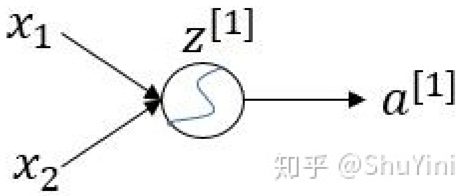
深度学习中激活函数的作用

1.引言

在深度学习网络中，对于某个隐藏层节点的激活值一般分为两步，如下图：



第一步，输入节点x1，x2，先做一个线性变换：

$$z^{[1]} = w_1x_1 + w_2x_2 + b^{[1]} = W^{[1]}x + b^{[1]}$$

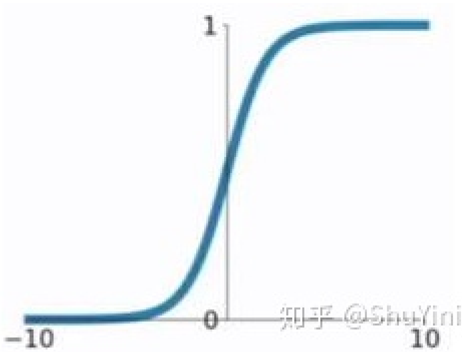
第二步，进行一个非线性变换，也就是经过非线性激活函数，计算出该节点的激活值。

$$a^{(1)} = g(z^{(1)})$$

g（z）为非线性函数。

1.1 什么是激活函数

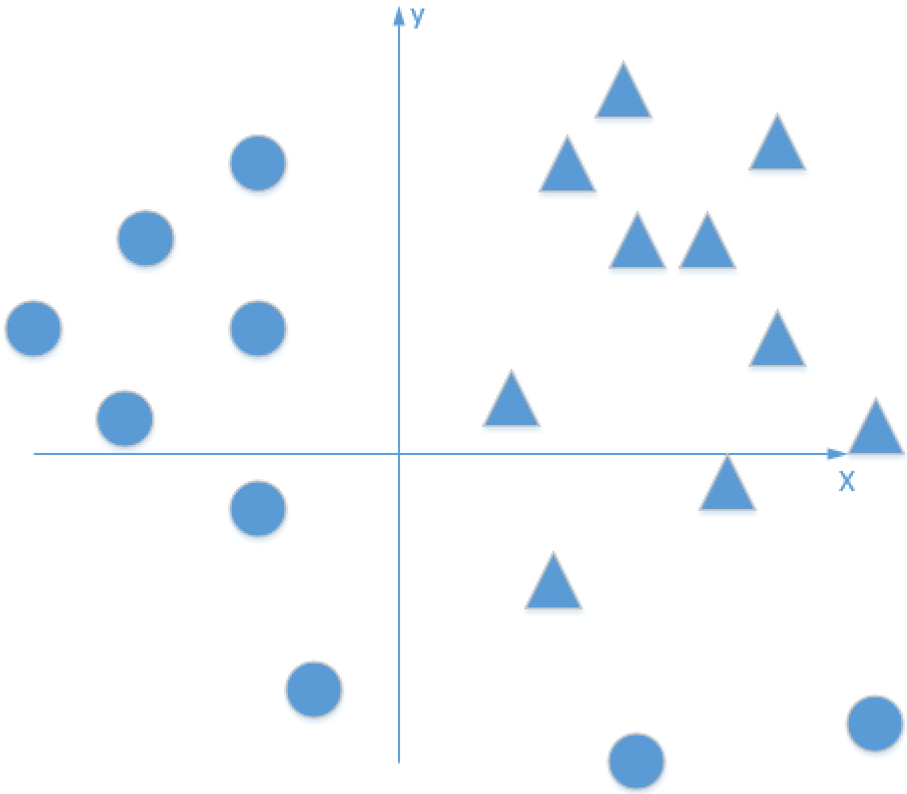
激活函数是神经网络中极其重要的概念。它们决定了某个神经元是否被激活，这个神经元接受到的信息是否是有用的，是否该留下或者是该抛弃。激活函数的形式如下：



