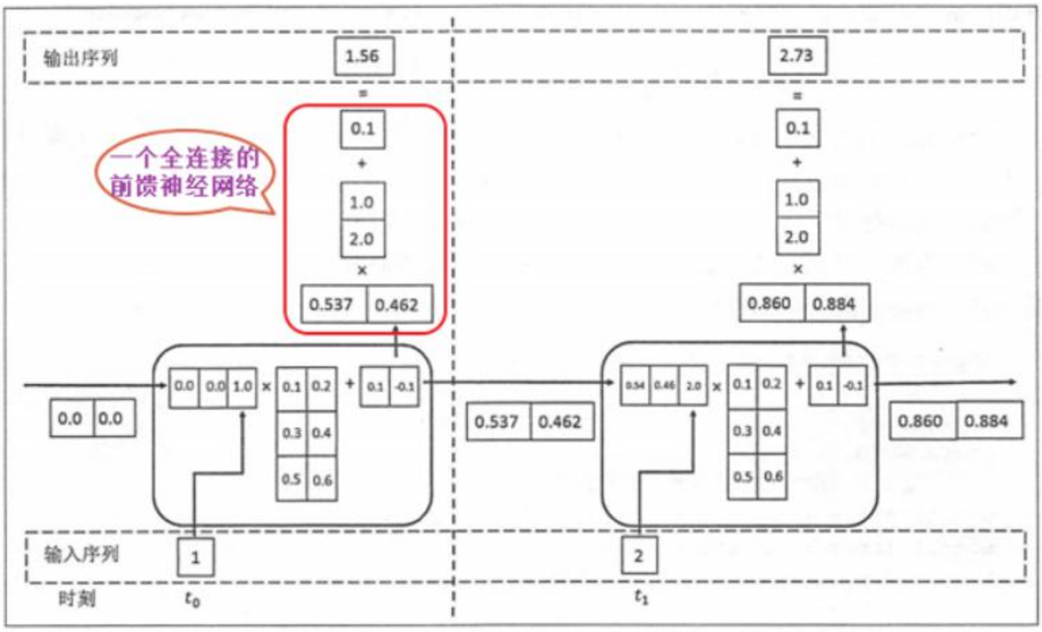
1. 符号描述



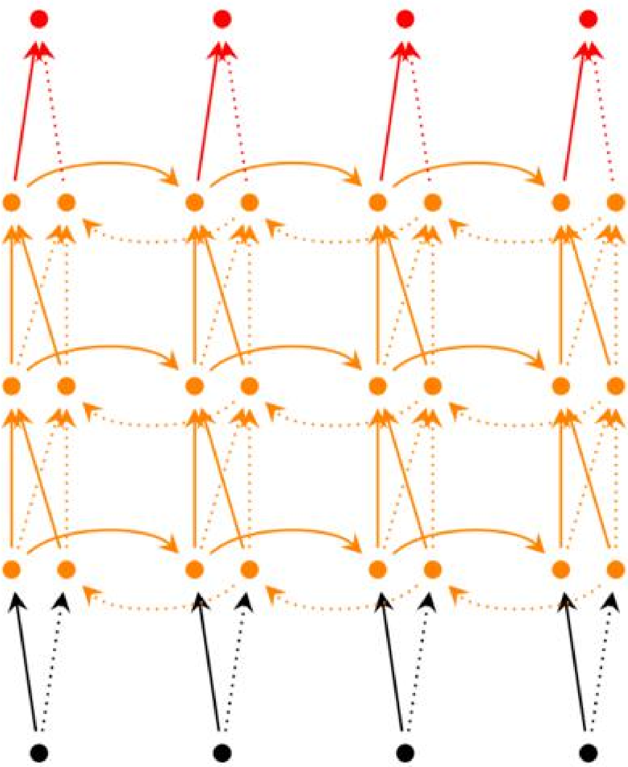
2. 节点描述



3. RNN 前向传播的具体计算过程



4. 深度 RNN（包含多个隐藏层）示意图



三、符号说明

符号	解释
K	词汇表的大小
T	句子的长度
H	隐藏层单元数
$\mathbf{x}=\{x_1,x_2,...,x_T\}$	句子的单词序列
$x_t\in\mathbb{R}^{K\times1}$	第t个时刻RNN的输入，one-hot vector
$\hat{y}_t\in\mathbb{R}^{K\times1}$	第t时刻softmax层的输出，估计每个词出现的概率
$y_t\in\mathbb{R}^{K\times1}$	第t时刻的label，为每个词出现的概率，one-hot vector
E_t	第t个时刻（第t个word）的损失函数,定义为交叉熵误差$E_t=-y_t^T log(\hat{y}_t)$
E	一个句子的损失函数，由各个时刻（即每个word）的损失函数组成,$E=\sum_t^T E_t$（注：由于我们要推导的是SGD算法，更新梯度是相对于一个训练样例而言的，因此我们一次只考虑一个句子的误差，而不是整个训练集的误差（对应BGD算法））
$s_t\in\mathbb{R}^{H\times1}$	第t个时刻RNN隐藏层的输入
$h_t\in\mathbb{R}^{H\times1}$	第t个时刻RNN隐藏层的输出
$z_t\in\mathbb{R}^{K\times1}$	输出层的汇集输入
$r_t=\hat{y}_t-y_t$	残差向量
$W\in\mathbb{R}^{H\times K}$	从输入层到隐藏层的权值
$U\in\mathbb{R}^{H\times H}$	隐藏层上一个时刻到当前时刻的权值
$V\in\mathbb{R}^{K\times H}$	隐藏层到输出层的权值

四、RNN 的 BPTT

RNN 中上述符号之间关系如下：

$$\begin{cases} s_t = Uh_{t-1} + Wx_t + b \\ h_t = \tanh(s_t) \\ z_t = Vh_t + c \\ \hat{y}_t = soft\max(z_t) \\ E_t = -y_t^T \log(\hat{y}_t) \\ E = \sum_t^T E_t \end{cases}$$

U、V、W、b、c 为 RNN 共享参数，BPTT 目标是求

$$\frac{\partial E}{\partial U},\frac{\partial E}{\partial V},\frac{\partial E}{\partial W},\frac{\partial E}{\partial b},\frac{\partial E}{\partial c}$$

