# 常见激活函数比较

## 1 前言（为什么需要激活函数）

首先数据的分布绝大多数是非线性的，而一般神经网络的计算是线性的，引入激活函数，是在神经网络中引入非线性，强化网络的学习能力。所以激活函数的最大特点就是非线性。

首先，我们知道神经网络模拟了人类神经元的工作机理，激活函数（Activation Function）是一种添加到人工神经网络中的函数，旨在帮助网络学习数据中的复杂模式。在神经元中，输入的input经过一系列加权求和后作用于另一个函数，这个函数就是这里的激活函数。类似于人类大脑中基于神经元的模型，激活函数最终决定了是否传递信号以及要发射给下一个神经元的内容。在人工神经网络中，一个节点的激活函数定义了该节点在给定的输入或输入集合下的输出。标准的计算机芯片电路可以看作是根据输入得到开（1）或关（0）输出的数字电路激活函数。

激活函数可以分为线性激活函数（线性方程控制输入到输出的映射，如f(x)=x等）以及非线性激活函数（非线性方程控制输入到输出的映射，比如Sigmoid、Tanh、ReLU、LReLU、PReLU、Swish 等）

因为神经网络中每一层的输入输出都是一个线性求和的过程，下一层的输出只是承接了上一层输入函数的线性变换，所以如果没有激活函数，那么无论你构造的神经网络多么复杂，有多少层，最后的输出都是输入的线性组合，纯粹的线性组合并不能够解决更为复杂的问题。而引入激活函数之后，我们会发现常见的激活函数都是非线性的，因此也会给神经元引入非线性元素，使得神经网络可以逼近其他的任何非线性函数，这样可以使得神经网络应用到更多非线性模型中。

一般来说，在神经元中，激活函数是很重要的一部分，为了增强网络的表示能力和学习能力，神经网络的激活函数都是非线性的，通常具有以下几点性质：

连续并可导（允许少数点上不可导），可导的激活函数可以直接利用数值优化的方法来学习网络参数；

* 激活函数及其导数要尽可能简单一些，太复杂不利于提高网络计算率；
* 激活函数的导函数值域要在一个合适的区间内，不能太大也不能太小，否则会影响训练的效率和稳定性。

不同的激活函数，根据其特点，应用也不同。Sigmoid和tanh的特点是将输出限制在(0,1)和(-1,1)之间，说明Sigmoid和tanh适合做概率值的处理，例如LSTM中的各种门；而ReLU就不行，因为ReLU无最大值限制，可能会出现很大值。同样，根据ReLU的特征，Relu适合用于深层网络的训练，而Sigmoid和tanh则不行，因为它们会出现梯度消失。

在使用relu的网络中，是否还存在梯度消失的问题？梯度衰减因子包括激活函数导数，此外，还有多个权重连乘也会影响。梯度消失只是表面说法，按照这样理解，底层使用非常大的学习率，或者人工添加梯度噪音，原则上也能回避，有不少论文这样试了，然而目前来看，有用，但没太大的用处。深层原因训练不好的本质难题可能不是衰减或者消失（残差网络论文也提到这一点），目前数理派也搞不清楚究竟是什么，所以写了论文也顺势这样说开了。

## 2. Sigmoid函数

#### 2.1 公式及函数，导函数图像

Sigmoid函数也叫Logistic函数，用于隐层神经元输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。sigmoid是一个十分常见的激活函数，函数的表达式如下：

Sigmoid公式：

（2.1）

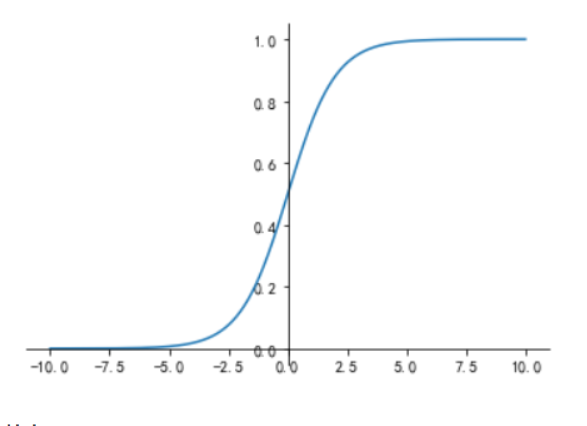


图2.1 函数图像

Sigmoid倒数：

（2.2）

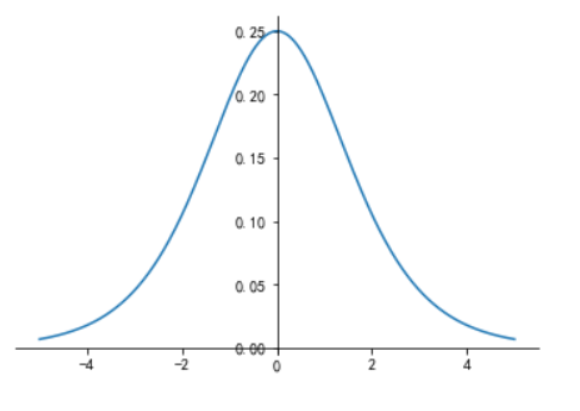


图2.2 导函数图像

#### 2.2 适用情况及缺点

适用情况：

* Sigmoid 函数的输出范围是0到1。由于输出值限定在0到1，因此它对每个神经元的输出进行了归一化；
* 用于将预测概率作为输出的模型。由于概率的取值范围是0到1，因此 Sigmoid 函数非常合适；
* 梯度平滑，避免「跳跃」的输出值；
* 函数是可微的。这意味着可以找到任意两个点的sigmoid曲线的斜率；
* 明确的预测，即非常接近1或0。