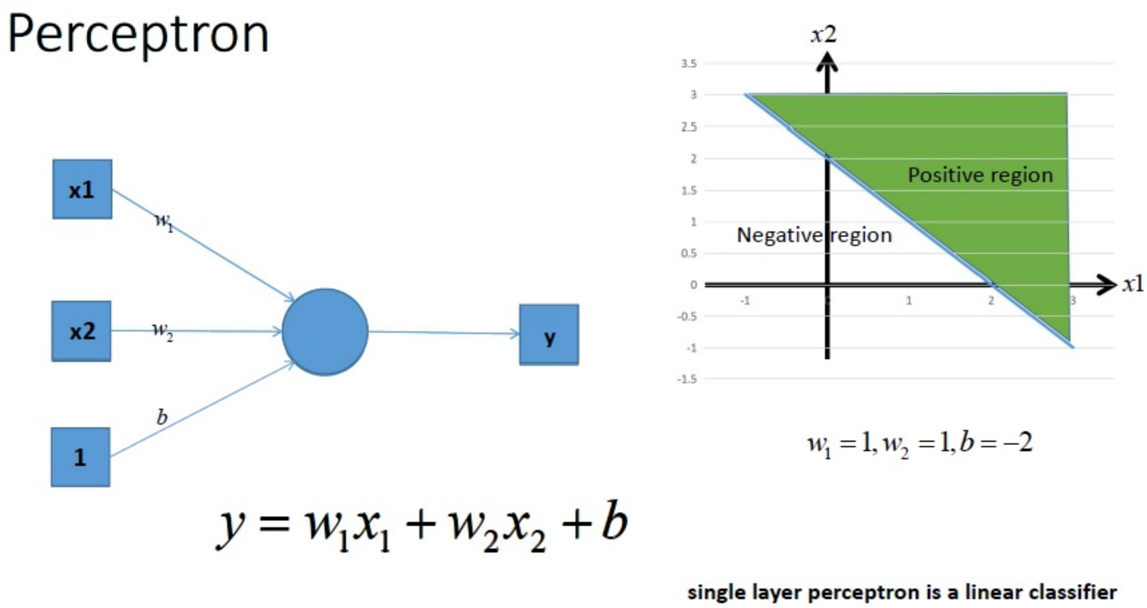
激活函数是我们对输入做的一种非线性的转换。转换的结果输出，并当作下一个隐藏层的输入。

2 直观理解激活函数的作用

首先我们有这个需求，就是二分类问题，如我要将下面的三角形和圆形点进行正确的分类，如下图：

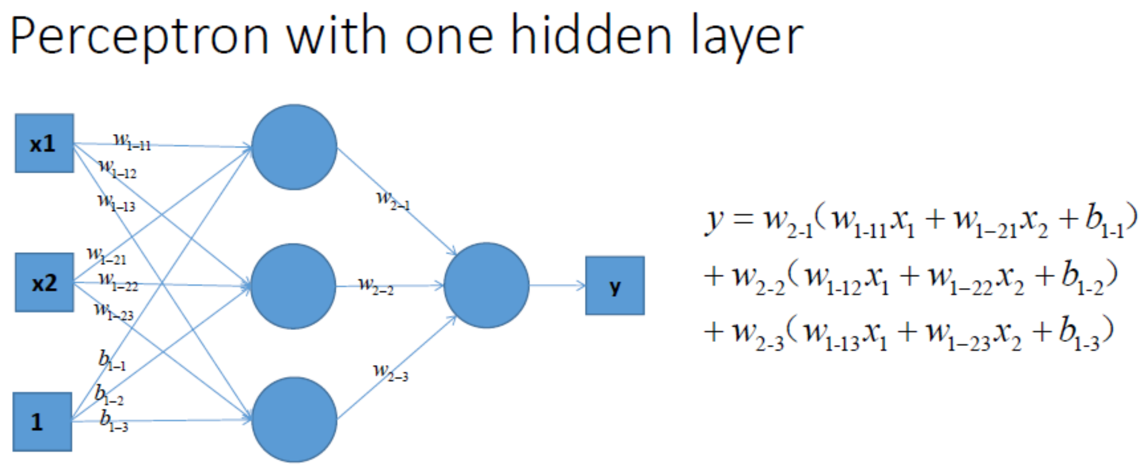


利用我们单层的感知机, 用它可以划出一条线, 把平面分割开：



上图直线是由$w_1x_1 + w_2x_2 + b = 0$得到，那么该感知器实现预测的功能步骤如下，就是我已经训练好了一个感知器模型，后面对于要预测的样本点，带入模型中，如果$y > 0$,那么就说明是直线的右侧，也就是正类（我们这里是三角形），如果$y < 0$,那么就说明是直线的左侧，也就是负类（我们这里是圆形），虽然这和我们的题目关系不大，但是还是提一下~

