Microsoft

Reactor

一起学人工智能系列
- 激活函数

2021-09-29

Map


Stockholm
COMING SOON
London
Redmond
Toronto
New York
San Francisco
Tel Aviv
Shanghai
Abu Dhabi
Bangalore
Sao Paulo
Sydney

个人介绍

Kinfey Lo – (卢建晖)

Microsoft Cloud Advocate

前微软MVP、Xamarin MVP和微软RD，拥有超过10年的云原生、人工智能和移动应用经验，为教育、金融和医疗提供应用解决方案。 Microsoft Iginte，Teched 会议讲师，Microsoft AI 黑客马拉松教练，目前在微软，为技术人员和不同行业宣讲技术和相关应用场景。

爱编程(Python， C#，TypeScript，Swift，Rust，Go ）

专注于人工智能，云原生，跨平台移动开发

Github : https://github.com/kinfey

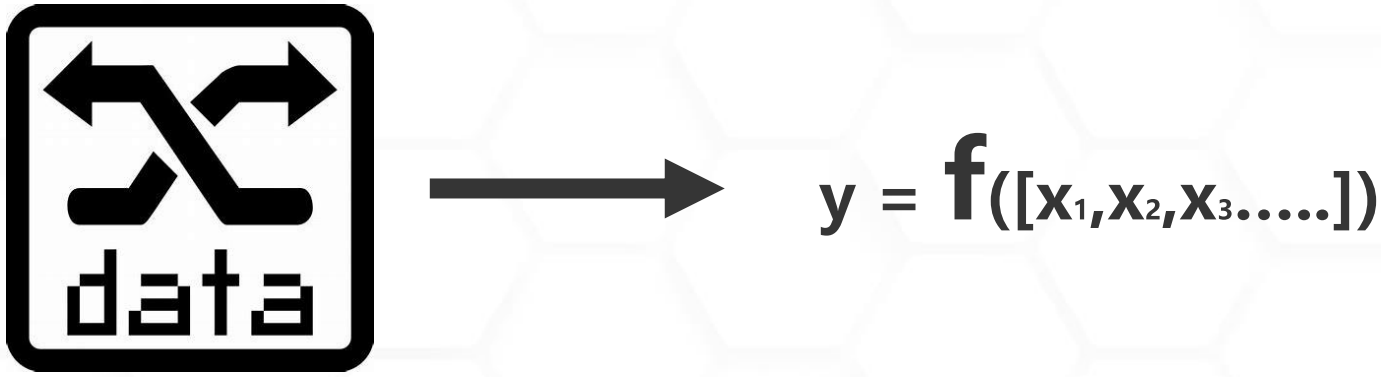
Email : kinfeylo@microsoft.com **Blog :** https://blog.csdn.net/kinfey

Twitter : @Ljh8304

回顧

一切从监督学习谈起

Supervised Learning – 预测未来



经验决定一切，整合大量的标注数据，应用在预测价格，判断分类等场景上

需要找到一个基于特征数据，生成结果的方法

一般都有多个特征数据

x值拟合到计算中，从而为训练数据集中的所有情况合理准确地生成y。

单变量线性回归问题

回归分析是一种数学模型。当因变量和自变量为线性关系时，它是一种特殊的线性模型。

最简单的情形是一元线性回归，由大体上有线性关系的一个自变量和一个因变量组成，模型是：

$$y = \mathrm{a} + \mathrm{b}X + \varepsilon$$

X是自变量，Y 是因变量，ε 是随机误差，a 和 b 是参数，在线性回归模型中，a,b 是要通过算法学习出来的

线性回归模型

对于线性回归模型，有如下一些概念需要了解：

•通常假定随机误差 ε 的均值为 0，方差为σ2（σ2>0，σ2 与 X 的值无关）

•若进一步假定随机误差遵从正态分布，就叫做正态线性模型

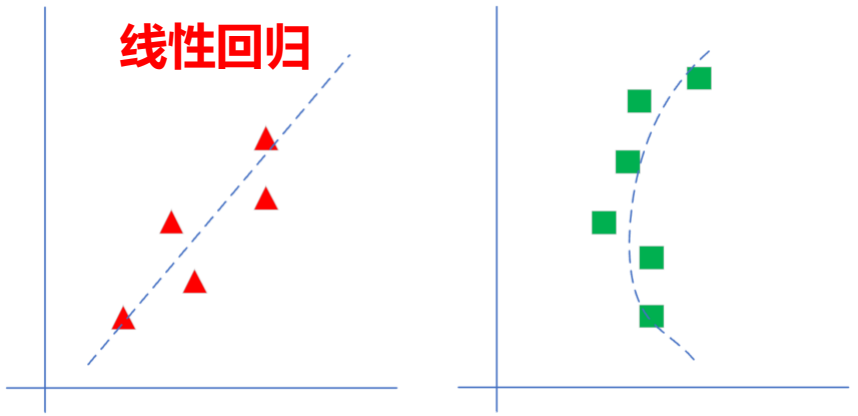
•一般地，若有 k 个自变量和 1 个因变量（即公式1中的 Y），则因变量的值分为两部分：一部分由自变量影响，即表示为它的函数，函数形式已知且含有未知参数；另一部分由其他的未考虑因素和随机性影响，即随机误差