缺点：

* 梯度消失：注意：Sigmoid 函数趋近 0 和 1 的时候变化率会变得平坦，也就是说，Sigmoid 的梯度趋近于 0。神经网络使用 Sigmoid 激活函数进行反向传播时，输出接近 0 或 1 的神经元其梯度趋近于 0。这些神经元叫作饱和神经元。因此，这些神经元的权重不会更新。此外，与此类神经元相连的神经元的权重也更新得很慢。该问题叫作梯度消失。因此，想象一下，如果一个大型神经网络包含 Sigmoid 神经元，而其中很多个都处于饱和状态，那么该网络无法执行反向传播。
* 不以零为中心：Sigmoid 输出不以零为中心的,，输出恒大于0，非零中心化的输出会使得其后一层的神经元的输入发生偏置偏移（Bias Shift），并进一步使得梯度下降的收敛速度变慢。
* 计算成本高昂：exp()函数与其他非线性激活函数相比，计算成本高昂，计算机运行起来速度较慢。

## 3. Tanh/双曲正切激活函数

Tanh 激活函数又叫作双曲正切激活函数（hyperbolic tangent activation function）。与 Sigmoid 函数类似，Tanh 函数也使用真值，但 Tanh 函数将其压缩至-1 到 1 的区间内。与 Sigmoid 不同，Tanh 函数的输出以零为中心，因为区间在-1 到 1 之间。

函数表达式：

（3.1）

我们可以发现Tanh 函数可以看作放大并平移的Logistic 函数，其值域是(−1, Tanh与sigmoid的关系如下：

（3.2）

#### 3.1 函数图像

tanh 激活函数的图像也是 S 形，作为一个双曲正切函数，tanh 函数和 sigmoid 函数的曲线相对相似。但是它比 sigmoid 函数更有一些优势。

Tanh函数的图像如图3.1所示：

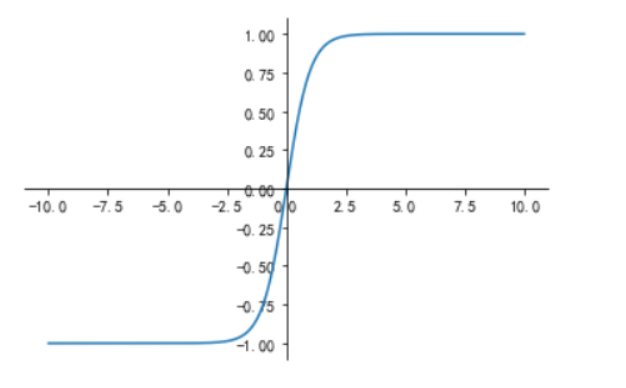


图3.1 tanh的函数图像

你可以将 Tanh 函数想象成两个 Sigmoid 函数放在一起。在实践中，Tanh 函数的使用优先性高于 Sigmoid 函数。负数输入被当作负值，零输入值的映射接近零，正数输入被当作正值：

* 当输入较大或较小时，输出几乎是平滑的并且梯度较小，这不利于权重更新。二者的区别在于输出间隔，tanh 的输出间隔为 1，并且整个函数以 0 为中心，比 sigmoid 函数更好；
* 在 tanh 图中，负输入将被强映射为负，而零输入被映射为接近零。

#### 3.2 导函数图像

其导函数是公式3.3：

（3.3）

导函数图像如图3.2：

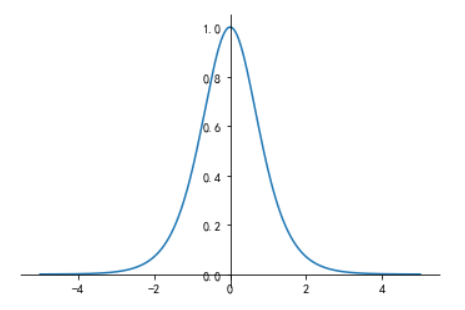


图3.2 tanh导函数图像

#### 3.3 优缺点

优点：

* 解决了sigmoid函数不是zero-centered的问题, 但是梯度消失依旧存在

缺点：

* 与sigmoid类似，Tanh 函数也会有梯度消失的问题，因此在饱和时（x很大或很小时）也会「杀死」梯度。

注意：在一般的二元分类问题中，tanh 函数用于隐藏层，而 sigmoid 函数用于输出层，但这并不是固定的，需要根据特定问题进行调整。

## 4. ReLU函数

ReLU函数又称为修正线性单元（Rectified Linear Unit），是一种分段线性函数，其弥补了sigmoid函数以及tanh函数的梯度消失问题，在目前的深度神经网络中被广泛使用。

