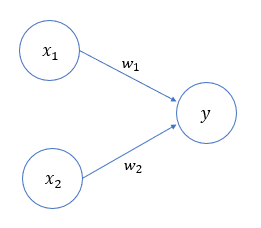
大模型技术之深度学习核心

# 2.神经网络基础

## 神经网络的的构成

### 感知机

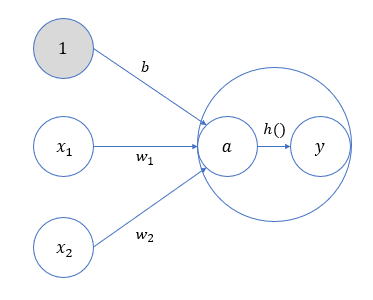
二分类模型，接受多个信号，输出一个信号，且只有0和1的取值；



是输入信号，是输出信号，是权重，○ 称为神经元或节点激活函数

参数，被称为 **偏置；**偏置则可以用来控制神经元被激活的容易程度

### 激活函数



可以将输入信号的加权总和转换为输出信号，起到“激活神经元”的作用，所以被称为 **激活函数**。

如果定义一个函数：

（2.2）

那么式（2.1）就可以简化为：

（2.3）

为了更明显地表示出两个处理步骤，可以进一步写成：

（2.4）

（2.5）

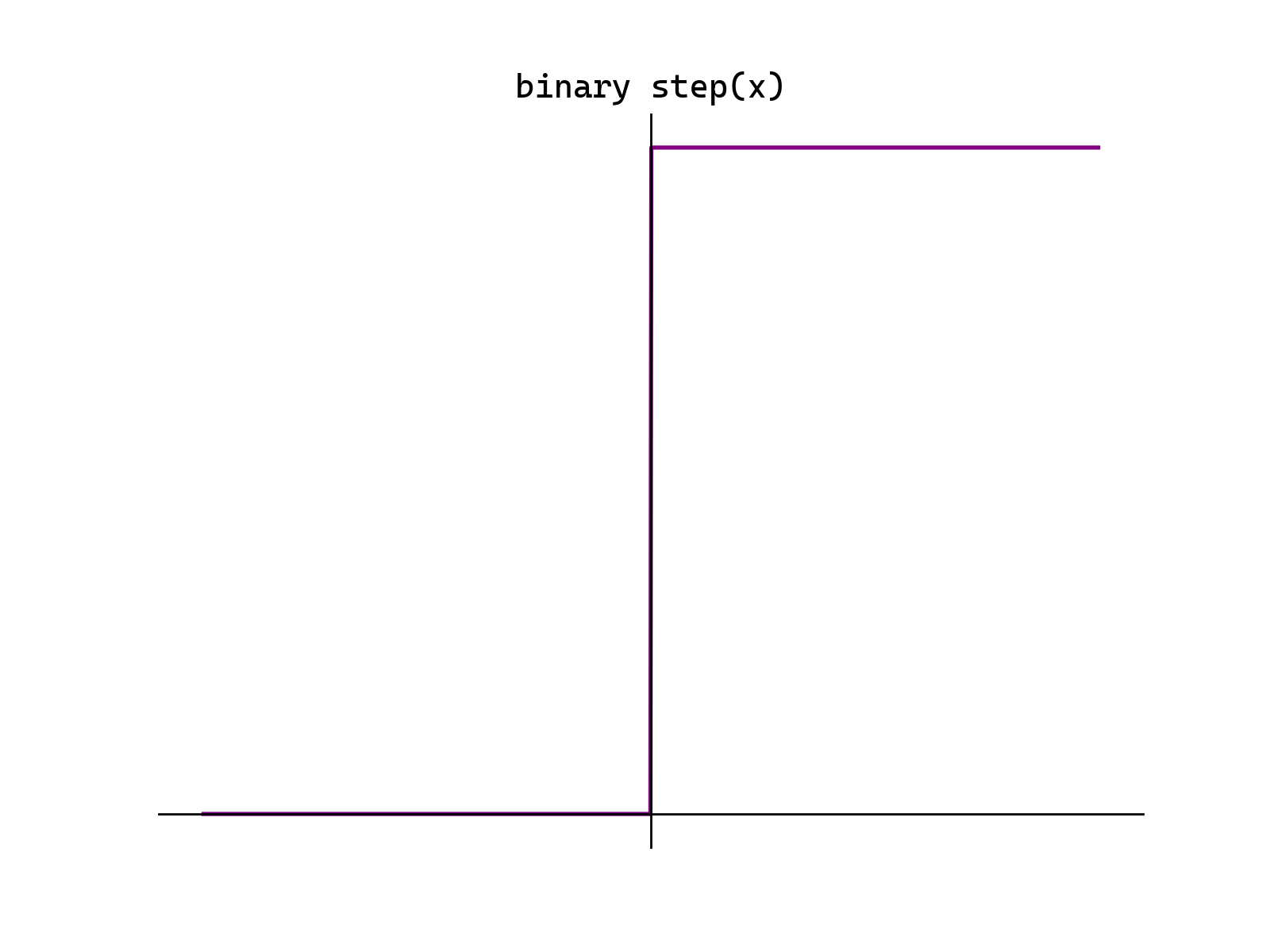
#### 激活函数的作用

a. 激活函数式连接感知机和神经网络的桥梁

b. 激活函数，整个神经网络就等效于单层线性变换，不论如何加深层数，总是存在与之等效的“无隐藏层的神经网络”;