•当函数为参数未知的线性函数时，称为线性回归分析模型

•当函数为参数未知的非线性函数时，称为非线性回归分析模型

•当自变量个数大于 1 时称为多元回归

•当因变量个数大于 1 时称为多重回归



多元线性回归

线性回归问题，而且是典型的多元线性回归，即包括两个或两个以上自变量的回归。多元线性回归的函数模型如下：

$$y = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_k x_k$$

具体化到房价预测问题，上面的公式可以简化成：

$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + b$$

多元线性回归

1.自变量对因变量必须有显著的影响，并呈密切的线性相关；

2.自变量与因变量之间的线性相关必须是真实的，而不是形式上的；

3.自变量之间应具有一定的互斥性，即自变量之间的相关程度不应高于自变量与因变量之因的相关程度；

4.自变量应具有完整的统计数据，其预测值容易确定。

数学方式解决问题

多元线性方程式

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots + a_kx_k$$

函数拟合（回归）时，我们假设函数 为

$$H(w,b) = b + x_1w_1 + x_2w_2 + \cdots + x_nw_n$$

b = w0

$$H(W) = w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n$$

x 是一个样本的 n 个特征值，如果我们把 m 个样本一起计算，将会得到下面这个矩阵

$$H(W) = X \cdot W$$

二分类数学原理-代数方式

代数方式：通过一个分类函数计算所有样本点在经过线性变换后的概率值，使得正例样本的概率大于0.5，而负例样本的概率小于0.5。

1. 正向计算

$$z = x_1 w_1 + x_2 w_2 + b \tag{1}$$

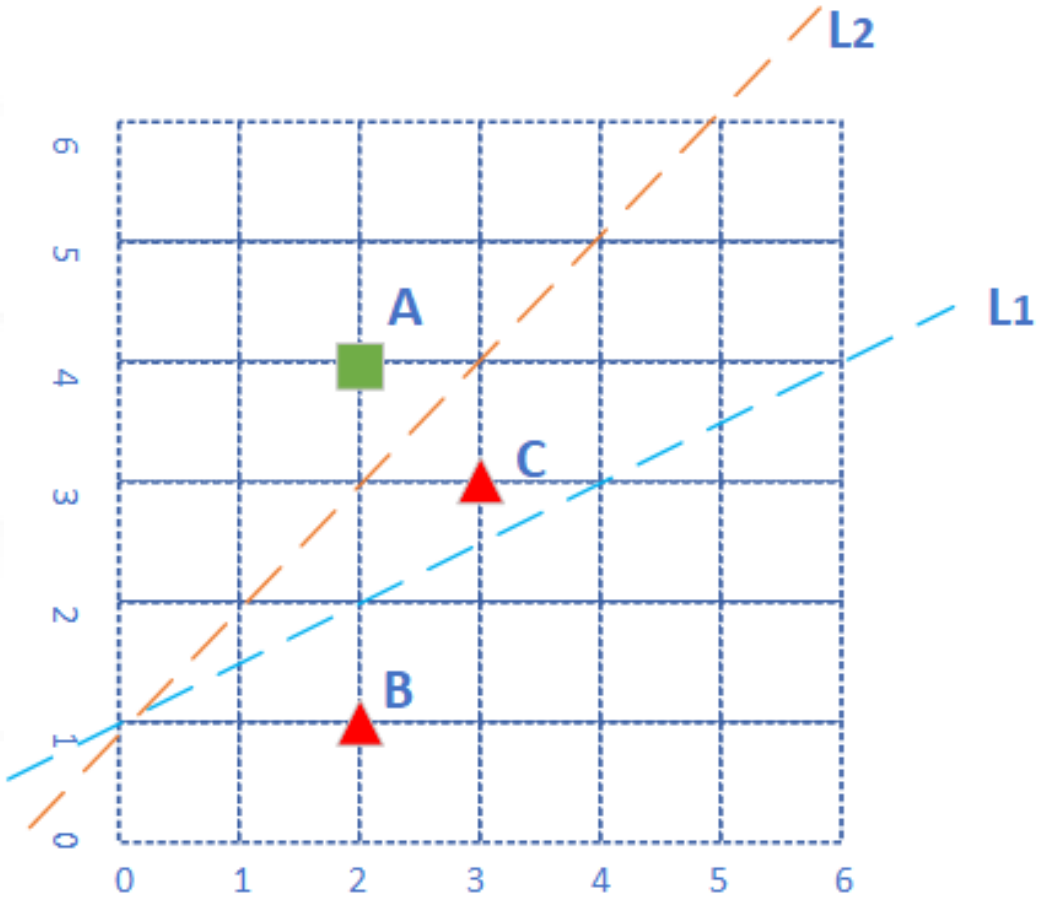
1. 分类计算

$$a = \frac{1}{1+e^{-z}} \tag{2}$$

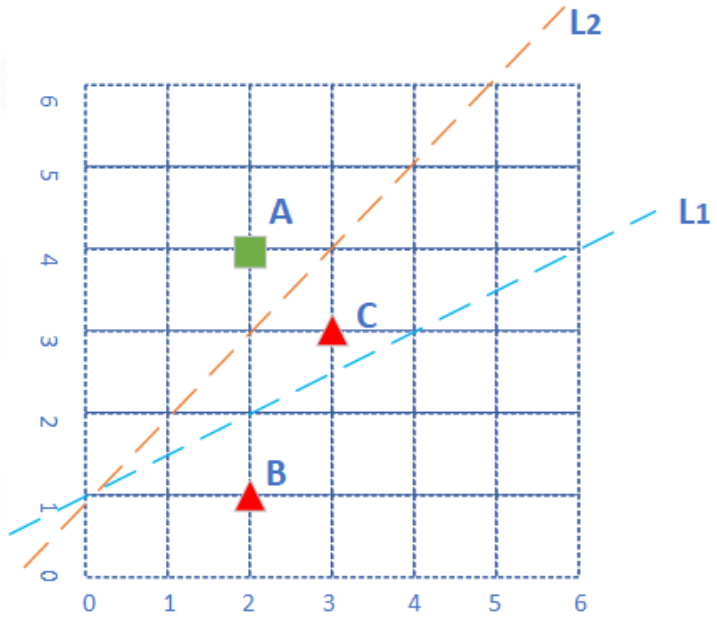
1. 损失函数计算

$$loss = -[y \ln(a) + (1-y) \ln(1-a)] \tag{3}$$

二分类数学原理-代数分析



二分类数学原理-代数分析



分类线为L1时

假设神经网络第一次使用 L_1 做为分类线，此时： $w_1 = -1, w_2 = 2, b = -2$ ，我们来计算一下三个点的情况。

A 点:

$$z_A = (-1) \times 2 + 2 \times 4 - 2 = 4 > 0 \quad \text{(正确)}$$

B 点:

$$z_B = (-1) \times 2 + 2 \times 1 - 2 = -2 < 0 \quad \text{(正确)}$$

C 点:

$$z_C = (-1) \times 3 + 2 \times 3 - 2 = 1 > 0 \quad \text{(错误)}$$

我们知道当 $z > 0$ 时， $Logistic(z) > 0.5$ 为正例，反之为负例，所以我们只需要看三个点的 z 值是否大于0或小于0就可以了，不用再计算 $Logistic$ 的函数值。

其中， A、B 点是处于正确的分类区，而 C 点处于错误的分类区。此时 C 点的损失函数值为（注意 C 的标签值 $y = 0$ ）：

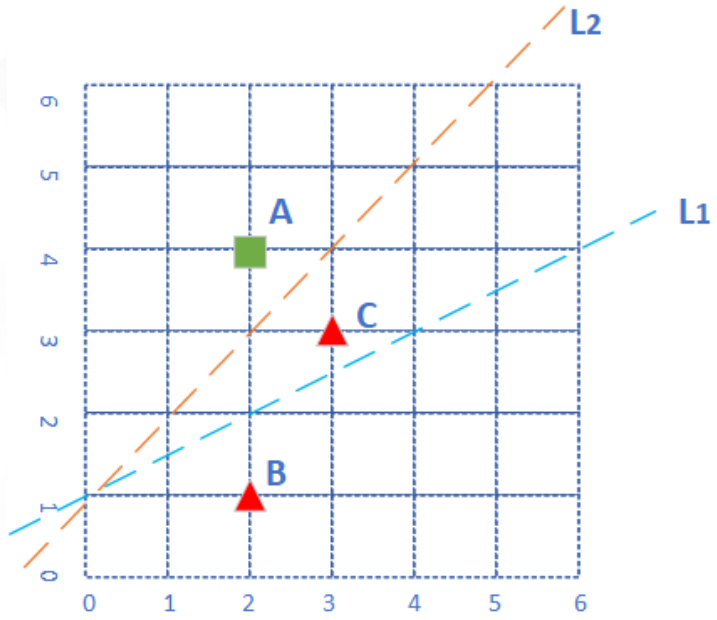
$$a_C = Sigmoid(z_C) = 0.731$$

$$loss_Z = -(0 \cdot \ln(0.731) + 1 \cdot \ln(1 - 0.731)) = 1.313$$

读者可能对1.313这个值没有什么概念，是大还是小呢？我们不妨计算一下分类正确的 A、B 点的坐标：

$$loss_A = 0.018, \quad loss_B = 0.112$$

二分类数学原理-代数分析



分类线为L1时

假设神经网络第一次使用 L_1 做为分类线，此时： $w_1 = -1, w_2 = 2, b = -2$ ，我们来计算一下三个点的情况。

A 点:

$$z_A = (-1) \times 2 + 2 \times 4 - 2 = 4 > 0 \quad \text{(正确)}$$

B 点:

$$z_B = (-1) \times 2 + 2 \times 1 - 2 = -2 < 0 \quad \text{(正确)}$$

C 点:

$$z_C = (-1) \times 3 + 2 \times 3 - 2 = 1 > 0 \quad \text{(错误)}$$

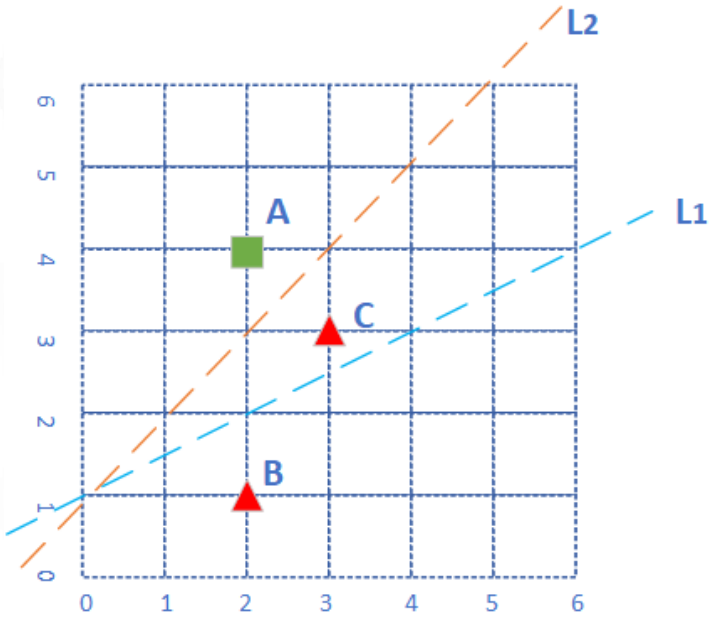
我们知道当 $z > 0$ 时， $Logistic(z) > 0.5$ 为正例，反之为负例，所以我们只需要看三个点的 z 值是否大于0或小于0就可以了，不用再计算 $Logistic$ 的函数值。

其中， A、B 点是处于正确的分类区，而 C 点处于错误的分类区。此时 C 点的损失函数值为（注意 C 的标签值 $y = 0$ ）：

$$a_C = Sigmoid(z_C) = 0.731$$

$$loss_Z = -(0 \cdot \ln(0.731) + 1 \cdot \ln(1 - 0.731)) = 1.313$$

二分类数学原理-代数分析



分类线为L2时

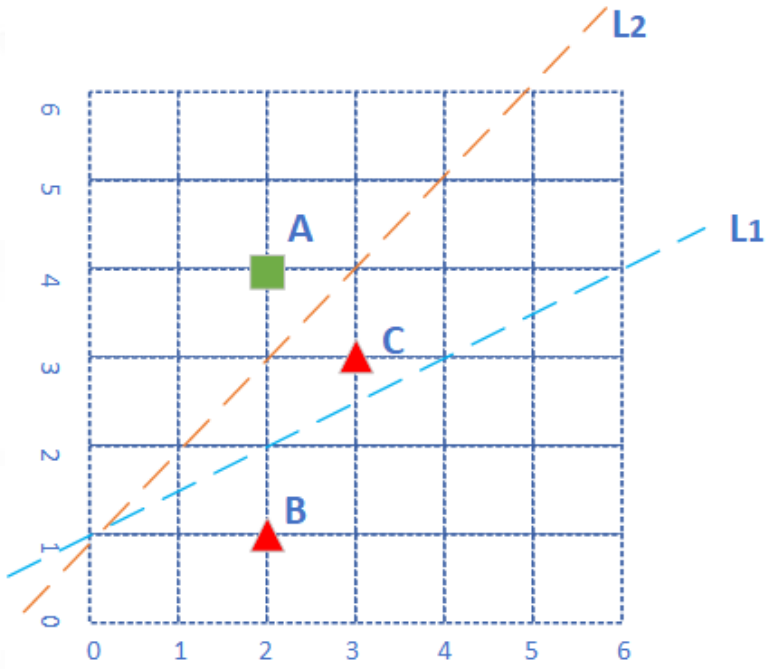
我们假设经过反向传播后，神经网络把直线的位置调整到 L_2，以 L_2 做为分类线，即 $w_1 = -1, w_2 = 1, b = -1$，则三个点的 z 值都会是符合其分类的：