好吧，很容易能够看出，我给出的样本点根本不是线性可分的，一个感知器无论得到的直线怎么动，都不可能完全正确的将三角形与圆形区分出来，那么我们很容易想到用多个感知器来进行组合，以便获得更大的分类问题，好的，下面我们上图，看是否可行：



好的，我们已经得到了多感知器分类器了，那么它的分类能力是否强大到能将非线性数据点正确分类开呢~我们来分析一下：

我们能够得到

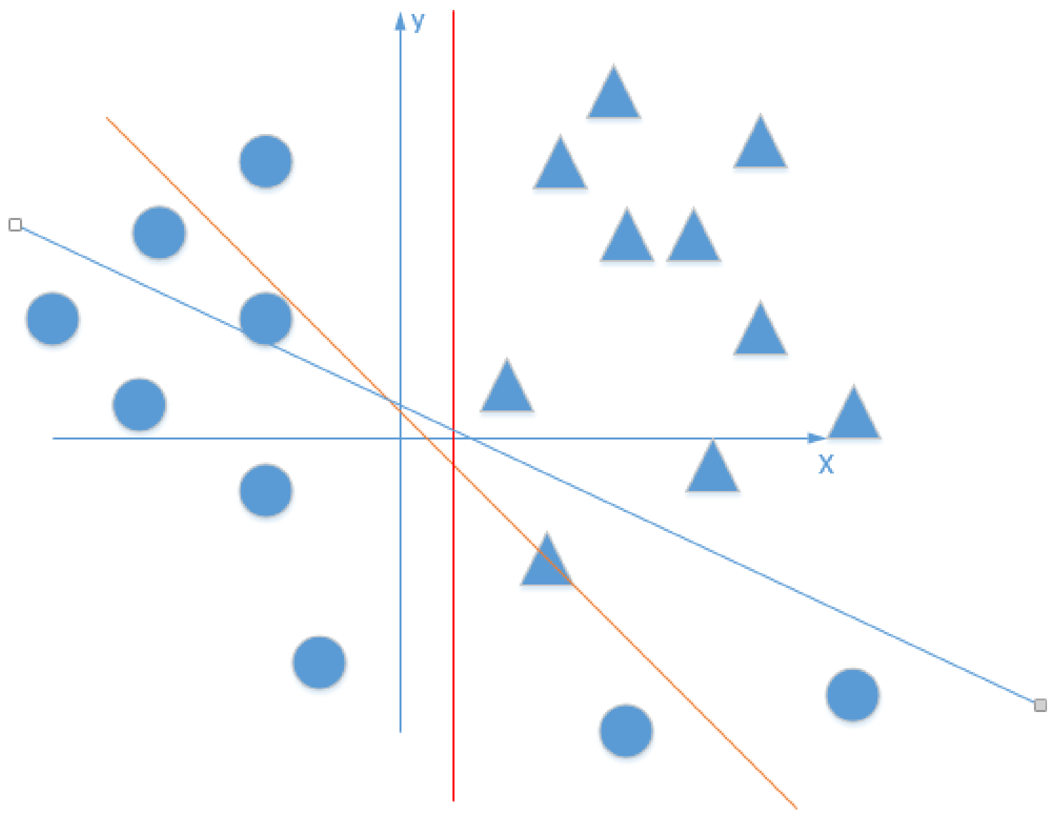
$$y = w_{2-1}(w_{1-11}x_1 + w_{1-21}x_2 + b_{1-1}) + w_{2-2}(w_{1-12}x_1 + w_{1-22}x_2 + b_{1-2}) + w_{2-3}(w_{1-13}x_1 + w_{1-23}x_2 + b_{1-3})$$