c. 激活函数必须是非线性函数，也正是激活函数的存在为神经网络引入了非线性，使得神经网络能够学习和表示复杂的非线性关系

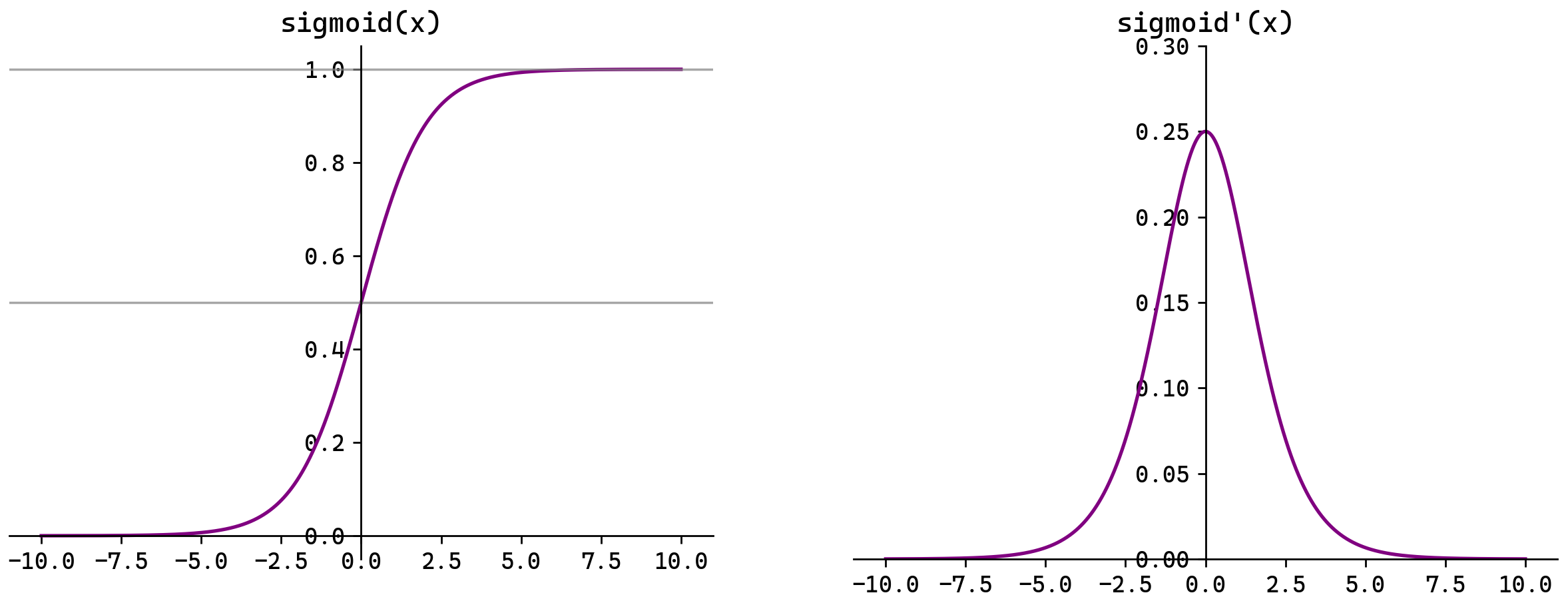
#### 阶跃函数



1. 阶跃函数的导数恒为零，所以应用不多；

2. 解决二分类模型的激活函数

#### Sigmoid函数



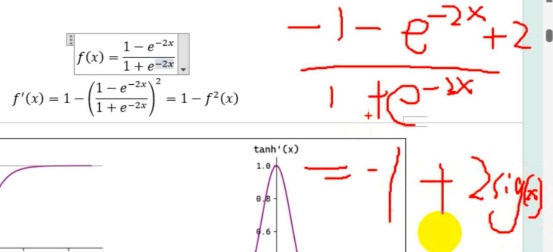
1. Sigmoid的导数范围为(0,0.25)，梯度较小。当输入在[-6,6]之外时，导数接近0，此时网络参数的更新将会极其缓慢，会出现数据丢失。使用Sigmoid作为激活函数，可能出现梯度消失（在逐层反向传播时，梯度会呈指数级衰减，），一般只作为输出层，或者只有两三层的中间层（隐藏层）；

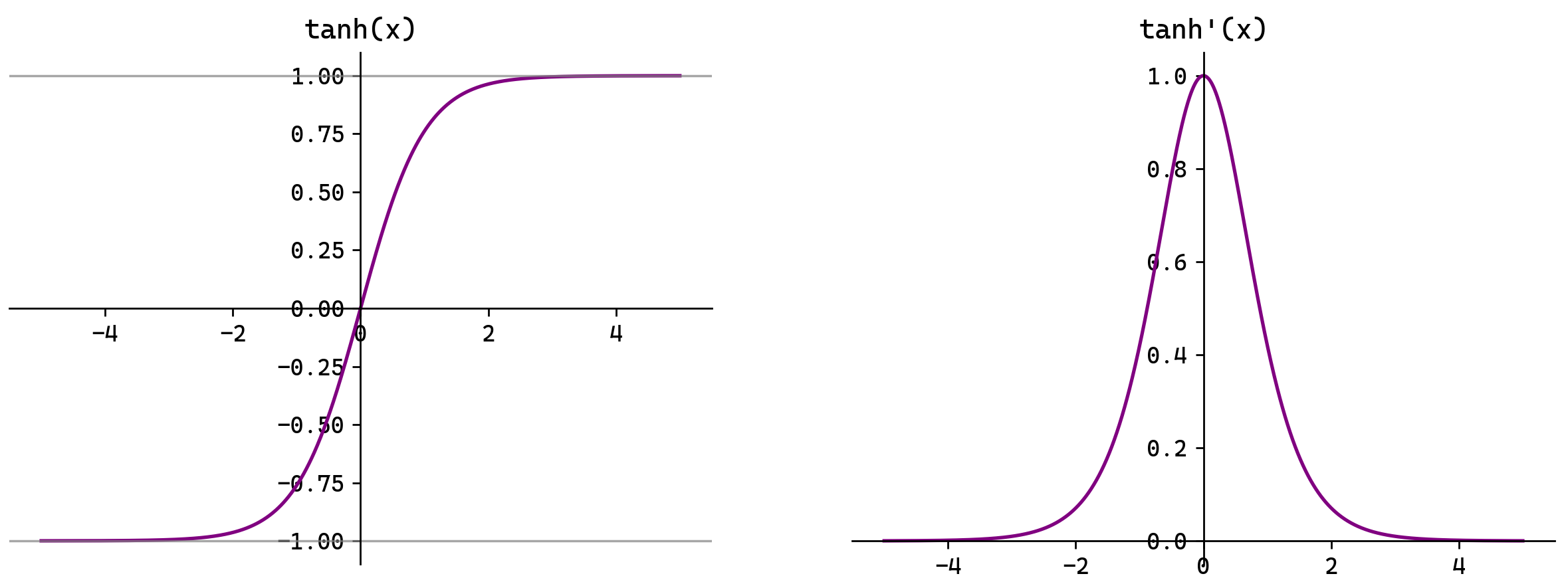
2. 并不是以圆点对称的函数；

3. 解决二分类模型的激活函数

#### Tanh（双曲正切）函数

在sigmod函数上先乘以2再减去1之后，得到的函数；





1. 在sigmoid函数上做的改进，将函数的对称原点，移到了坐标（0,0）的位置；

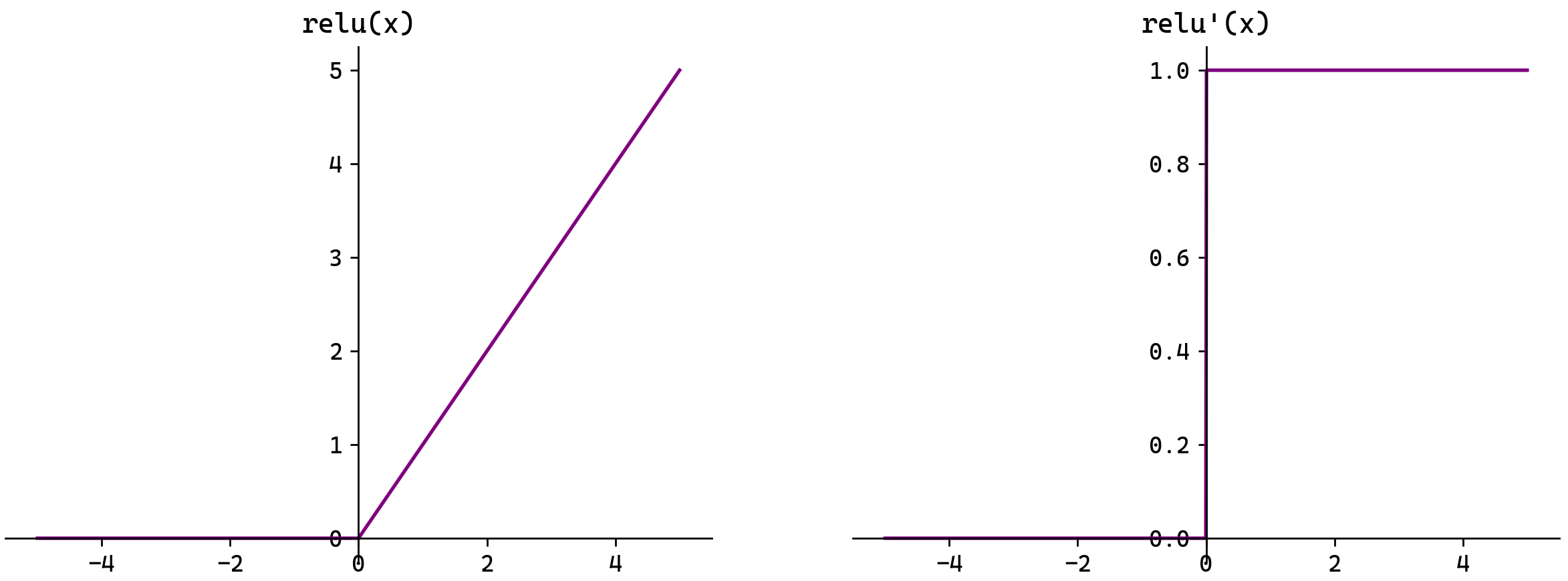
2. Sigmoid的导数范围为(0,1.0)变得更大了，但同样梯度较小

3. Tanh的输出以0为中心，且其梯度相较于Sigmoid更大，收敛速度相对更快。但同样也存在梯度消失现象；

4. 解决二分类模型的激活函数

#### RLU函数

注意：x=0时ReLU函数不可导，此时我们默认使用左侧的函数。



特点：

1. 不存在梯度消失的现象；

2. 当x< 0 ReLU函数输出为0，意味激活节点只是部分“活跃”；这种稀疏性有助于减少计算量和提高模型效率；但同时，由于当x< 0 ReLU函数输出为永远为0，所以会导致大量神经元变成了“死神经元”；“