$$z_A = (-1) \times 2 + 1 \times 4 - 1 = 1 > 0 \quad \text{(正确)}$$

$$z_B = (-1) \times 2 + 1 \times 1 - 1 = -2 < 0 \quad \text{(正确)}$$

$$z_C = (-1) \times 3 + 1 \times 3 - 1 = -1 < 0 \quad \text{(正确)}$$

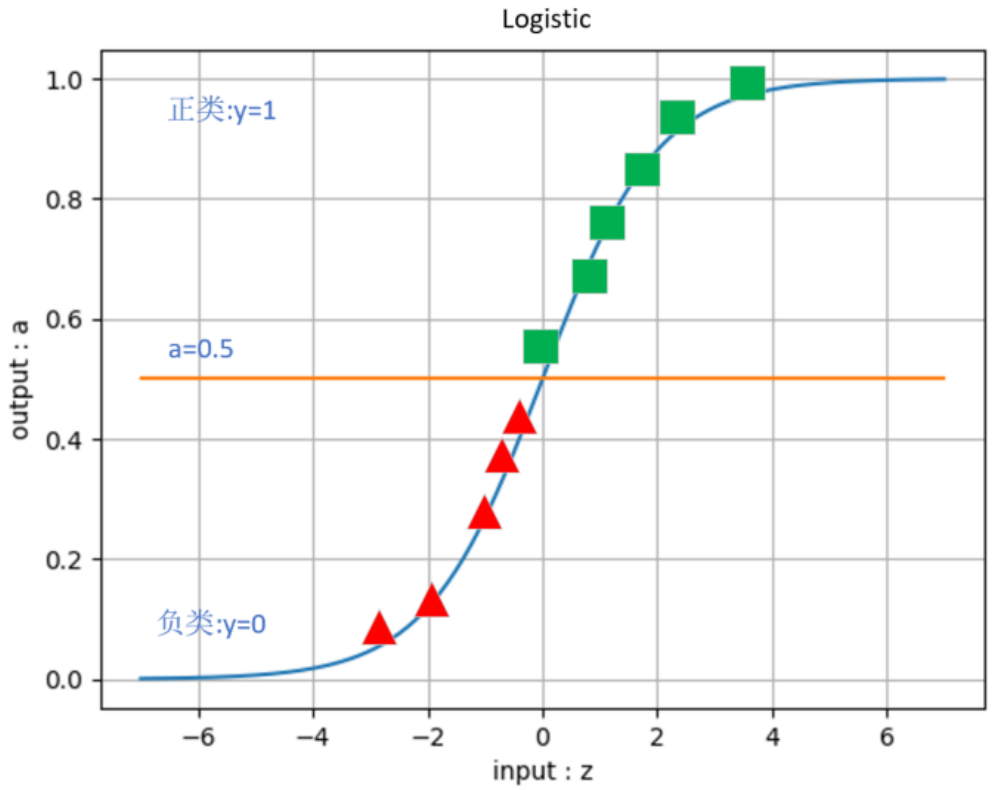
既然用z值是否大于0这个条件就可以判断出分类是否正确，那么二分类理论中为什么还要用Logistic函数做一次分类呢



只有z值的话，我们只能知道是大于0还是小于0，并不能有效地进行反向传播，也就是说我们无法告诉神经网络反向传播的误差的力度有多大。比如z=5和z=−1相比，难度意味着前者的力度是后者的5倍吗?

而有了Logistic分类计算后，得到的值是一个(0,1)之间的概率，比如：当z=5时，Logistic(5)=0.993；当z=−1时，Logistic(−1)=0.269。这两个数值的含义是这两个样本在分类区内的概率，前者概率为99.3%，偏向正例，后者概率为26.9%，偏向负例。然后再计算损失函数，就可以得到神经网络可以理解的反向传播误差，比如上面曾经计算过的lossA、lossB、lossC。

二分类数学原理-几何



$$a = Logistic(z) = \frac{1}{1+e^{-z}} > 0.5$$

做公式变形，两边取自然对数，可以得到：

$$z > 0$$

即： $z = x_1 \cdot w_1 + x_2 \cdot w_2 + b > 0$

对上式做一下变形，把 x_2 放在左侧，其他项放在右侧（假设 $w_2 > 0$，则不等号方向不变）：

$$x_2 > -\frac{w_1}{w_2}x_1 - \frac{b}{w_2} \tag{5}$$

简化一下两个系数，令 $w' = -w1/w2$，$b' = -b/w2$：

$$x_2 > w' \cdot x_1 + b' \tag{6}$$

公式6用几何方式解释，就是：有一条直线，方程为 $z = w' \cdot x_1 + b'$，所有的正例样本都处于这条直线的上方；同理可得所有的负例样本都处于这条直线的下方。

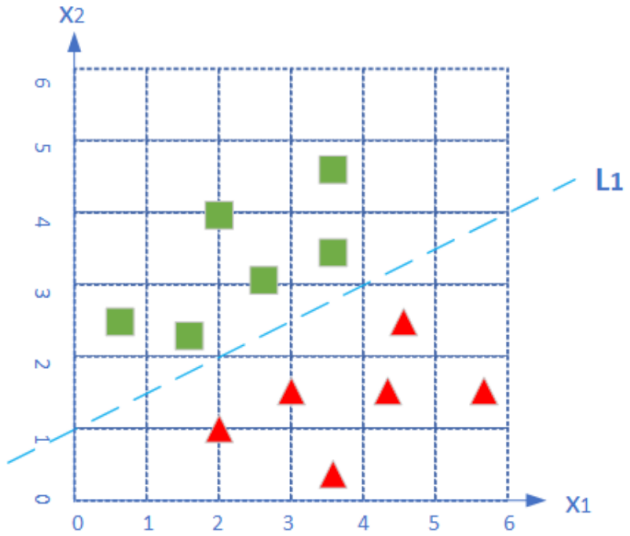
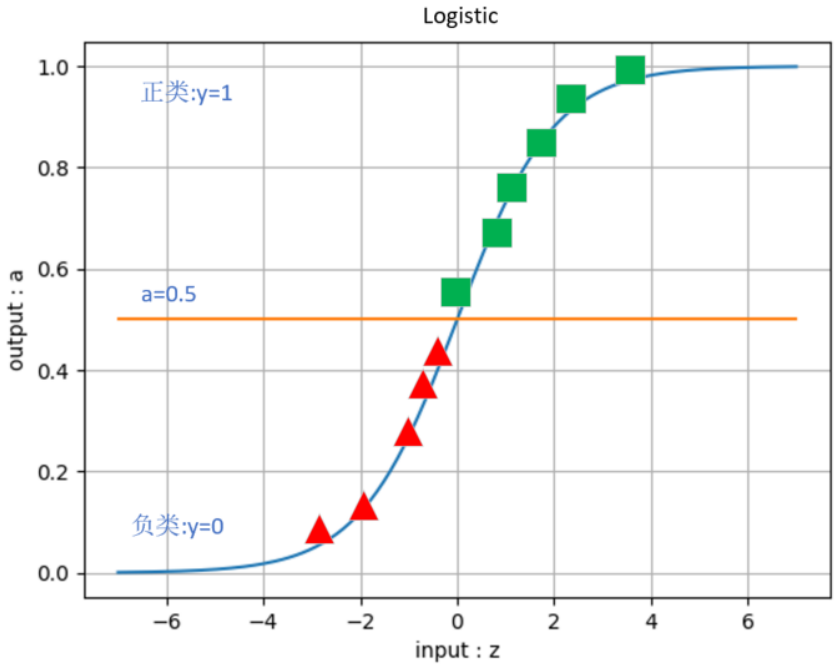


图6-7 用直线分开的两类样本

二分类数学原理-几何



假设绿色方块为正类：标签值y=1，红色三角形为负类：标签值y=0。从几何关系上理解，如果我们有一条直线，其公式为：$z=w'\cdot x1+b'$，如图中的虚线L1所示，则所有正类的样本的x2都大于z，而所有的负类样本的x2都小于z，那么这条直线就是我们需要的分割线。

这就说明神经网络的工作原理和我们在二维平面上的直观感觉是相同的，即神经网络的工作就是找到这么一条合适的直线，尽量让所有正例样本都处于直线上方时，负例样本处于直线的下方。其实这与线性回归中找到一条直线穿过所有样本点的过程有异曲同工之处。