哎呀呀，不得了，这个式子看起来非常复杂，估计应该可以处理我上面的情况了吧，哈哈哈哈~不一定额，我们来给它变个形.上面公式合并同类项后等价于下面公式：

$$y = x_1(w_{2-1}w_{1-11} + w_{2-2}w_{1-12} + w_{2-3}w_{1-13}) + x_2(w_{2-1}w_{1-21} + w_{2-2}w_{1-22} + w_{2-3}w_{1-23}) + w_{2-1}b_{1-1} + w_{2-2}b_{1-2} + w_{2-3}b_{1-3}$$

啧啧，估计大家都看出了，不管它怎么组合，最多就是线性方程的组合，最后得到的分类器本质还是一个线性方程，该处理不了的非线性问题，它还是处理不了。

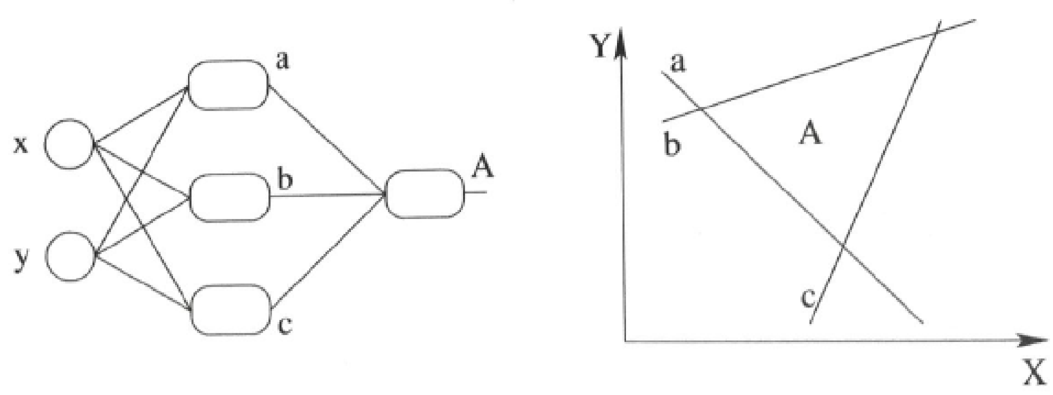
就好像下图，直线无论在平面上如果旋转，都不可能完全正确的分开三角形和圆形点：



既然是非线性问题，总有线性方程不能正确分类的地方~

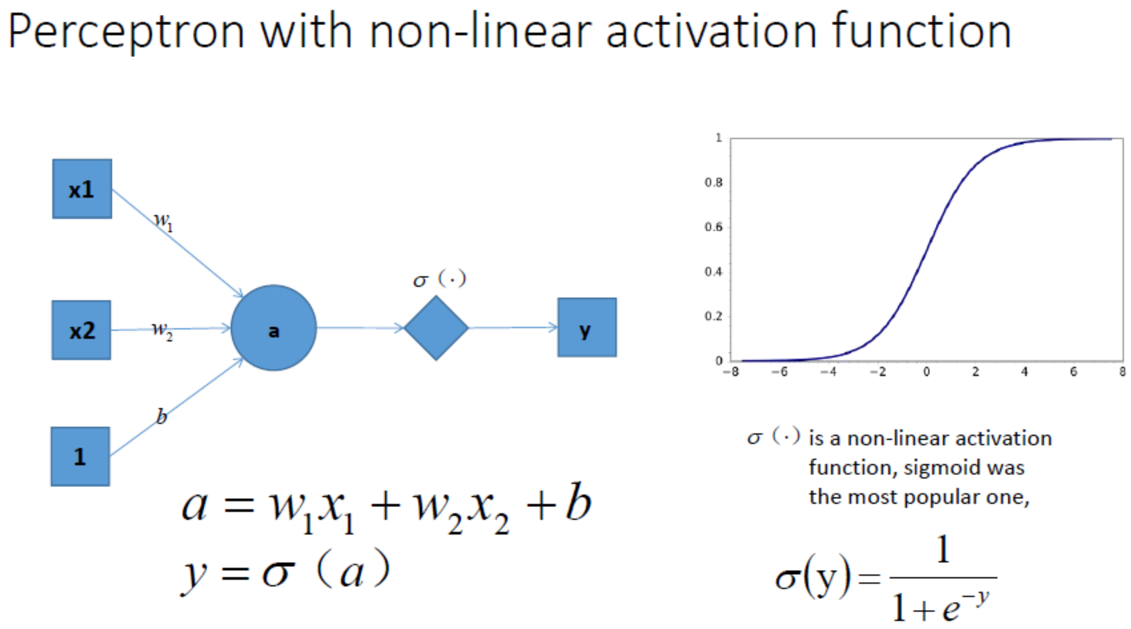
那么抛开神经网络中神经元需不需要激活函数这点不说，如果没有激活函数，仅仅是线性函数的组合解决的问题太有限了，碰到非线性问题就束手无策了.那么加入激活函数是否可能能够解决呢?