3. 会出现梯度爆炸问题；

4. 改进：使用leakyReLU函数作为激活函数

Leaky ReLU在负数区域引入一个小的斜率来解决“神经元死亡”问题；

4. 在（中间层）隐藏层使用

#### Softmax函数（重点）

1. 将一个任意的实数向量转换为一个概率分布，并确保输出的值总和为1；

2. 是二分类激活函数sigmoid在多分类的推过；

3. 最大输入值对应的输出概率较大，其他较小的值会被压缩。即在类别之间起到了一定的区分作用。

4. 基本代码待验证

重点指数函数值非常大，对溢出情况的策略；

一维向量操作

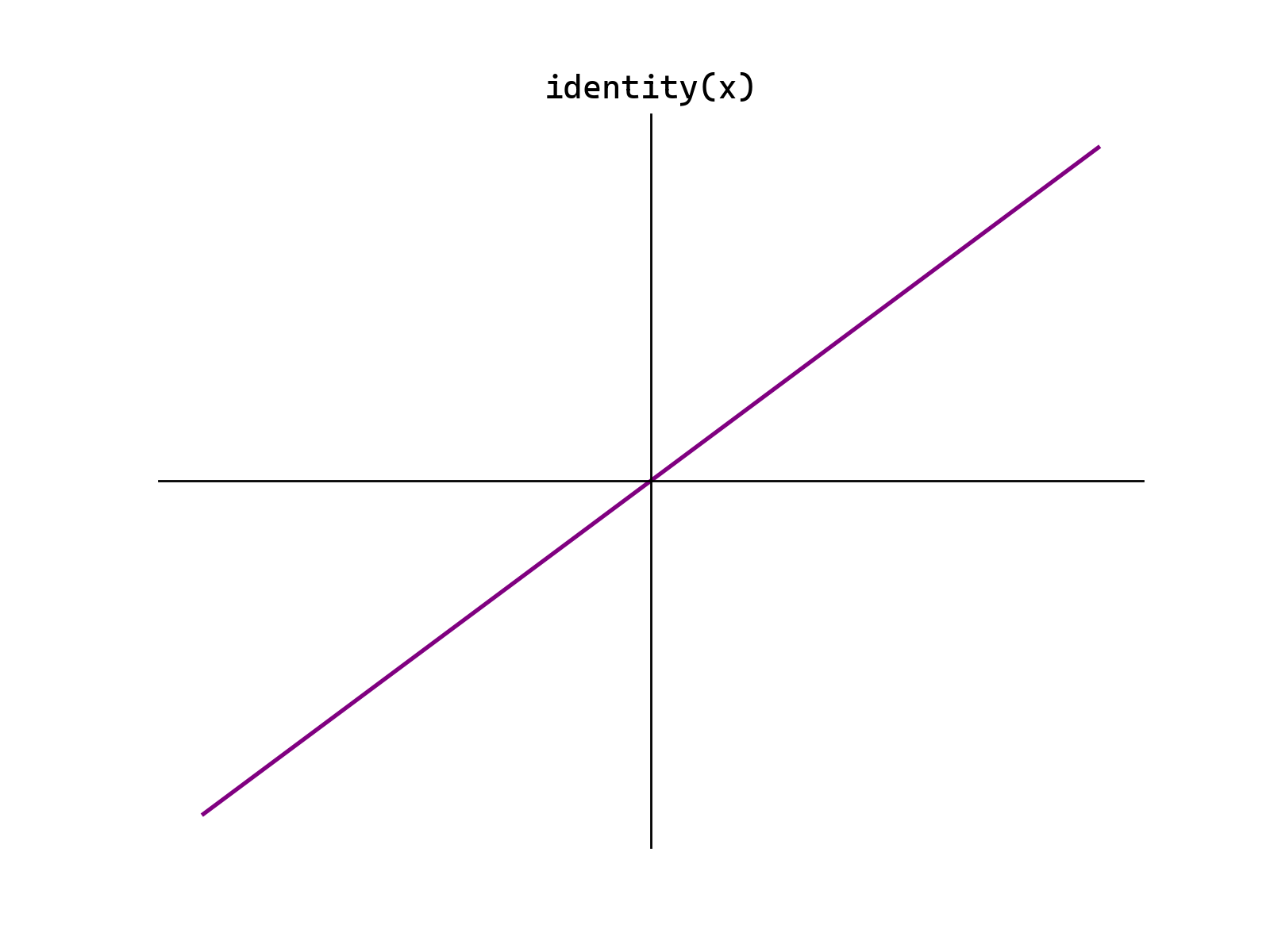
def softmax(x):  
 x = x - np.max(x) *# 溢出对策：指的是向量（分子分母）的每一个元素都减去最大的标量值，结果可能是负值，操作的同时是分母和分子，所以最终结果是不变的；* return np.exp(x) / np.sum(np.exp(x))

而维矩阵操作

def softmax(x):  
 if x.ndim == 2:  
 x = x.T  
 x = x - np.max(x, axis=0)  
 y = np.exp(x) / np.sum(np.exp(x), axis=0)  
 return y.T   
  
 x = x - np.max(x) *# 溢出对策* return np.exp(x) / np.sum(np.exp(x))