又因为

$$\frac{\partial E}{\partial U}=\sum_t \frac{\partial E_t}{\partial U}$$

只需对每个时刻的损失函数求偏导再相加即可

五、U、W、V、b、c 偏导

首先参考文献[1]定义 prod 运算符，对输入或输出 X，Y，Z 为任意形状张量的函数 Y=f(X)和 Z=g(Y)，通过链式法则，我们有

$$\frac{\partial Z}{\partial X}=prod(\frac{\partial Z}{\partial Y},\frac{\partial Y}{\partial X})$$

其中 prod 运算符将根据两个输入的形状，在必要的操作（如转置和互换输入位置）后对两个输入做乘法。

1. 对 V 求偏导

E_t 为标量，V 为 $K\times H$ 权值矩阵，即标量对矩阵求偏导

$$\frac{\partial E_t}{\partial V}=prod(\frac{\partial E_t}{\partial z_t},\frac{\partial z_t}{\partial V})=(\hat{y}_t-y_t)\otimes h_t$$

$$\frac{\partial E}{\partial V}=\sum_{t=1}^{T}\frac{\partial E_t}{\partial V}=\sum_{t=1}^{T}(\hat{y}_t-y_t)\otimes h_t$$

$\otimes$ 为向量外积，对于 $\frac{\partial E_t}{\partial z_t}=(\hat{y}_t-y_t)$ 推导细节见附录 1。

2. 对 c 求偏导

$$\frac{\partial E_t}{\partial c}=\frac{\partial E_t}{\partial z_t}\frac{\partial z_t}{\partial c}=\hat{y}_t-y_t$$

$$\frac{\partial E}{\partial c}=\sum_{t=1}^{T}(\hat{y}_t-y_t)$$

3. 对 U、W、b 求偏导

U、W、b 虽然参数共享，但是他们不仅对 t 时刻输出有贡献，同时对 t+1 时刻隐藏层的输入 s_{t+1} 有贡献。所以，以 U 为例有

$$\frac{\partial E_t}{\partial U}=\sum_{k=0}^{t}\frac{\partial s_k}{\partial U}\frac{\partial E_t}{\partial s_k}$$