在上面线性方程的组合过程中（在加入阶跃激活函数的时候），我们其实类似在做三条直线的组合，如下图：



with step activation function

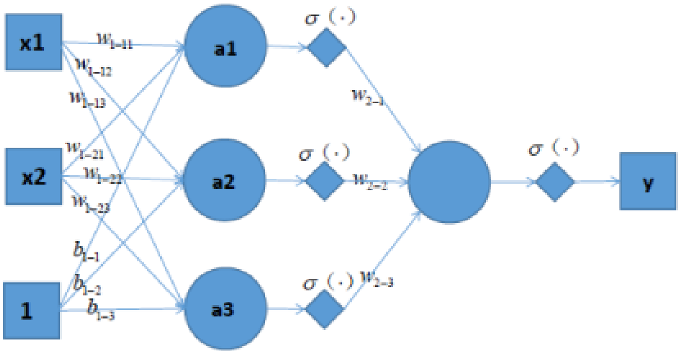
下面我们来讲一下激活函数，我们都知道，每一层叠加完了之后，我们需要加入一个激活函数（激活函数的种类也很多，如sigmoid等等~）这里就给出sigmoid例子，如下图：



通过这个激活函数映射之后，输出很明显就是一个非线性函数！能不能解决一开始的非线性分类问题不清楚，但是至少说明有可能啊，上面不加入激活函数神经网络压根就不可能解决这个问题~

同理，扩展到多个神经元组合的情况时候，表达能力就会更强~对应的组合图如下：（现在已经升级为三个非线性感知器在组合了）

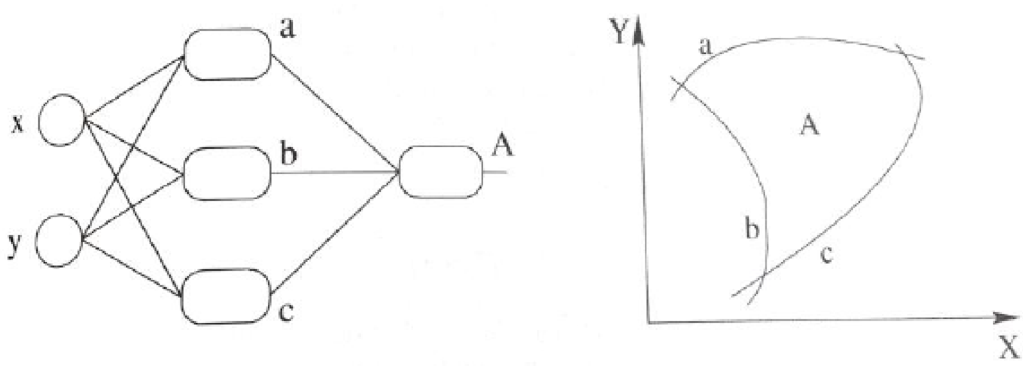
Perceptron with non-linear activation function



$$a1 = w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}$$
$$a2 = w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}$$
$$a3 = w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3}$$