#### 其他激活函数

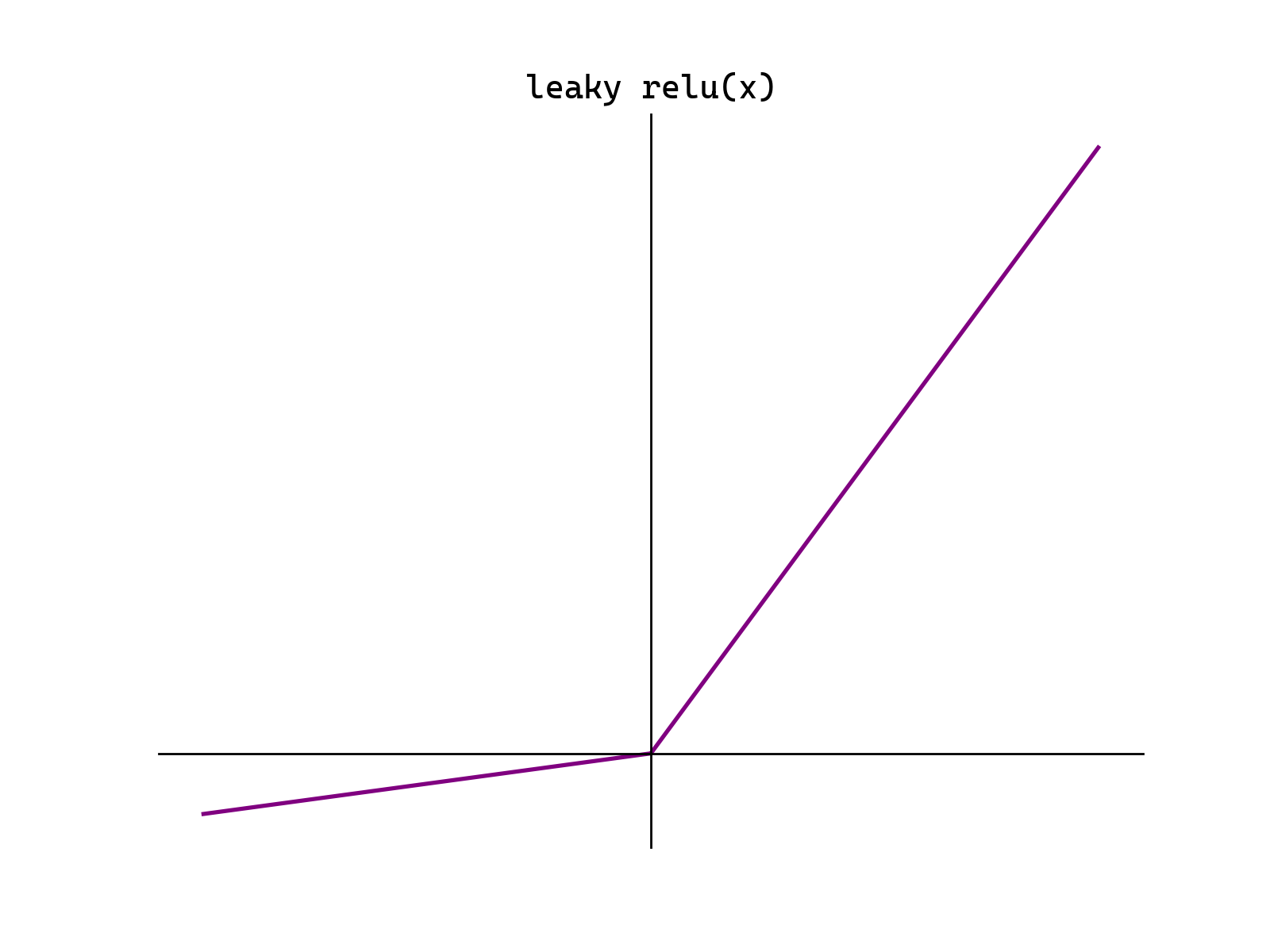
##### 恒等函数（Identity）



导数恒为1，可以作为隐藏层（中间层）；

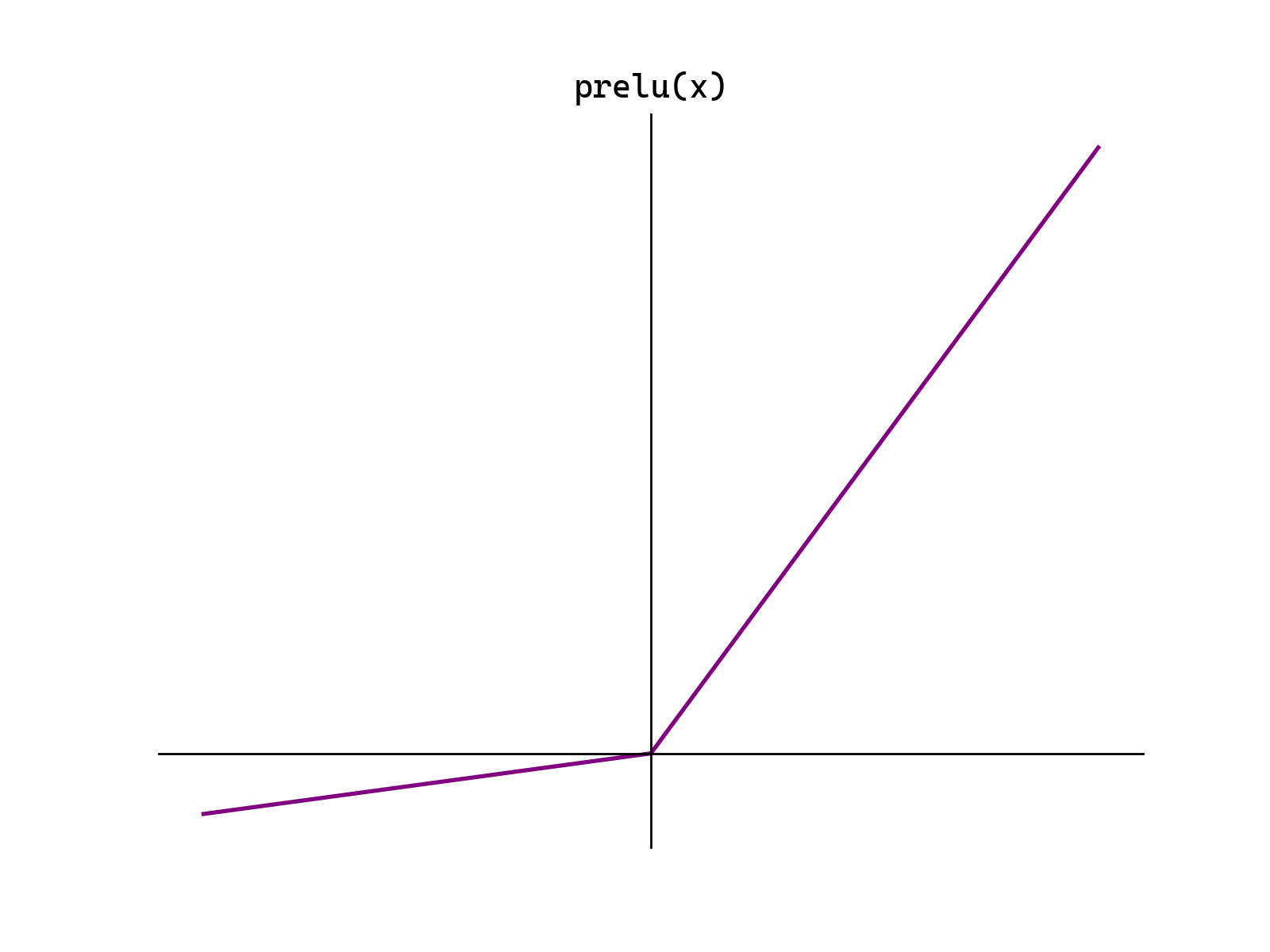
##### Leaky ReLU（Leaky Recttified Linear Unit）

ReLU函数的变体，添加了系数的固定参数



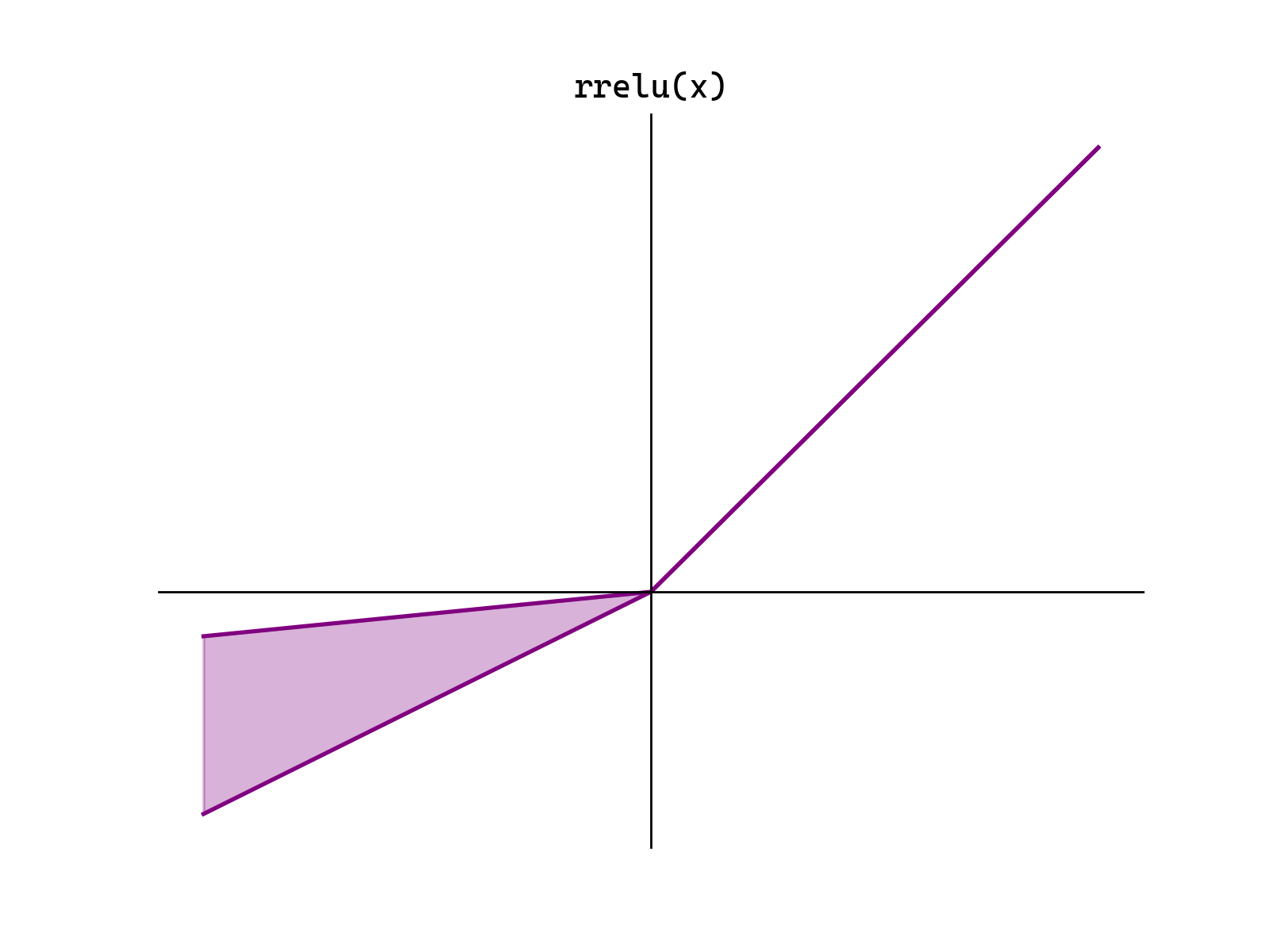
##### PreLU（Parametric Rectified Linear Unit）