对 t 时刻的 U、W、b 求导，利用链式法则可得

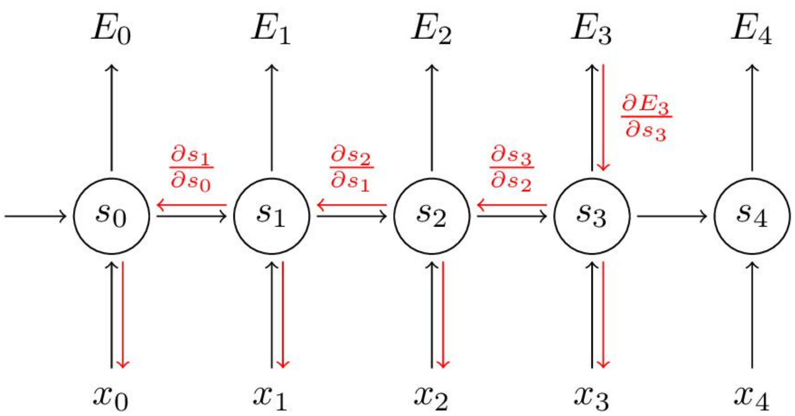
$$\frac{\partial E}{\partial W}=\frac{\partial E}{\partial s_t}\frac{\partial s_t}{\partial W}$$

$$\frac{\partial E}{\partial U}=\frac{\partial E}{\partial s_t}\frac{\partial s_t}{\partial U}$$

$$\frac{\partial E}{\partial b}=\frac{\partial E}{\partial s_t}\frac{\partial s_t}{\partial b}$$

以上均需求出 $\frac{\partial E}{\partial s_t}$，根据如下传递关系

$$s_k \to h_k \to s_{k+1} \to h_{k+1} \to \ldots \to E_t$$



上图隐藏了 h_t，简化了误差传递路径。

有如下关系

$$\begin{aligned}\delta_k &= \frac{\partial E_t}{\partial s_k}=\frac{\partial h_k}{\partial s_k}\frac{\partial s_{k+1}}{\partial h_k}\frac{\partial E_t}{\partial s_{k+1}} \\ &= diag(1-h_k \bullet h_k)U^T\delta_{k+1} \\ &= (U^T\delta_{k+1})\bullet(1-h_k \bullet h_k)\end{aligned}$$

$$\begin{aligned}\delta_t &= \frac{\partial E_t}{\partial s_t}=\frac{\partial h_t}{\partial s_t}\frac{\partial z_t}{\partial h_t}\frac{\partial E_t}{\partial z_t} \\ &= diag(1-h_t \bullet h_t)V^T(\hat{y}_t-y_t) \\ &= (V^T(\hat{y}_t-y_t))\bullet(1-h_t \bullet h_t)\end{aligned}$$

其中 $\bullet$ 为向量点积，即得到如下递推关系

$$\delta_k = (U^T\delta_{k+1})\bullet(1-h_k \bullet h_k)$$

$$\delta_t = (V^T(\hat{y}_t-y_t))\bullet(1-h_t \bullet h_t)$$

通过δ_t可以推出$\delta_1, \delta_2, \delta_3, ..., \delta_t$，可推出如下

$$\frac{\partial E_t}{\partial U} = prod(\frac{\partial E_t}{\partial s_t}, \frac{\partial s_t}{\partial U}) = prod(\frac{\partial E_t}{\partial s_t}, h_{t-1}) = \sum_{k=0}^{k} \delta_k \otimes h_{t-1}$$

$$\frac{\partial E_t}{\partial W} = prod(\frac{\partial E_t}{\partial s_t}, \frac{\partial s_t}{\partial W}) = prod(\frac{\partial E_t}{\partial s_t}, x_t) = \sum_{k=0}^{k} \delta_k \otimes x_t$$

$$\frac{\partial E_t}{\partial b} = prod(\frac{\partial E_t}{\partial s_t}, \frac{\partial s_t}{\partial b}) = prod(\frac{\partial E_t}{\partial s_t}, \frac{\partial b}{\partial b}) = \sum_{k=0}^{k} \delta_k$$