我们还有一个额外的收获，即：

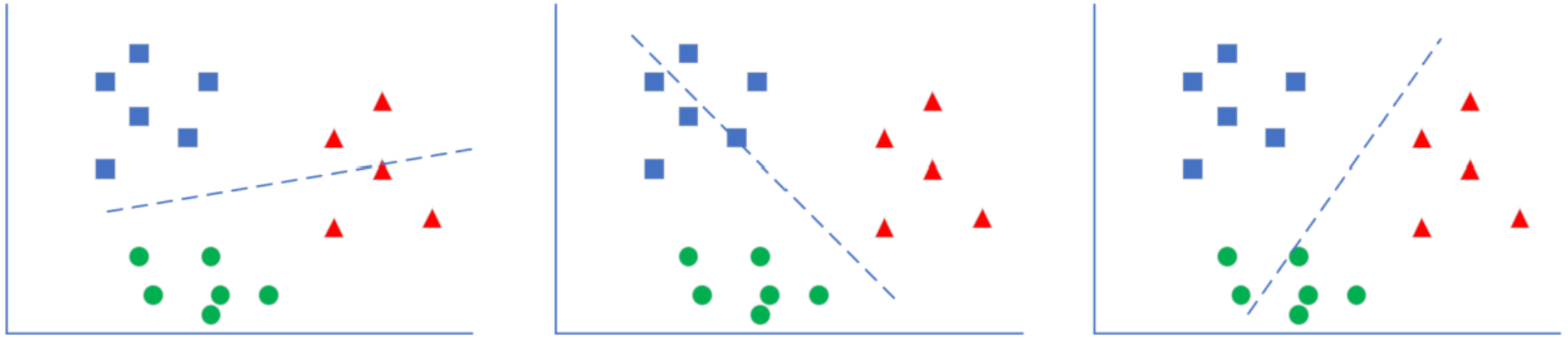
$$w'=-w1/w2(7)$$

$$b'=-b/w2(8)$$

我们可以使用神经网络计算出w1，w2，bw_1，w_2，bw1，w2，b三个值以后，换算成w'，b'w'，b'w'，b'，以便在二维平面上画出分割线，来直观地判断神经网络训练结果的正确性。

多分类问题解法 - 一对一方式

每次先只保留两个类别的数据，训练一个分类器。如果一共有N个类别，则需要训练C_N^2个分类器。以N=3时举例，需要训练A|B，B|C，A|CA|B，B|C，A|CA|B，B|C，A|C三个分类器。

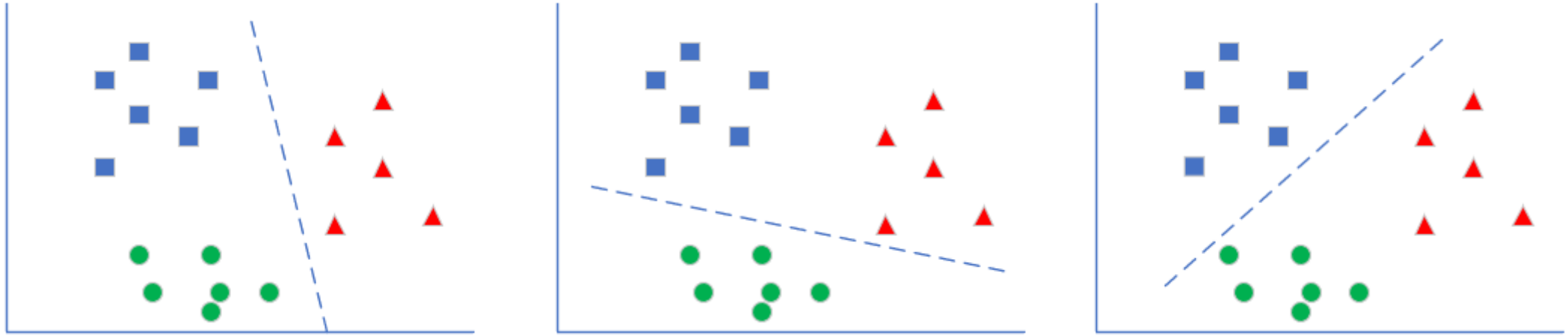


推理时，(A|B)分类器告诉你是A类时，需要到(A|C)分类器再试一下，如果也是A类，则就是A类。如果(A|C)告诉你是C类，则基本是C类了，不可能是B类，不信的话可以到(B|C)分类器再去测试一下。

多分类问题解法 - 一对多方式

处理一个类别时，暂时把其它所有类别看作是一类，这样对于三分类问题，可以得到三个分类器

这种情况是在训练时，把红色样本当作一类，把蓝色和绿色样本混在一起当作另外一类。



同时调用三个分类器，再把三种结果组合起来，就是真实的结果。比如，第一个分类器告诉你是“红类”，那么它确实就是红类；如果告诉你是非红类，则需要看第二个分类器的结果，绿类或者非绿类；依此类推。

多分类问题解法 – 多对多方式

假设有4个类别ABCD，我们可以把AB算作一类，CD算作一类，训练一个分类器1；再把AC算作一类，BD算作一类，训练一个分类器2。

推理时，第1个分类器告诉你是AB类，第二个分类器告诉你是BD类，则做“与”操作，就是B类。

多分类与多标签

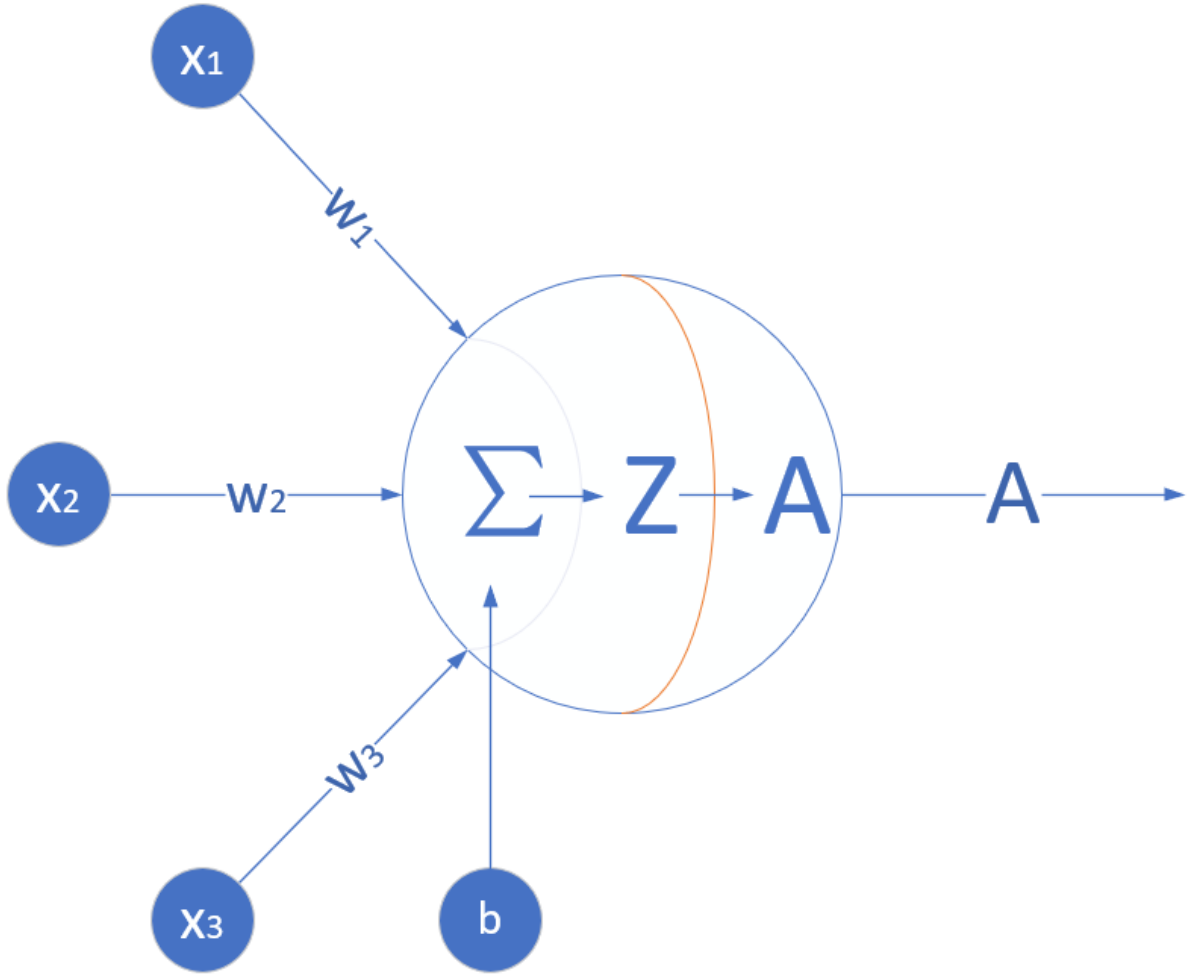
多分类学习中，虽然有多个类别，但是每个样本只属于一个类别。
有一种情况也很常见，比如一幅图中，既有蓝天白云，又有花草树木，那么这张图片可以有两种标注方法：

•标注为“风景”，而不是“人物”，属于风景图片，这叫做分类

•被同时标注为“蓝天”、“白云”、“花草”、“树木”等多个标签，这样的任务不叫作多分类学习，而是“多标签”学习，multi-label learning。我们此处不涉及这类问题。