#### 4.1 函数及导函数图像：

ReLU函数本质上是一个斜坡（ramp）函数，公式如4.1所示下：

（4.1）

图像如图4.1所示：

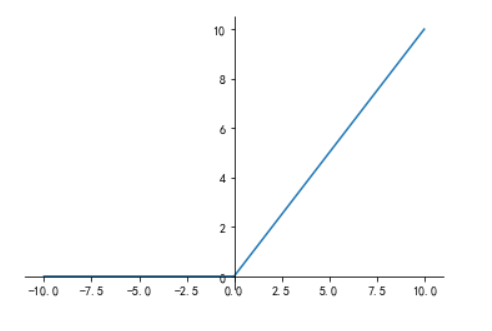


图4.1 ReLU函数图像

（4.2）

函数图像如图4.2：

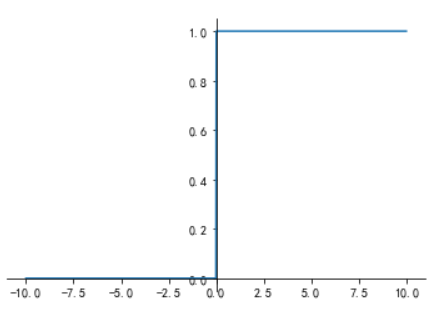


图4.2 ReLU导函数图像

#### 4.2 优缺点：

优点：

* 解决了梯度消失问题
* 计算速度非常快
* 收敛速度远快于sigmoid和tanh

缺点：

* Dead ReLU 问题。当输入为负时，ReLU 完全失效，在正向传播过程中，这不是问题。有些区域很敏感，有些则不敏感。但是在反向传播过程中，如果输入负数，则梯度将完全为零
* 【Dead ReLU问题】ReLU神经元在训练时比较容易“死亡”。在训练时，如果参数在一次不恰当的更新后，第一个隐藏层中的某个ReLU 神经元在所有的训练数据上都不能被激活，那么这个神经元自身参数的梯度永远都会是0，在以后的训练过程中永远不能被激活。这种现象称为死亡ReLU问题，并且也有可能会发生在其他隐藏层。
* 不以零为中心：和 Sigmoid 激活函数类似，ReLU 函数的输出不以零为中心，ReLU 函数的输出为 0 或正数,给后一层的神经网络引入偏置偏移，会影响梯度下降的效率。

## 5. Leaky ReLU函数

为了解决 ReLU 激活函数中的梯度消失问题，当 x < 0 时，我们使用 Leaky ReLU——该函数试图修复 dead ReLU 问题。下面我们就来详细了解 Leaky ReLU。

函数表达式如公式5.1，相当于是一个比较简单的Maxout单元：

（5.1）

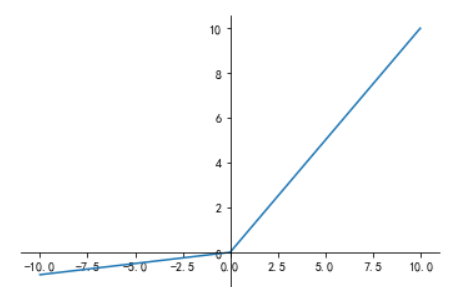


图5.1 Leaky ReLU函数图像（）

导函数图像如图5.2所示：

（5.2）

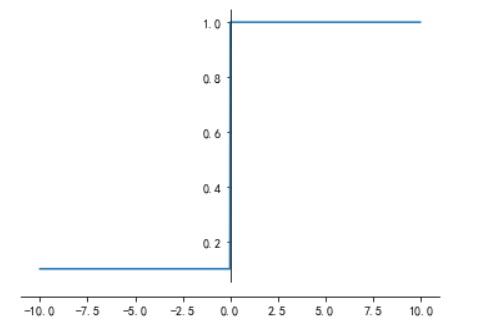


图5.2 Leaky ReLU导函数图像

为什么使用Leaky ReLU会比ReLU效果要好呢？

* Leaky ReLU 通过把 x 的非常小的线性分量给予负输入（0.01x）来调整负值的零梯度（zero gradients）问题，当 x < 0 时，它得到 0.1 的正梯度。该函数一定程度上缓解了 dead ReLU 问题，
* Leak 有助于扩大 ReLU 函数的范围，通常的值为 0.01 左右；
* Leaky ReLU 的函数范围是（负无穷到正无穷），不会有Dead ReLU问题

尽管Leaky ReLU具备 ReLU 激活函数的所有特征（如计算高效、快速收敛、在正区域内不会饱和），但并不能完全证明在实际操作中Leaky ReLU 总是比 ReLU 更好。