得到

$$\frac{\partial E}{\partial U} = \sum_{t=0}^{T} \frac{\partial E_t}{\partial U} = \sum_{t=0}^{T} \sum_{k=0}^{t} \delta_k \otimes h_{t-1}$$

$$\frac{\partial E}{\partial W} = \sum_{t=0}^{T} \frac{\partial E_t}{\partial W} = \sum_{t=0}^{T} \sum_{k=0}^{t} \delta_k \otimes x_t$$

$$\frac{\partial E}{\partial b} = \sum_{t=0}^{T} \frac{\partial E_t}{\partial b} = \sum_{t=0}^{T} \sum_{k=0}^{t} \delta_k$$

参数更新

$$V := V - \lambda \sum_{t=1}^{T} (\hat{y}_t - y_t) \otimes h_t$$

$$U := U - \lambda \sum_{t=0}^{T} \sum_{k=0}^{t} \delta_k \otimes h_{t-1}$$

$$W := W - \lambda \sum_{t=0}^{T} \sum_{k=0}^{t} \delta_k \otimes x_t$$

$$b := b - \lambda \sum_{t=0}^{T} \sum_{k=0}^{t} \delta_k$$

$$c := c - \lambda \sum_{t=1}^{T} (\hat{y}_t - y_t)$$

其中$\delta_t = (V^T(\hat{y}_t - y_t)) \bullet (1 - h_t \bullet h_t)$

六、RNN 梯度爆炸或消失

以上推到可知

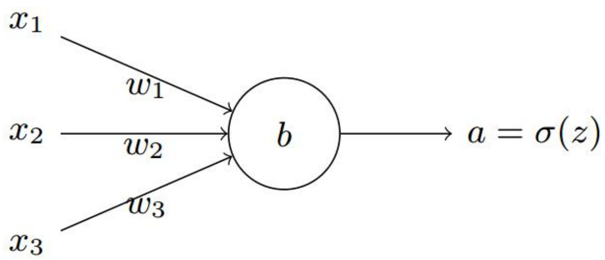
$$
\begin{aligned}
\delta_k &= (U^T\delta_{k+1})\bullet(1-h_k\bullet h_k) = (U^T(U^T\delta_{k+2})\bullet(1-h_{k+1}\bullet h_{k+1}))\bullet(1-h_k\bullet h_k) \\
&= \delta_t\prod_{i=k}^{t-1}U(1-h_{\mathrm{i}}\bullet h_{\mathrm{i}}) \\
&\le \delta_t(\beta_U\xi_{(1-h_{\mathrm{i}}\bullet h_{\mathrm{i}})})^{t-k} = \delta_t(\gamma)^{t-k}
\end{aligned}
$$

其中 $\beta_U,\xi_{(1-h_k\bullet h_k)}$ 为各自矩阵模的上界，γ 为 $\beta_U\xi_{(1-h_k\bullet h_k)}$，$t-k$ 较大时，如果 $\gamma>1$ 引起梯度爆炸，反之引起梯度消失。

附录 1. softmax 交叉熵损失函数求导

1，softmax 函数

神经网络分类中经常用到 softmax，如下神经元



其输出为

$$z_i = \sum_j w_{ij} x_{ij} + b$$

w_{ij} 是第 i 个神经元的第 j 个权重，b 是偏移量， z_i 表示该网络的第 i 个输出

$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

a_i 代表 softmax 的第 i 个输出值，右侧就是套用了 softmax 函数。

2，损失函数 loss function

损失函数可以有很多形式，这里用的是交叉熵函数，主要是由于这个求导结果比较简单，易于计算，并且交叉熵解决某些损失函数学习缓慢的问题。交叉熵的函数是这样的

$$C = -\sum_i y_i \ln a_i$$

其中 y_i 表示真实的分类结果

我们的目的是求 loss 对神经网络输出 z_i 的导数即

$$\frac{\partial C}{\partial z_i}$$

根据复合函数求导法则

$$\frac{\partial C}{\partial z_i} = \sum_j \left(\frac{\partial C_j}{\partial a_j}\frac{\partial a_j}{\partial z_i}\right)$$