激活函数介绍

激活函数的作用



假设该神经元有三个输入，分别为 x_1, x_2, x_3 ，那么：

$$z = x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

$$a = \sigma(z)$$

给神经网络增加非线性因素

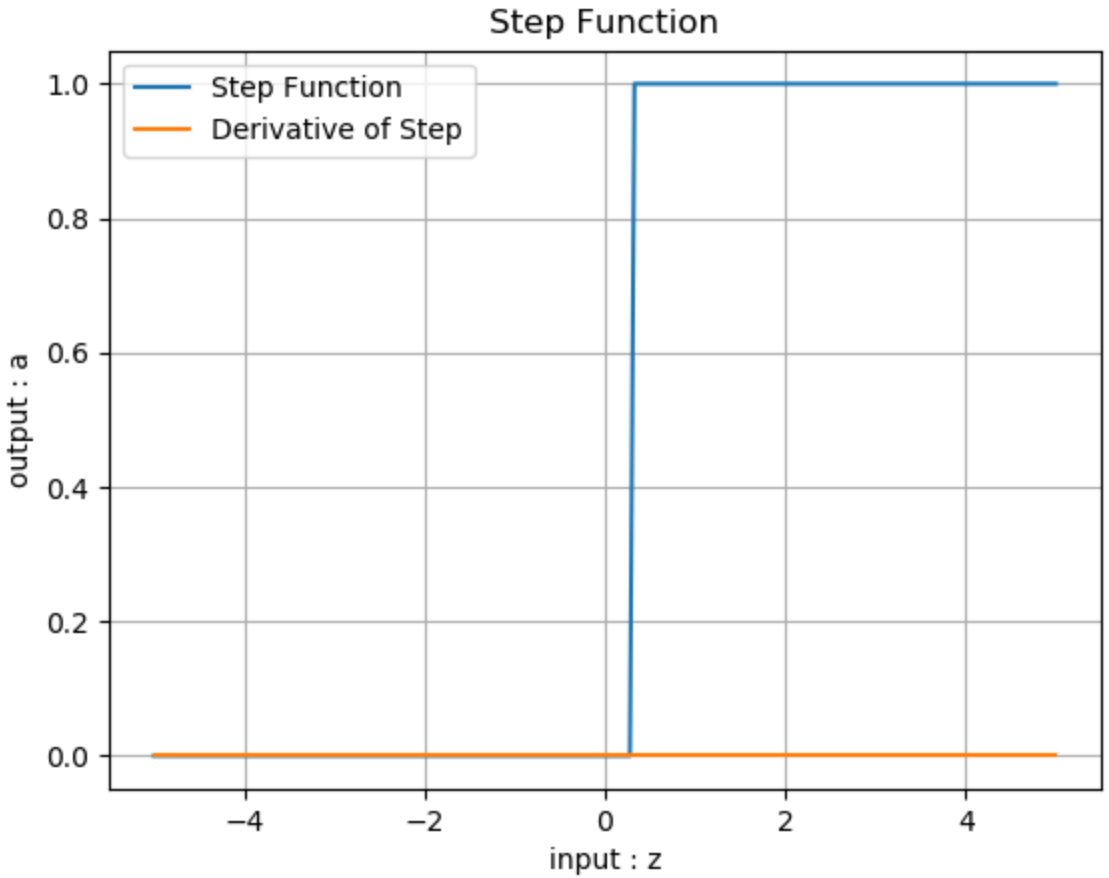
把公式1的计算结果压缩到[0,1]之间，便于后面的计算。

激活函数的基本性质

- 非线性：线性的激活函数和没有激活函数一样；
- 可导性：做误差反向传播和梯度下降，必须要保证激活函数的可导性；
- 单调性：单一的输入会得到单一的输出，较大值的输入得到较大值的输出。

场景

在物理试验中使用的继电器，是最初的激活函数的原型：当输入电流大于一个阈值时，会产生足够的磁场，从而打开下一级电源通道



这个Step函数有什么不好的地方呢？主要的一点就是，他的梯度（导数）恒为零（个别点除外）。反向传播公式中，梯度传递用到了链式法则，如果在这样一个连乘的式子其中有一项是零，这样的梯度就会恒为零，是没有办法进行反向传播的。

用1来代表一个神经元被激活，0代表一个神经元未被激活

激活函数的应用场景

激活函数用在神经网络的层与层之间的连接，神经网络的最后一层不用激活函数。

神经网络不管有多少层，最后的输出层决定了这个神经网络能干什么。

网络	输入	输出	激活函数	分类函数	功能
单层	单变量	单输出	无	无	线性回归
单层	多变量	单输出	无	无	线性回归
单层	多变量	单输出	无	二分类函数	二分类
单层	多变量	多输出	无	多分类函数	多分类

单层的神经网络的参数与功能

网络	输入	输出	激活函数	分类函数	功能
单层	单变量	单输出	无	无	线性回归
单层	多变量	单输出	无	无	线性回归
单层	多变量	单输出	无	二分类函数	二分类
单层	多变量	多输出	无	多分类函数	多分类

我们一直没有使用激活函数，而只使用了分类函数。对于多层神经网络也是如此，在最后一层只会用到分类函数来完成二分类或多分类任务，如果是拟合任务，则不需要分类函数

一些小结

- 神经网络最后一层不需要激活函数
- 激活函数只用于连接前后两层神经网络

挤压型激活函数

挤压型激活函数介绍

这一类函数的特点是，当输入值域的绝对值较大的时候，其输出在两端是饱和的，都具有S形的函数曲线以及压缩输入值域的作用，所以叫挤压型激活函数，又可以叫饱和型激活函数。

在英文中，通常用Sigmoid来表示，原意是S型的曲线，在数学中是指一类具有压缩作用的S型的函数，在神经网络中，有两个常用的Sigmoid函数，一个是Logistic函数，另一个是Tanh函数。

Logistic函数

对数几率函数（Logistic Function，简称对率函数）

很多文字材料中通常把激活函数和分类函数混淆在一起说，有一个原因是：在二分类任务中最后一层使用的对率函数与在神经网络层与层之间连接的Sigmoid激活函数，是同样的形式。所以它既是激活函数，又是分类函数，是个特例。

凡是用到“Logistic”词汇的，指的是二分类函数；而用到“Sigmoid”词汇的，指的是本激活函数。

Logistic函数公式

公式

$$Sigmoid(z) = \frac{1}{1+e^{-z}} \rightarrow a$$

导数

$$Sigmoid'(z) = a(1-a)$$

Logistic函数推导方式

推导过程如下：

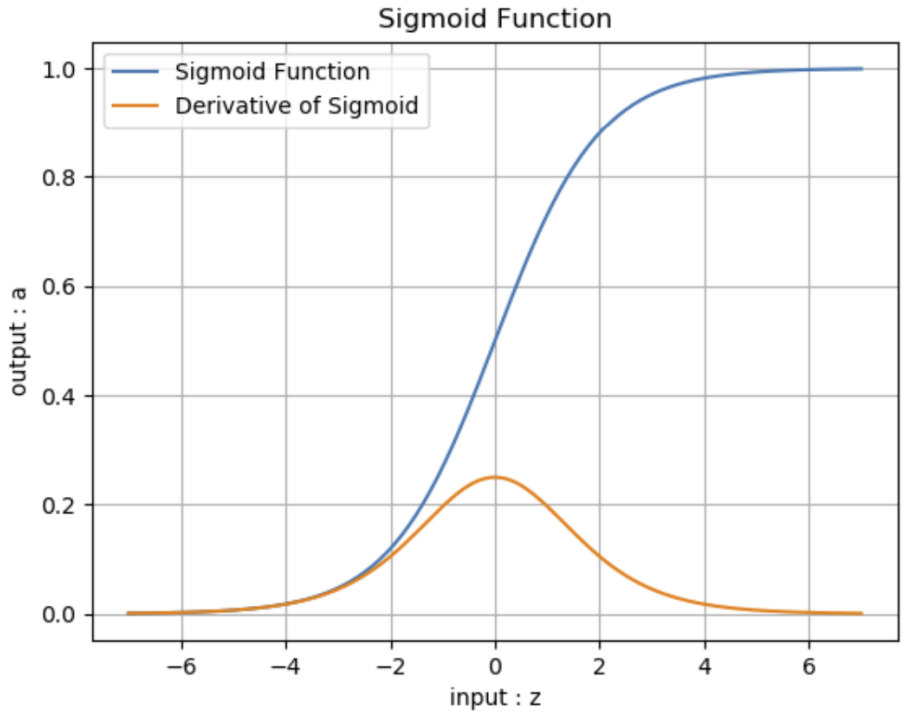
令：$u=1$，$v=1+e^{-z}$ 则：

$$
\begin{aligned}
Sigmoid'(z)=\frac{u'v-v'u}{v^2}&=\frac{0-(1+e^{-z})'}{(1+e^{-z})^2}\\
&=\frac{e^{-z}}{(1+e^{-z})^2}=\frac{1+e^{-z}-1}{(1+e^{-z})^2}\\
&=\frac{1}{1+e^{-z}}-\left(\frac{1}{1+e^{-z}}\right)^2\\
&=a-a^2=a(1-a)
\end{aligned}
$$