这里是一个可训练的参数，而非固定的参数。



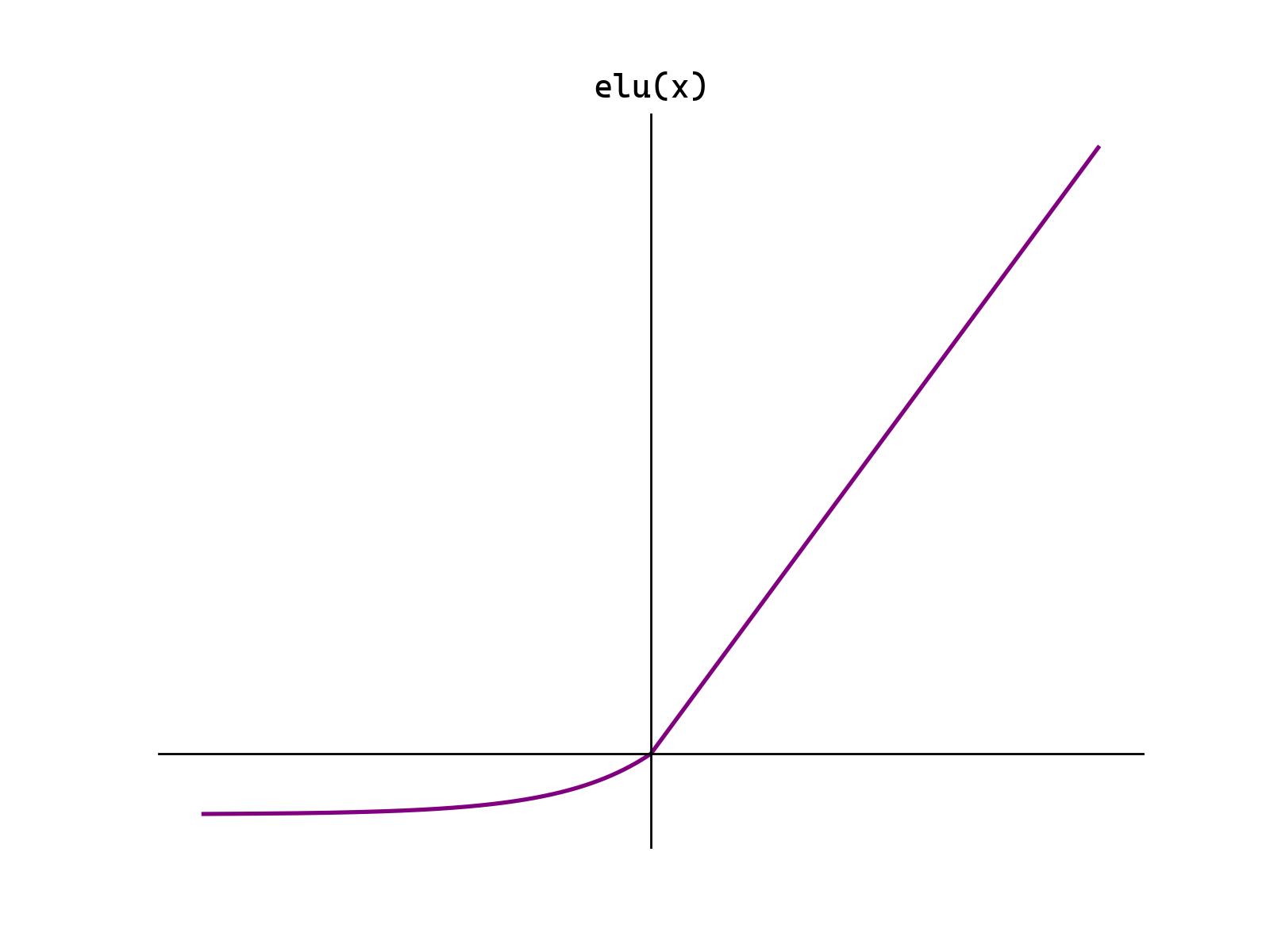
##### RReLU（Randomized Leaky ReLU）

这里是一个在训练时从一个均匀分布中随机选择的参数。



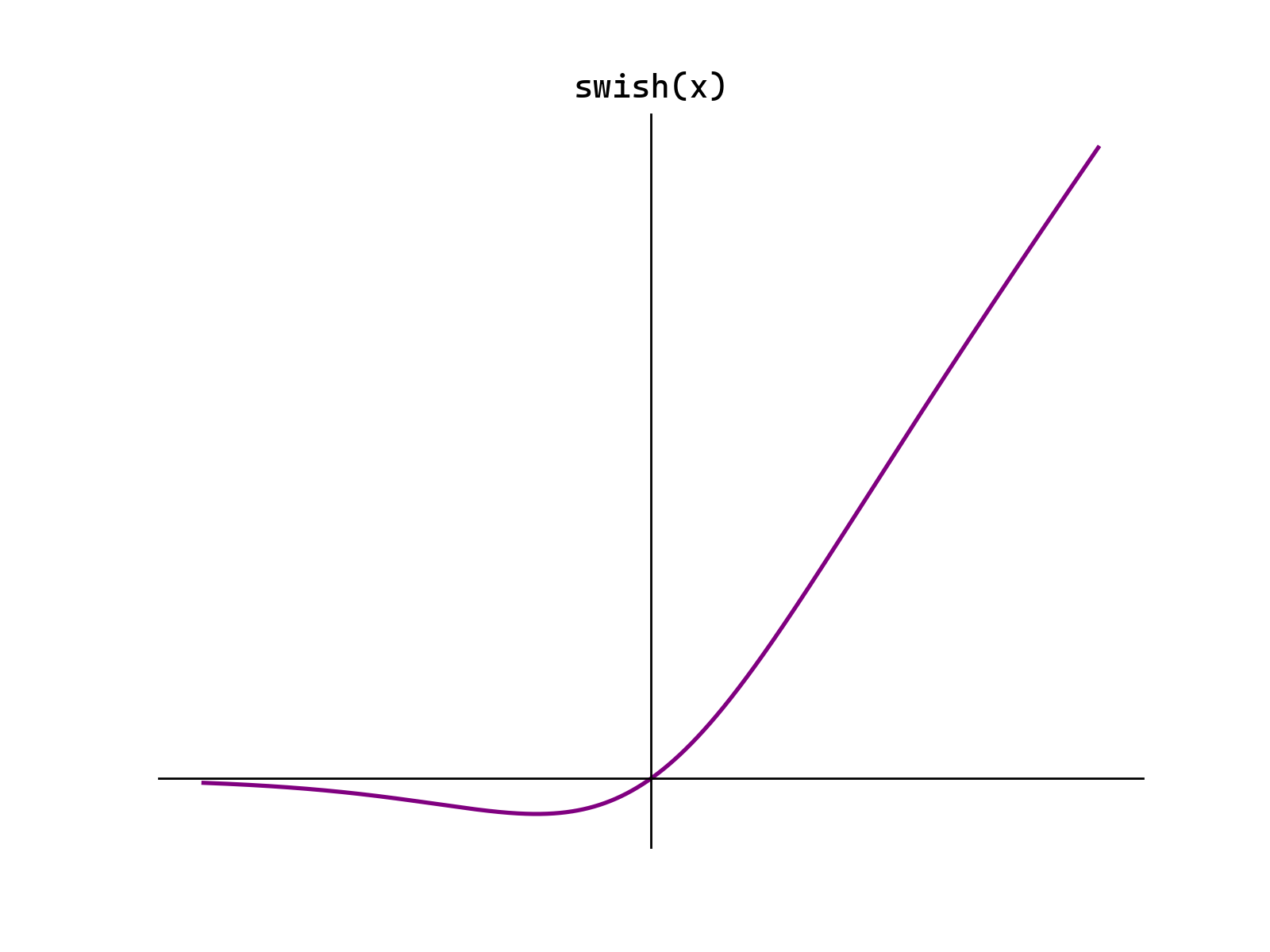
##### ELU(Exponential Linear Unit)