注意：因为 softmax 公式的特性，它的分母包含了所有神经元的输出，所以对于不等于 i 的其他输出里面，也包含 z_i ，需要把所有 a 纳入计算范围。

对于 $\frac{\partial C_i}{\partial a_j}$ 有

$$\frac{\partial C_j}{\partial a_j} = \frac{\partial(-y_j \ln a_j)}{\partial a_j} = -y_j \frac{1}{a_j}$$

对于 $\frac{\partial a_j}{\partial z_i}$，分为两种情况

（1） $i = j$

$$\frac{\partial a_i}{\partial z_i} = \frac{\partial(\frac{e^{z_i}}{\sum_k e^{z_k}})}{\partial z_i} = \frac{\sum_k e^{z_k} e^{z_i} - (e^{z_i})^2}{(\sum_k e^{z_k})^2} = (\frac{e^{z_i}}{\sum_k e^{z_k}})(1 - \frac{e^{z_i}}{\sum_k e^{z_k}}) = a_i(1 - a_i)$$

（2） $i \neq j$

$$\frac{\partial a_j}{\partial z_i} = \frac{\partial(\frac{e^{z_j}}{\sum_k e^{z_k}})}{\partial z_i} = -e^{z_j}(\frac{1}{\sum_k e^{z_k}})^2 e^{z_i} = -a_i a_j$$

以上两种情况组合为

$$\begin{aligned}
\frac{\partial C}{\partial z_i} &= \sum_j (\frac{\partial C_j}{\partial a_j}\frac{\partial a_j}{\partial z_i}) = \sum_{j \neq i} (\frac{\partial C_j}{\partial a_j}\frac{\partial a_j}{\partial z_i}) + \sum_{j=i} (\frac{\partial C_j}{\partial a_j}\frac{\partial a_j}{\partial z_i}) \\
&= \sum_{j \neq i} -y_j \frac{1}{a_j}(-a_i a_j) + (-y_i \frac{1}{a_i})(a_i(1-a_i)) \\
&= \sum_{j \neq i} y_j a_i - y_i + y_i a_i \\
&= a_i \sum_j y_j - y_i
\end{aligned}$$

针对分类问题，我们给定的结果 y_i，最终只会有一个类别是 1，其他类别都是 0，因此，对于分类问题，进一步化简为

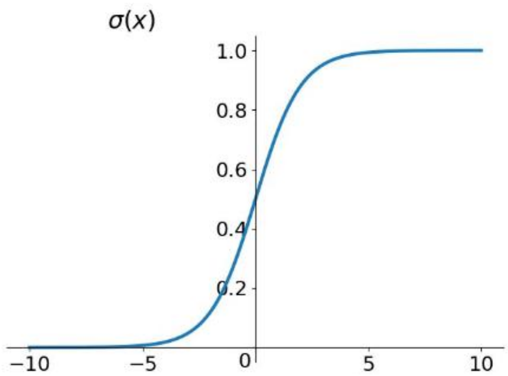
$$\frac{\partial C}{\partial z_i} = a_i - y_i$$