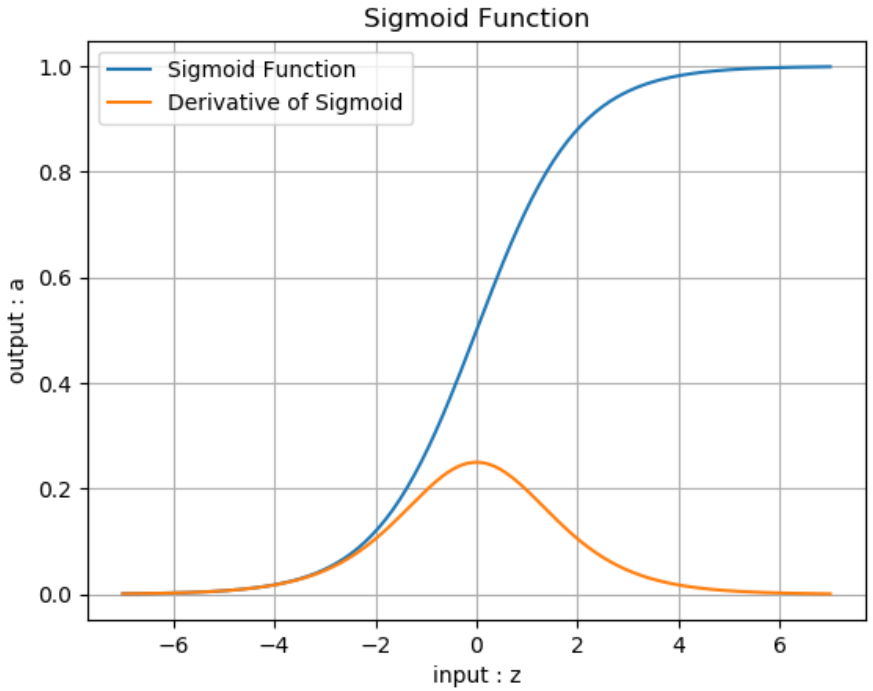
值域

- 输入值域：$(-\infty, \infty)$
- 输出值域：$(0, 1)$
- 导数值域：$[0, 0.25]$

函数图像



Logistic函数的优点



从函数图像来看，Sigmoid函数的作用是将输入压缩到(0, 1)这个区间范围内，这种输出在0~1之间的函数可以用来模拟一些概率分布的情况。他还是一个连续函数，导数简单易求。

从数学上来看，Sigmoid函数对中央区的信号增益较大，对两侧区的信号增益小，在信号的特征空间映射上，有很好的效果。

从神经科学上来看，中央区酷似神经元的兴奋态，两侧区酷似神经元的抑制态，因而在神经网络学习方面，可以将重点特征推向中央区， 将非重点特征推向两侧区。

分类功能：我们经常听到这样的对白：
- 甲： “你觉得这件事情成功概率有多大？”
- 乙： “我有六成把握能成功。”

Sigmoid函数在这里就起到了如何把一个数值转化成一个通俗意义上的“把握”的表示。z坐标值越大，经过Sigmoid函数之后的结果就越接近1，把握就越大。

Logistic函数的缺点

指数计算代价大。

反向传播时梯度消失：从梯度图像中可以看到，Sigmoid的梯度在两端都会接近于0，根据链式法则，如果传回的误差是 δ，那么梯度传递函数是 $\delta \cdot a'$，而 a' 这时接近零，也就是说整体的梯度也接近零。这就出现梯度消失的问题，**并且这个问题可能导致网络收敛速度比较慢。**

给个纯粹数学的例子，假定我们的学习速率是0.2，Sigmoid函数值是0.9（处于饱和区了），如果我们想把这个函数的值降到0.5，需要经过多少步呢?

我们先来做数值计算：

1. 求出当前输入的值

$$a = \frac{1}{1+e^{-z}} = 0.9$$

$$z = ln9$$

1. 求出当前梯度

$$\delta = a \times (1-a) = 0.9 \times 0.1 = 0.09$$

1. 根据梯度更新当前输入值

$$z_{new} = z - \eta \times \delta = ln9 - 0.2 \times 0.09 = ln(9) - 0.018$$

1. 判断当前函数值是否接近0.5

$$a = \frac{1}{1+e^{-z_{new}}} = 0.898368$$

1. 重复步骤2-3，直到当前函数值接近0.5

67次

Tanh函数 TanHyperbolic，即双曲正切函数

公式

$$Tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = (\frac{2}{1+e^{-2z}} - 1) \to a$$

即

$$Tanh(z) = 2 \cdot Sigmoid(2z) - 1$$

导数公式

$$Tanh'(z) = (1+a)(1-a)$$

利用基本导数公式23，令：$u = e^z - e^{-z}$，$v = e^z + e^{-z}$ 则有：

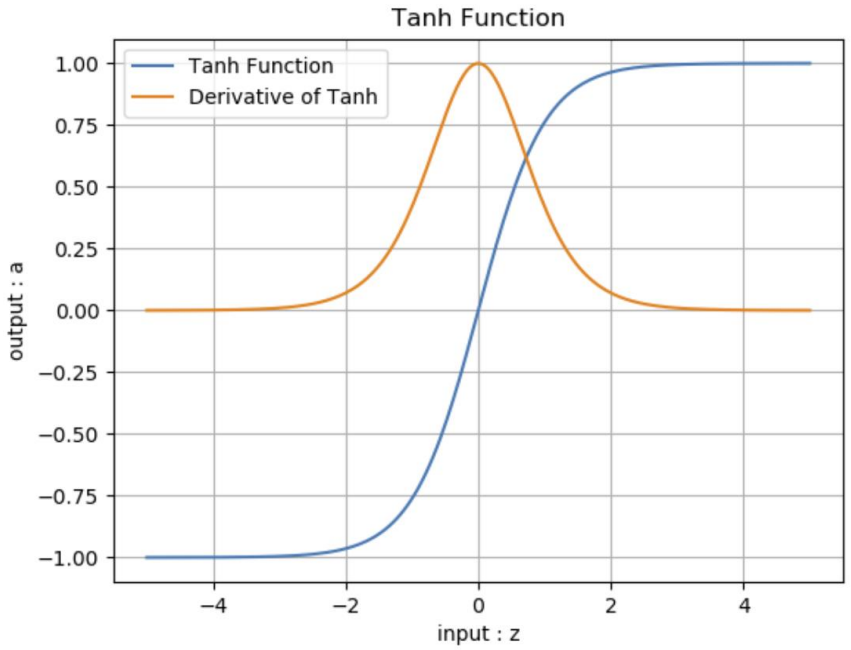
$$\begin{aligned} Tanh'(z) &= \frac{u'v - v'u}{v^2} \\ &= \frac{(e^z - e^{-z})'(e^z + e^{-z}) - (e^z + e^{-z})'(e^z - e^{-z})}{(e^z + e^{-z})^2} \\ &= \frac{(e^z + e^{-z})(e^z + e^{-z}) - (e^z - e^{-z})(e^z - e^{-z})}{(e^z + e^{-z})^2} \\ &= \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2}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