X小于零的函数变成了指数项



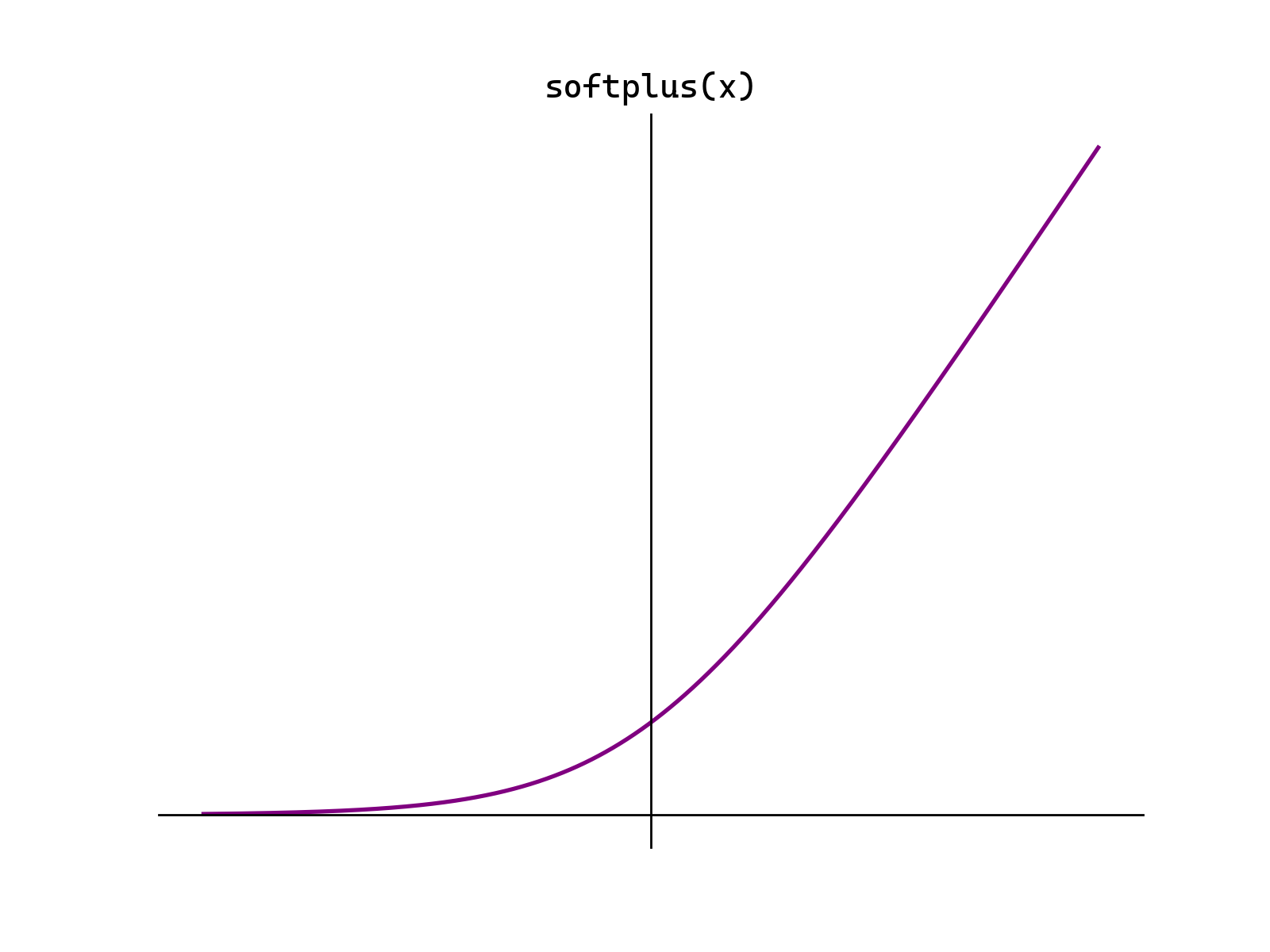
##### Swish(sigmoid Linear Unit, SiLU)

在sigmoid的函数上乘了一个x



##### Softplus

Sigmoid函数的微积分函数



#### 激活函数的选择

##### 隐藏层

首选ReLU函数，效果不好选择Leaky ReLU等 多用于深度层次的网络；

避免出现梯度消失的sigmoid函数，简单的浅层网络；

Tanh函数对称点过原点输出均值为0，对中心化数据更友好，但仍有梯度消失问题，所以，可适用于浅层网络。要比sigmoid好；

##### 输出层

二分类：sigmoid函数

多分类：softmax函数

回归类：Identity函数（恒值函数）

##### 代码实现

待实现

#### 神经网路的简单实现

##### 核心要素

主要因素有：信息传递（线性函数的权重w+偏置b）、神经元（线性函数+激活函数）、前向传递；

特点：每一次信号的传递权重是二维矩阵、偏置是一组向量；

模型的参会主要是权重w 和偏置b 最终要训练的就是这玩意;