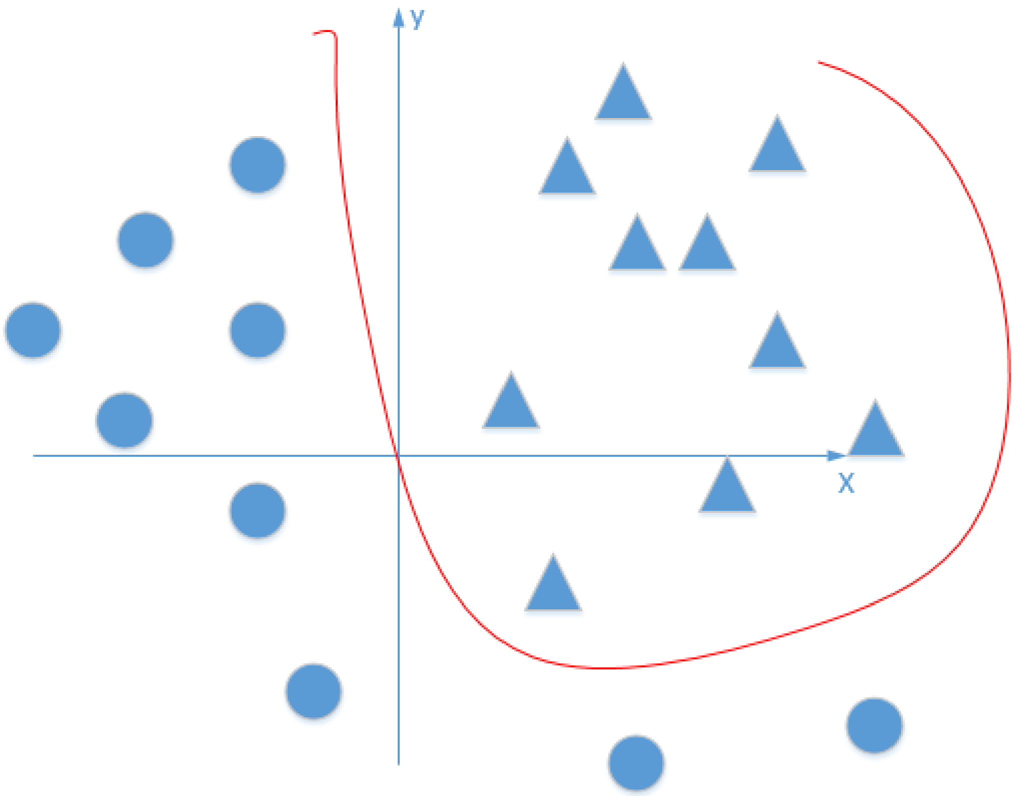
$$y = \sigma(w_{2-1}\sigma\ (a1) + w_{2-2}\sigma\ (a2) + w_{2-3}\sigma\ (a3))$$

跟上面线性组合相对应的非线性组合如下：



with sigmoid activation function

这看起来厉害多了，是不是~最后再通过最优化损失函数的做法，我们能够学习到不断学习靠近能够正确分类三角形和圆形点的曲线，到底会学到什么曲线，不知道到底具体的样子，也许是下面这个~



那么随着不断训练优化，我们也就能够解决非线性的问题了~

所以到这里为止，我们就解释了这个观点，加入激活函数是用来加入非线性因素的，解决线性模型所不能解决的问题。

2.1 疑惑

线性回归并不适用于所有数据，有时我们需要曲线来适应我们的数据，比如一个二次方模型：$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$或者三次方模型：$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$。通常我们需要先观察数据然后再决定准备尝试怎样的模型。另外，我们可以令：$x_2 = x_2^2$，$x_3 = x_3^3$，从而将模型转化为线性回归模型。

3. 激活函数的作用

激活函数的作用:激活函数是用来加入非线性因素的，因为线性模型的表达能力不够。

- 线性变换太简单（只是加权偏移），限制了对复杂任务的处理能力。**没有激活函数的神经网络就是一个线性回归模型**。激活函数做的非线性变换可以使得神经网络处理非常复杂的任务。例如，我们希望我们的神经网络可以对语言翻译和图像分类做操作，这就需要非线性转换。同时，激活函数也使得反向传播算法变的可能。因为，这时候梯度和误差会被同时用来更新权重和偏移。没有可微分的线性函数，这就不可能了。
- 首先对于y=ax+b 这样的函数，当x的输入很大时，y的输出也是无限大/小的，经过多层网络叠加后，值更加膨胀的没边了，这显然不符合我们的预期，很多情况下我们希望的输出是一个概率。
- 即使经过多层网络的叠加，y=ax+b无论叠加多少层最后仍然是线性的。非线性，这样增加网络的深度才有意义