值域

- 输入值域：$(-\infty, \infty)$
- 输出值域：$(-1, 1)$
- 导数值域：$[0, 1]$

函数图像

图8-4是双曲正切的函数图像。

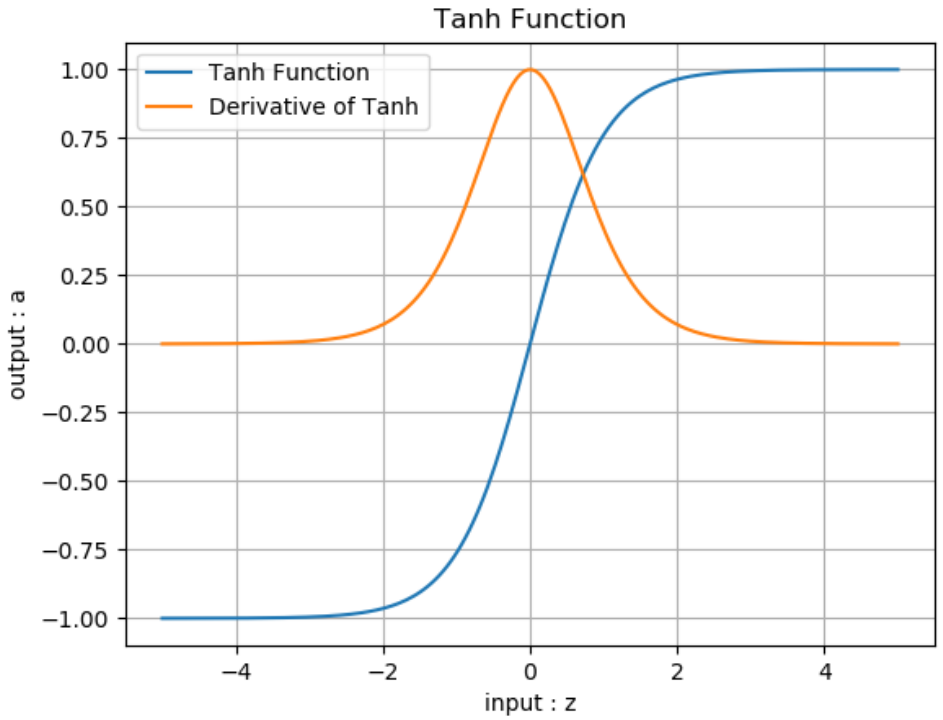


Tanh函数优缺点

优点

具有Sigmoid的所有优点。

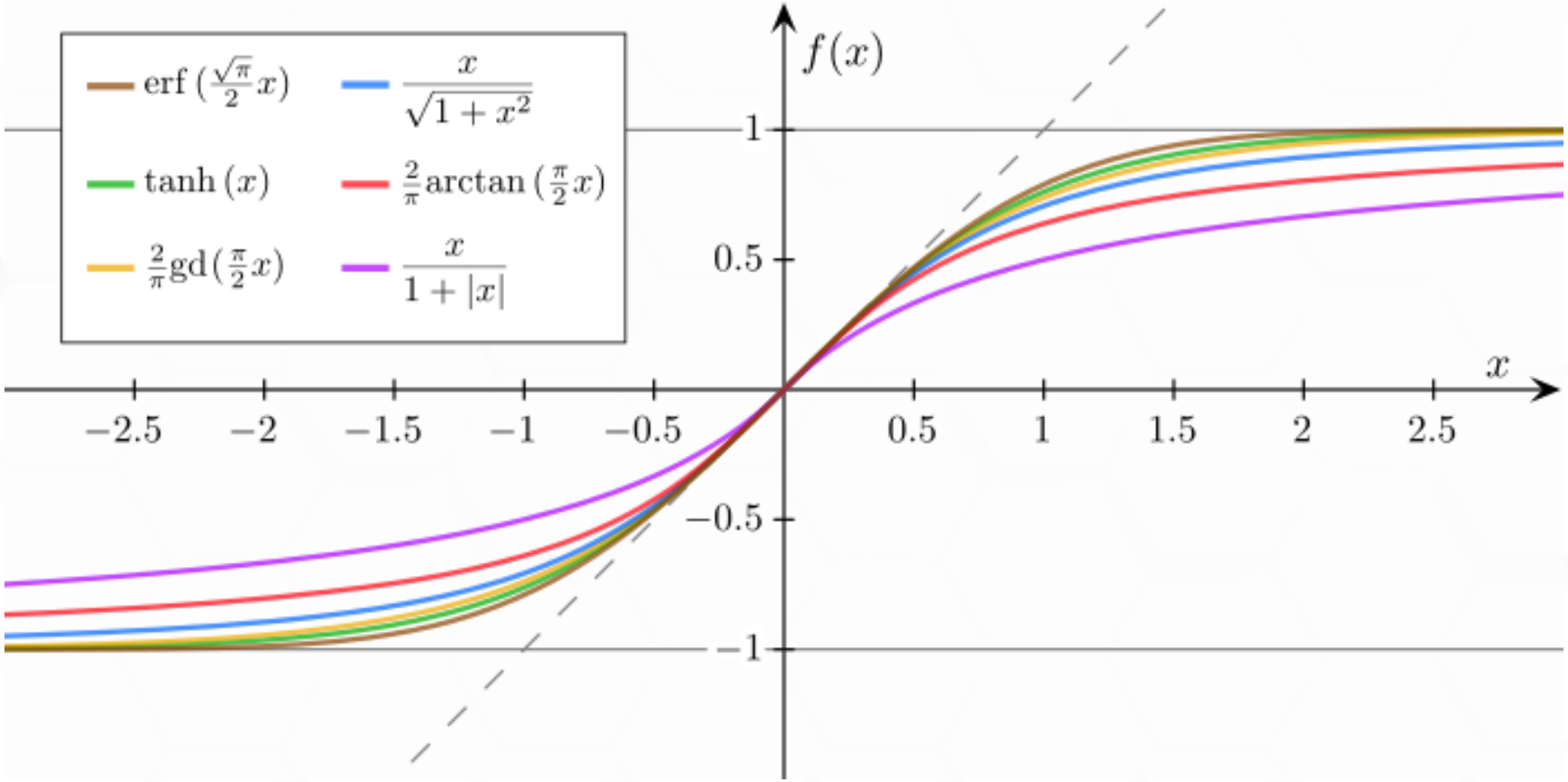
无论从理论公式还是函数图像，这个函数都是一个和Sigmoid非常相像的激活函数，他们的性质也确实如此。但是比起sigmoid，tanh减少了一个缺点，就是他本身是零均值的，也就是说，在传递过程中，输入数据的均值并不会发生改变，这就使他在很多应用中能表现出比Sigmoid优异一些的效果。



缺点

exp指数计算代价大。梯度消失问题仍然存在。

其他函数



常用的只有Sigmoid和Tanh两个

挤压型激活函数

ReLU函数

Rectified Linear Unit，修正线性单元，线性整流函数，斜坡函数。

公式

$$ReLU(z) = \max(0, z) = \left\{ \begin{array}{ll} z & (z \geq 0) \\ 0 & (z < 0) \end{array} \right\}$$

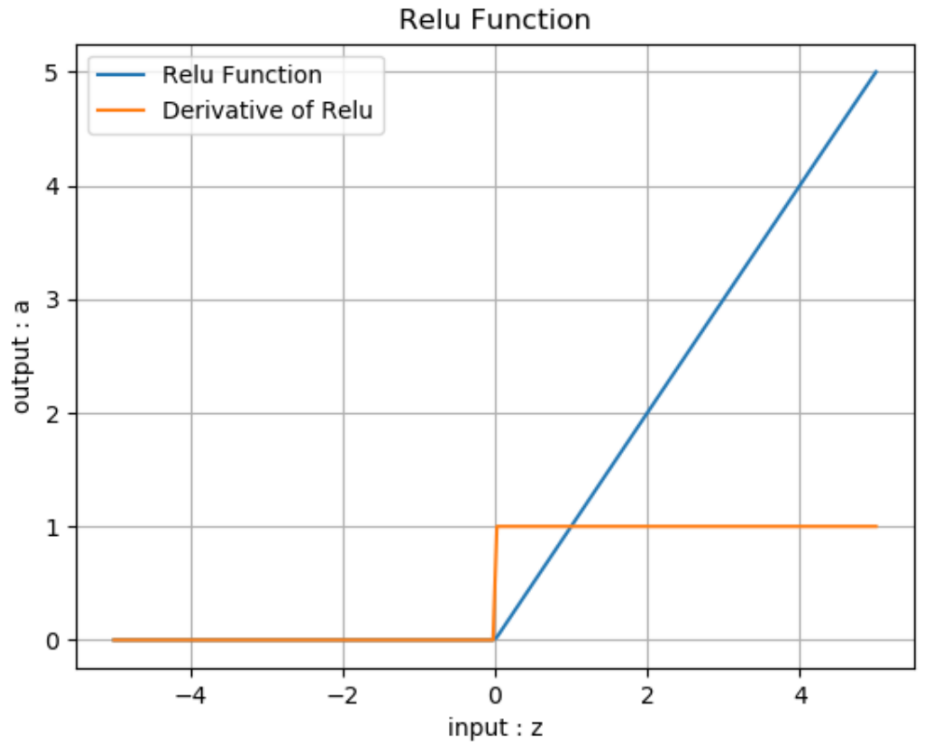
导数

$$ReLU'(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

ReLU函数

值域

- 输入值域：$(-\infty, \infty)$
- 输出值域：$(0, \infty)$
- 导数值域：$[0, 1]$



仿生学原理

相关大脑方面的研究表明生物神经元的信息编码通常是比较分散及稀疏的。通常情况下，大脑中在同一时间大概只有1%~4%的神经元处于活跃状态。使用线性修正以及正则化可以对机器神经网络中神经元的活跃度（即输出为正值）进行调试；相比之下，Sigmoid函数在输入为0时输出为0.5，即已经是半饱和的稳定状态，不够符合实际生物学对模拟神经网络的期望。不过需要指出的是，一般情况下，在一个使用修正线性单元（即线性整流）的神经网络中大概有50%的神经元处于激活态。

ReLU函数的优缺点

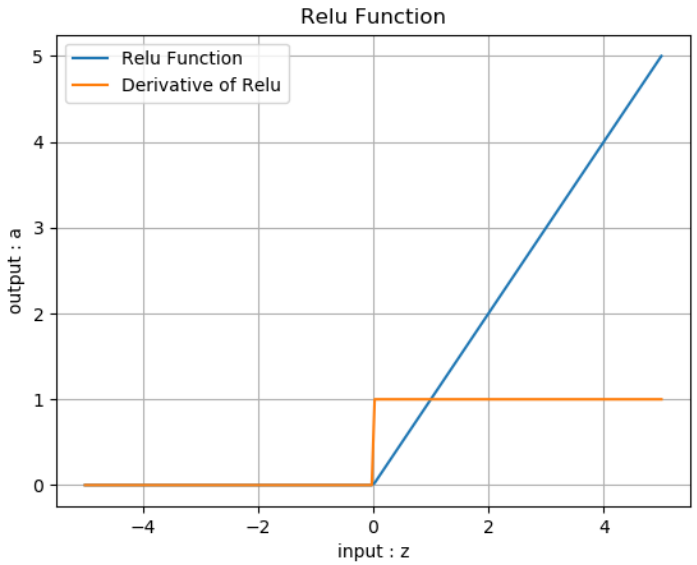
优点

•反向导数恒等于1，更加有效率的反向传播梯度值，收敛速度快；

•避免梯度消失问题；

•计算简单，速度快；

•活跃度的分散性使得神经网络的整体计算成本下降。



缺点

无界。

注意

梯度很大的时候可能导致的神经元“死”掉。

这个死掉的原因是什么呢？是因为很大的梯度导致更新之后的网络传递过来的输入是小于零的，从而导致ReLU的输出是0，计算所得的梯度是零，然后对应的神经元不更新，从而使ReLU输出恒为零，对应的神经元恒定不更新，等于这个ReLU失去了作为一个激活函数的作用。问题的关键点就在于输入小于零时，ReLU回传的梯度是零，从而导致了后面的不更新。在学习率设置不恰当的情况下，很有可能网络中大部分神经元“死”掉，也就是说不起作用了。

和Sigmoid函数对比

用和Sigmoid函数那里更新相似的算法步骤和参数，来模拟一下ReLU的梯度下降次数，也就是学习率 $\eta = 0.2$，希望函数值从0.9衰减到0.5，这样需要多少步呢?