##### 代码实现

待实现

#### 应用案例：手写数字识别

1. 全部数据的代码实现

2. 小批量数据测试代码实现

# 3. 神经网络的学习

## 概念特点

1. 是机器学习的一种，跟传统机器学习相比，神经网络不需要人工设置特征量；

2. 神经网络就是让训练数据自动决定最优的权重参数；

## 常见的损失函数

### 均方误差（MSE）

1. 也叫做L2Loss,常用于回归问题的损失函数；

2. 标准公式

其中n的作用不是很大，因为n代表维度，固定维度n的影响很小，可以用下面的变种；

3. 变种公式

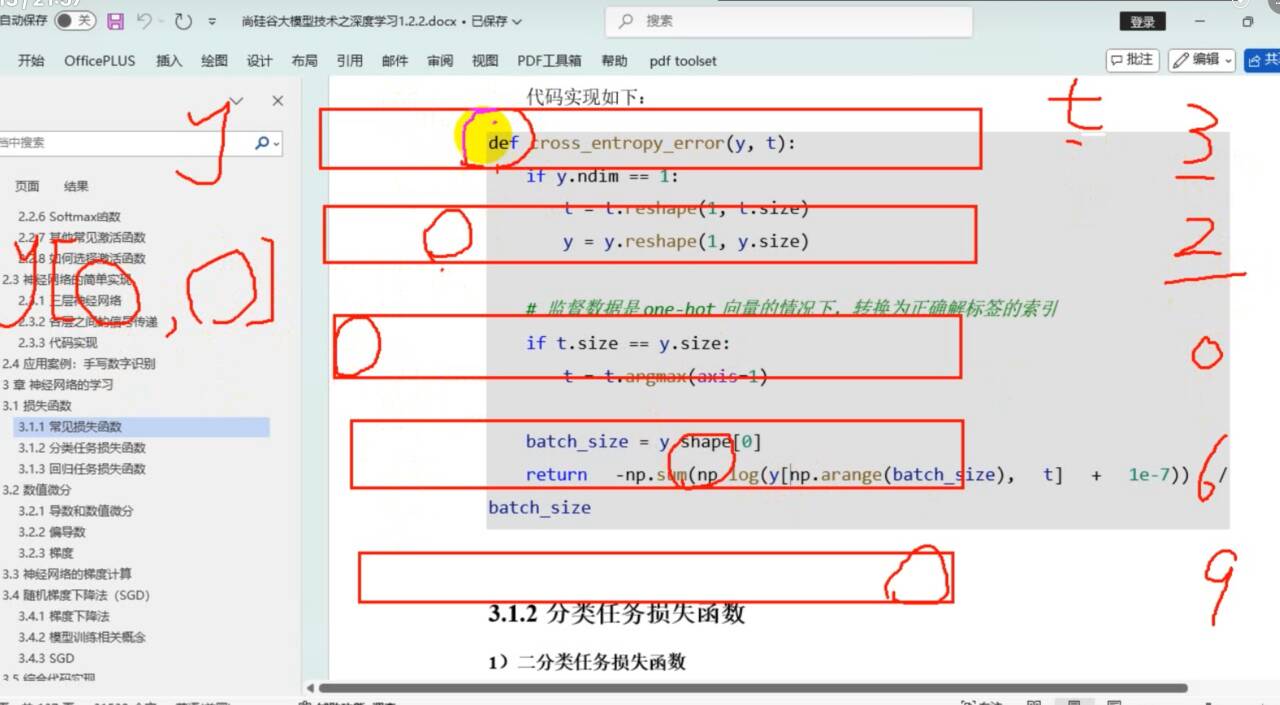
4. 对异常值敏感，遇到异常值时易发生梯度爆炸；

代码待练习；

### 交叉熵误差（CEE）

主要用于多分类问题的损失函数；也可以表示二元交叉熵的功能；

代码待练习，此代码，高维的矩阵和向量的计算比较难理解有待仔细研究 同时涉及到列表

索引；

t1就是独热编码的正确标签（多个），y1就是预测的每一类的概率（最大的概率是我们需要的）；少了表示分类的 原因是。。。

### 二元交叉熵损失函数（BCE）

正类和负类都要计算；

### 多分类任务损失函数

跟机器学习的softmax函数一致；

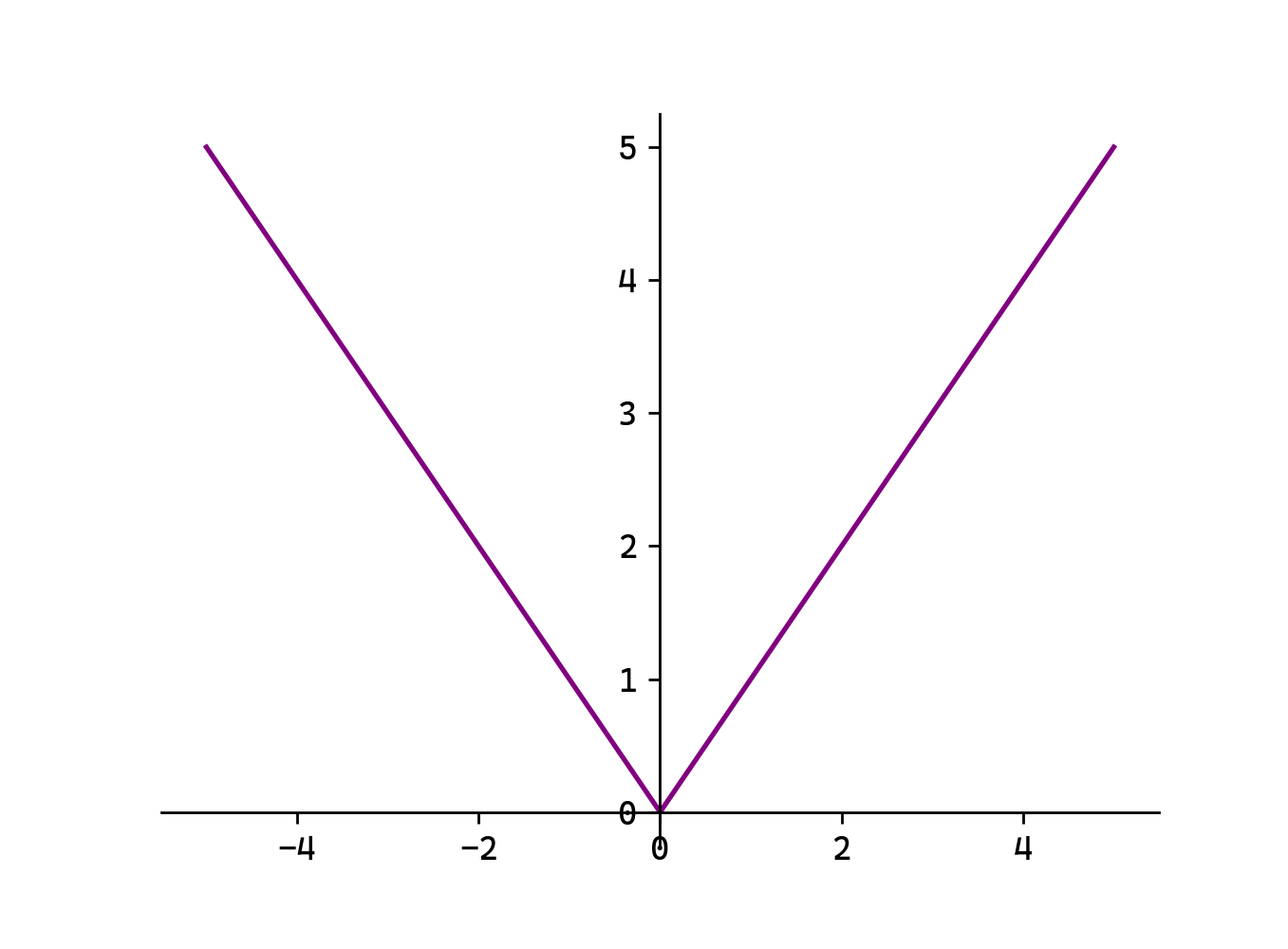
其中：

* C是类别数
* 为真实值（表示是否为类别c，通常为0或1）
* 为预测值（表示样本为类别c的概率）

### 回归任务损失函数

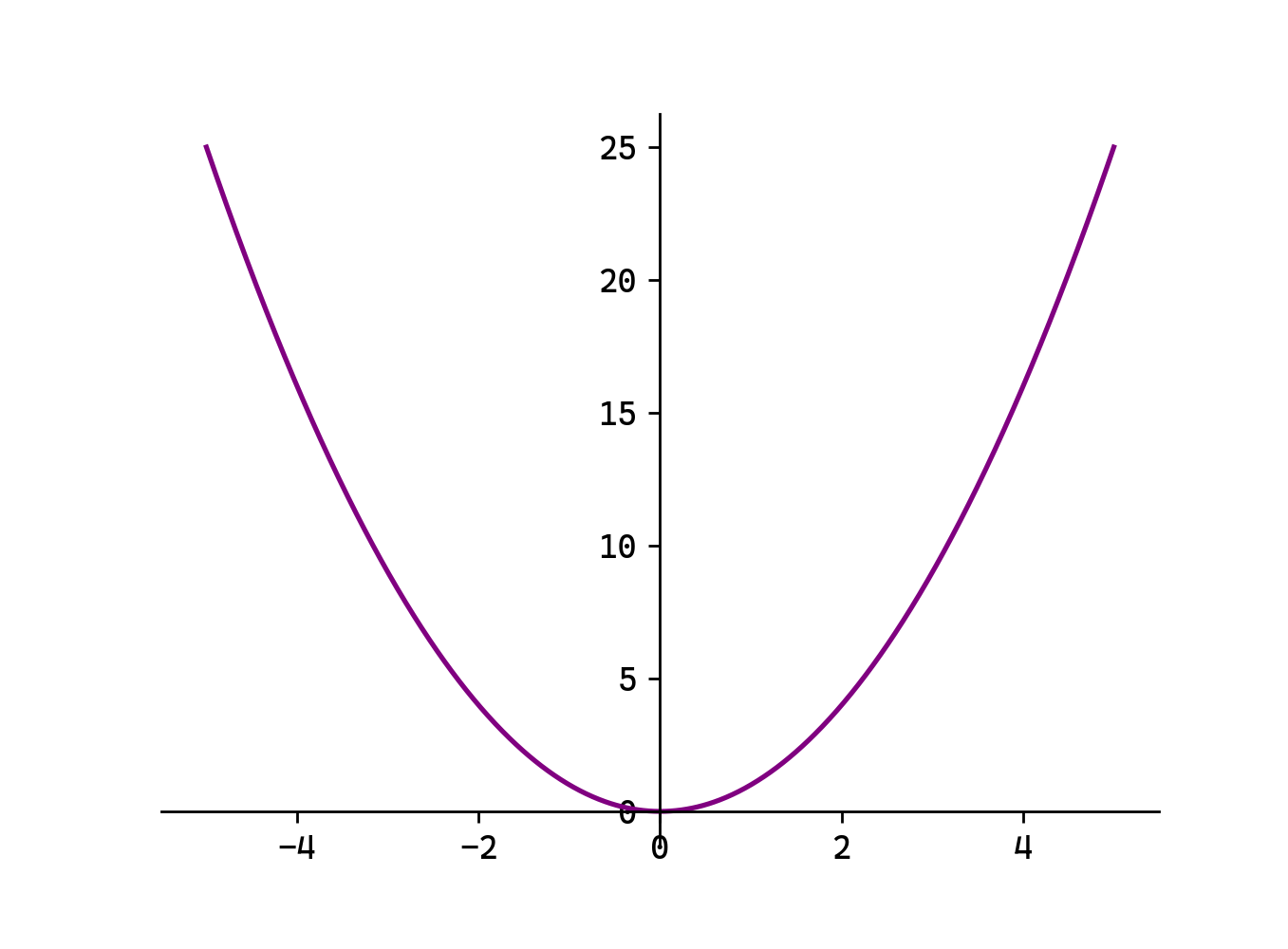
1. MAE :平均绝对误差 也成L1Loss