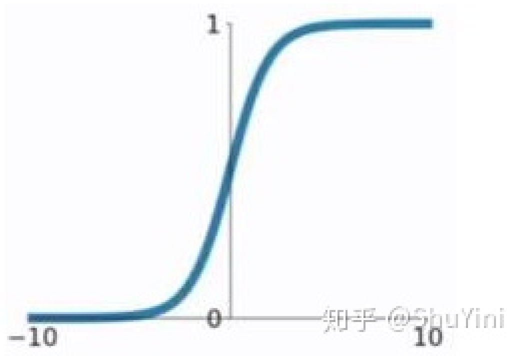
4.常见的激活函数

在深度学习中，常用的激活函数主要有：**sigmoid函数，tanh函数，ReLU函数。**

sigmoid函数

该函数是将取值为 $(-\infty,+\infty)$(的数映射到 $(0,1)$ 之间。sigmoid函数的公式以及图形如下：

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



对于sigmoid函数的求导推导为：

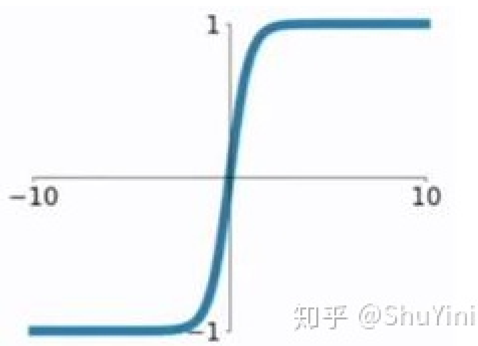
$$\begin{aligned} g'(z) &= \left(\frac{1}{1+e^{-z}}\right)' \\ &= \frac{e^{-z}}{(1+e^{-z})^2} \\ &= g(z)(1-g(z)) \end{aligned}$$

sigmoid函数作为非线性激活函数，但是其并不被经常使用，它具有以下几个缺点：　　（1）当 z 值非常大或者非常小时，通过上图我们可以看到，sigmoid函数的导数 $g'(z)$ 将接近0 。这会导致权重W的梯度将接近0，使得梯度更新十分缓慢，即梯度消失。

（2）函数的输出不是以0为均值，将不便于下层的计算。sigmoid函数可用在网络最后一层，作为输出层进行二分类，尽量不要使用在隐藏层。

tanh函数

tanh函数相较于sigmoid函数要常见一些，该函数是将取值为 $(-\infty,+\infty)$的数映射到 $(-1,1)$ 之间，其公式与图形为 $g(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



tanh函数在0附近很短一段区域内可看做线性的。由于tanh函数均值为0，因此弥补了sigmoid函数均值为0.5的缺点。对于tanh函数的求导推导为：

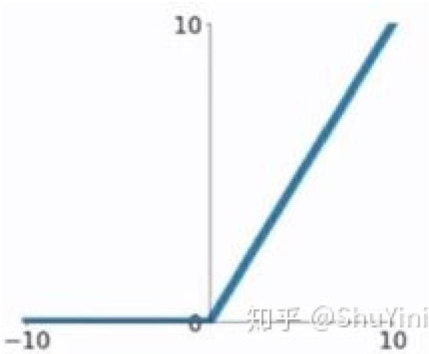
$$g'(z) = \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)' = \frac{4}{(e^z + e^{-z})^2} = 1 - g(z)^2$$

tanh函数的缺点同sigmoid函数的第一个缺点一样，当 z很大或很小时，g'(z)接近于0，会导致梯度很小，权重更新非常缓慢，即梯度消失问题。

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。ReLU函数的公式以及图形如下：

$$g(z) = \begin{cases} z, & \text{if } z > 0 \\ 0, & \text{if } z < 0 \end{cases}$$



对于ReLU函数的求导为： $g'(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z < 0 \end{cases}$

ReLU函数的优点：

（1）在输入为正数的时候（对于大多数输入空间来说），不存在梯度消失问题。

（2） 计算速度要快很多。ReLU函数只有线性关系，不管是前向传播还是反向传播，都比sigmod和tanh要快很多。（sigmod和tanh要计算指数，计算速度会比较慢）

ReLU函数的缺点：当输入为负时，梯度为0，会产生梯度消失问题。