由于ReLU的导数为1，所以:

$$0.9 - 1 \times 0.2 = 0.7 \; 0.7 - 1 \times 0.2 = 0.5$$

也就是说，同样的学习速率，ReLU函数只需要两步就可以做到Sigmoid需要67步才能达到的数值!

Leaky ReLU函数

LReLU，带泄露的线性整流函数。

公式

$$LReLU(z) = \begin{cases} z & z \geq 0 \\ \alpha * z & z < 0 \end{cases}$$

导数

$$LReLU'(z) = \begin{cases} z & 1 \geq 0 \\ \alpha & z < 0 \end{cases}$$

值域

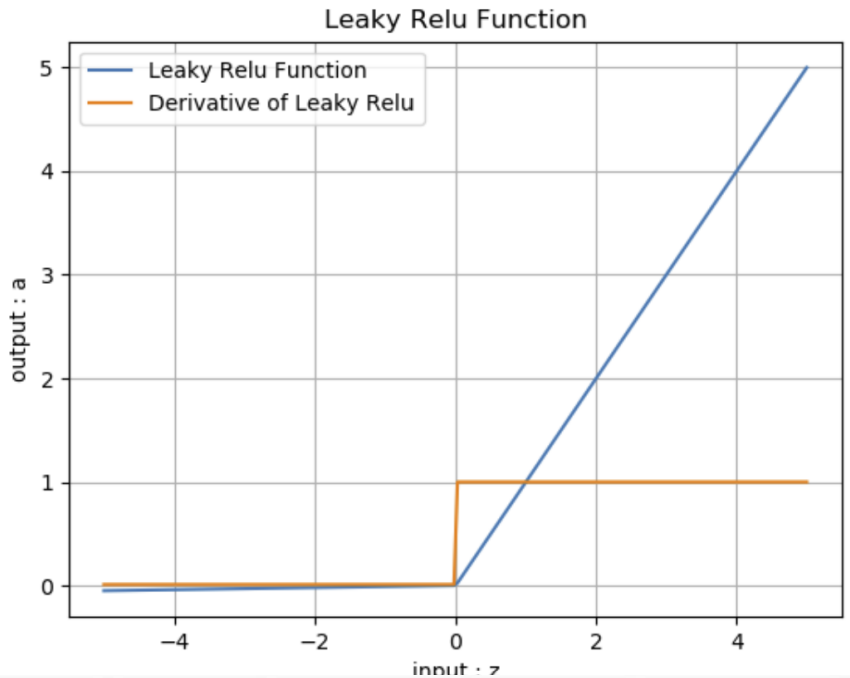
输入值域：$(-\infty, \infty)$

输出值域：$(-\infty, \infty)$

导数值域：$[\alpha, 1]$

函数图像

函数图像如图8-7所示。



优点

继承了ReLU函数的优点。

Leaky ReLU同样有收敛快速和运算复杂度低的优点，而且由于给了 $z < 0$ 时一个比较小的梯度 α ,使得 $z < 0$ 时依旧可以进行梯度传递和更新，可以在一定程度上避免神经元"死"掉的问题。

Softplus函数

公式

$$Softplus(z) = \ln(1 + e^z)$$

导数

$$Softplus'(z) = \frac{e^z}{1 + e^z}$$

Softplus函数

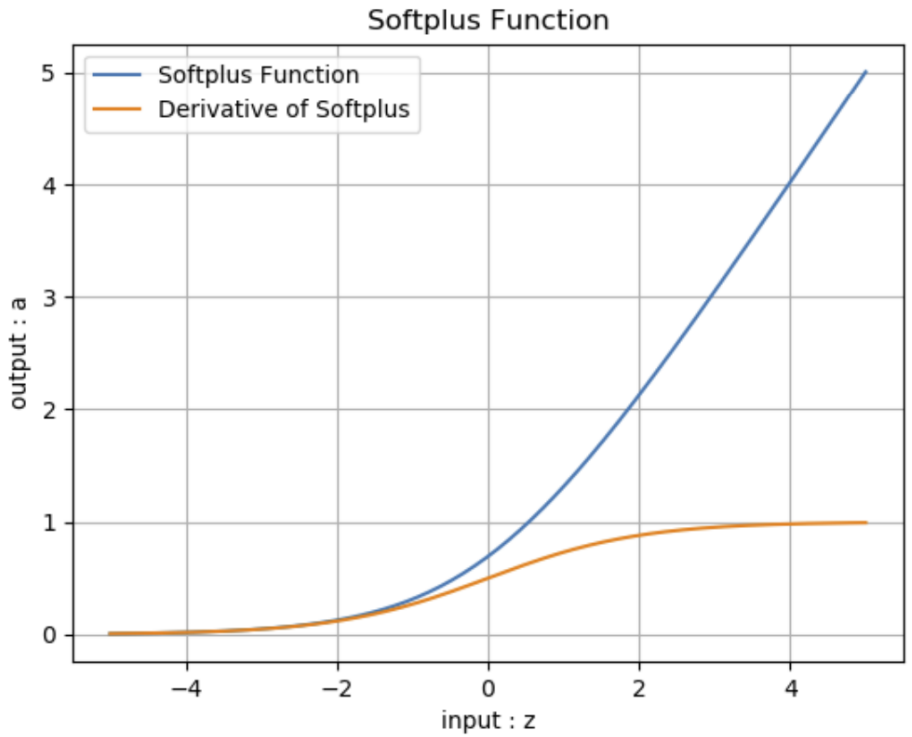
输入值域：$(-\infty, \infty)$

输出值域：$(0, \infty)$

导数值域：$(0, 1)$

函数图像

Softplus的函数图像如图8-8所示。



ELU函数

公式

$$ELU(z) = \begin{cases} z & z \geq 0 \\ \alpha(e^z - 1) & z < 0 \end{cases}$$

导数

$$ELU'(z) = \begin{cases} z & 1 \geq 0 \\ \alpha e^z & z < 0 \end{cases}$$

ELU函数

值域

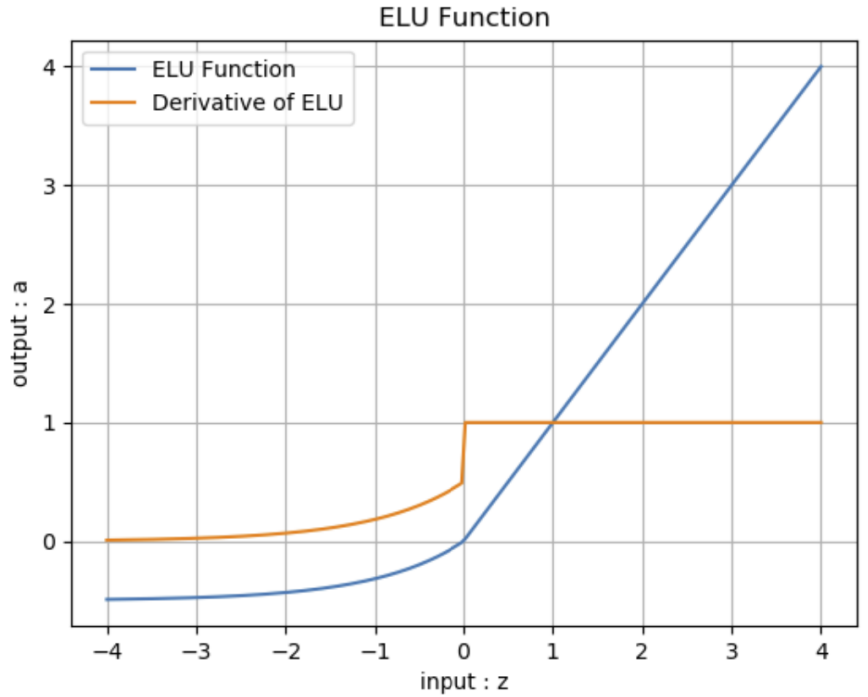
输入值域：$(-\infty, \infty)$

输出值域：$(-\alpha, \infty)$

导数值域：$(0, 1]$

函数图像

ELU的函数图像如图8-9所示。



Microsoft

Reactor

Thank You!