L1 Loss对异常值鲁棒（即异常值一比一的输出），但在0点处不可导。



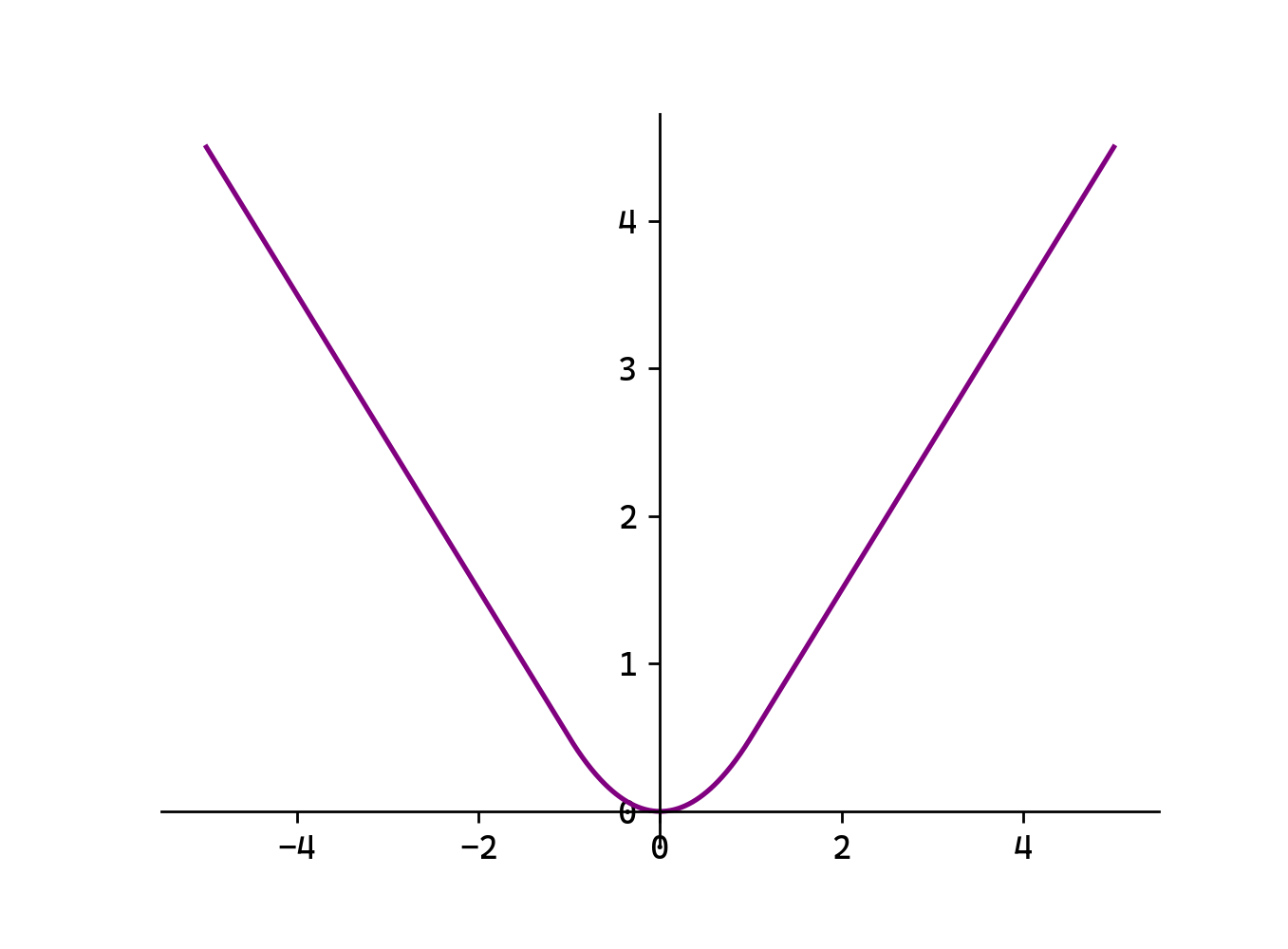
2. MSE 方差误差

会出现梯度爆炸



3. SmoothL1

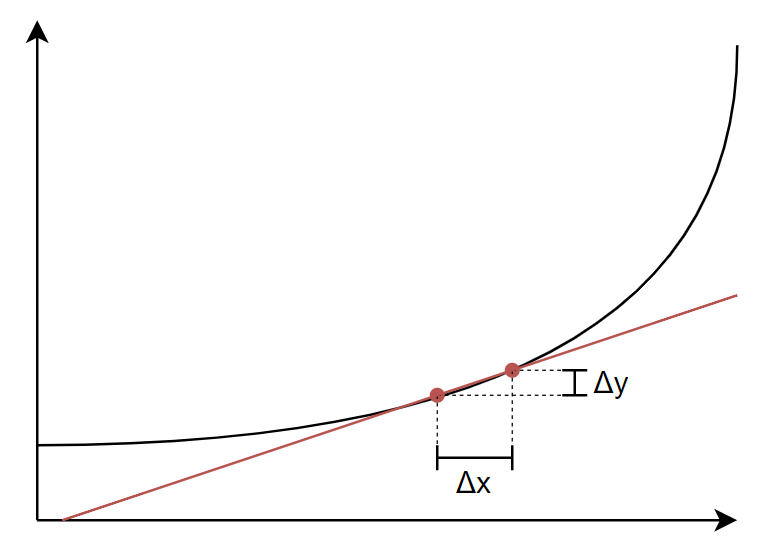
同时拥有了MAE 和 MSE的优点，即平滑处处可导，又不会出现梯度爆炸；



## 数值微分

### 导数和数值微分

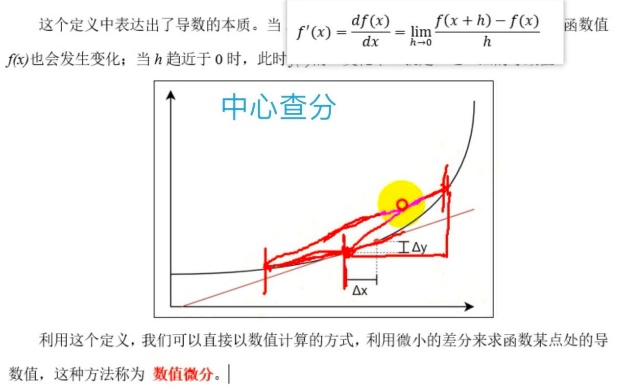
#### 导数

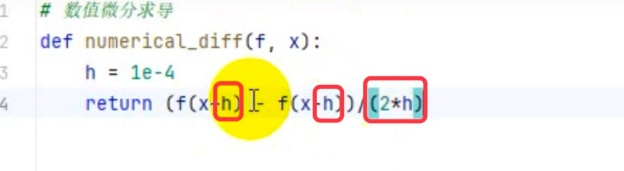


#### 数值微分

是函数求导的近似求解方法；之所以这么做是因为神经网络的前向传递函数，过于复杂（偏置和权重过多了），没有太具体的函数表达式

##### 中心差分





简单代码待操作？？

#### 偏导数

函数在某一点的针对某一个维度x的导数结果，结果是一个标量

#### 梯度

温故知新：

1. 多元函数在某一点a处，在各个维度（x1 ...xn）的多个偏导值，组成的向量，方向代表梯度下降最快的趋势，模长代表梯度下降的大小（多远函数变化率）

2. 极大值 极小值 鞍点 梯度都为零；

3. 重点选择负梯度向量

代码操作待练习？？？

#### 总结

神经网络模型的参数无法通过具体的函数表达，所以也就无法用数据解析（模型函数求导再极小值），同时也不能使用之前的梯度方法（损失函数的求导）方法，所以就只能用新的梯度方法（数值微分（中心差分））；

### 神经网络的梯度计算

梯度下降的幅度值，是用数值微分的方法计算的；