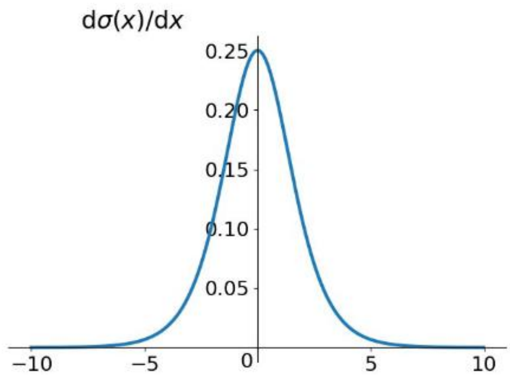
附录 2. 常用激活函数及其导数

1. Sigmoid 函数，表达式

$$y(x)=\frac{1}{1+e^{-x}}$$

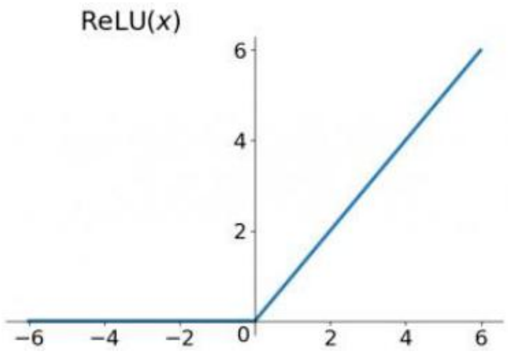
$$y'(x)=y(x)(1-y(x))$$

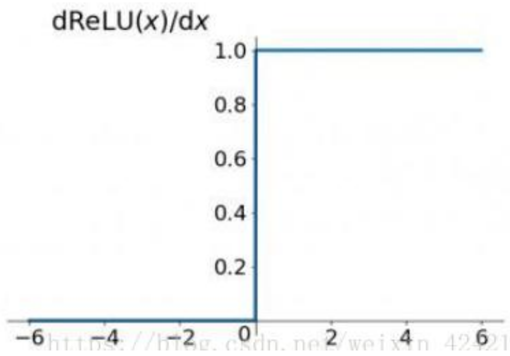




2. ReLu 函数，表达式

$$f(x)=\max(0,x)$$



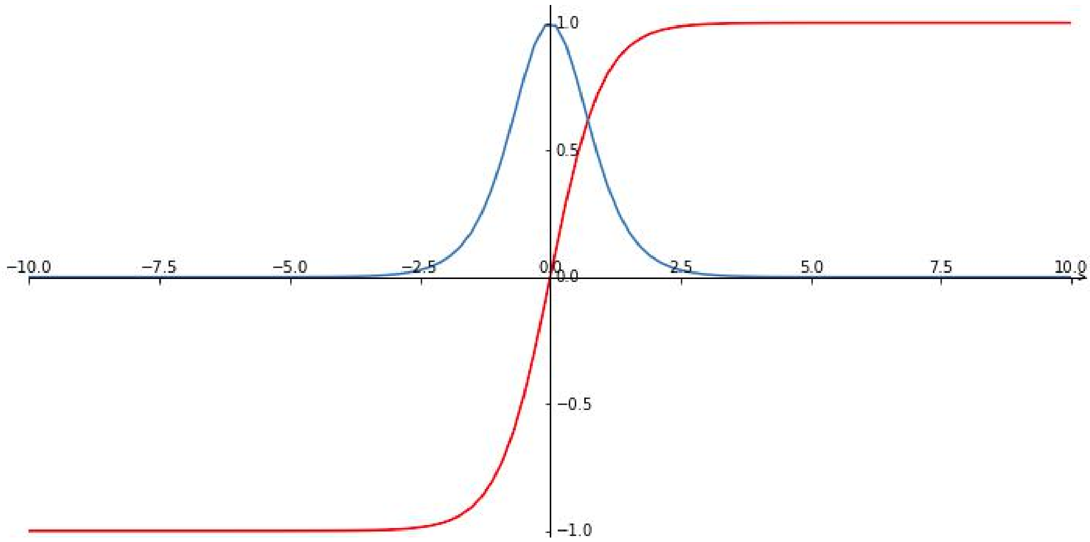


3. tanh 函数，表达式

$$tanhx=\frac{sinhx}{coshx}=\frac{e^x-e^{-x}}{e^x+e^{-x}}$$

导数为

$$\begin{aligned}
tanh'(x) &= ((e^x-e^{-x})(e^x+e^{-x}))' \\
&= (e^x+e^{-x})(e^x+e^{-x})^{-1}-(e^x-e^{-x})(e^x+e^{-x})^{-2}(e^x-e^{-x}) \\
&= 1-\frac{(e^x-e^{-x})^2}{(e^x+e^{-x})^2} \\
&= 1-tanh^2(x)
\end{aligned}$$



参考文献

[1] A guide to recurrent neural networks and backpropagation ,Mikael Boden, Dallas Project Sics Technical Report T Sics, 2001

[2] http://zh.gluon.ai/chapter_deep-learning-basics/backprop.html

[3] https://images2015.cnblogs.com/blog/583155/201608/583155-20160819153519750-1203124108.png

[4] https://zybuluo.com/hanbingtao/note/541458