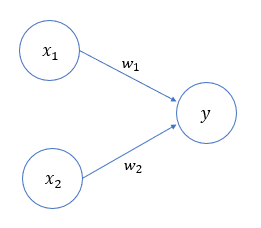
大模型技术之深度学习核心

# 2.神经网络基础

## 神经网络的的构成

### 感知机

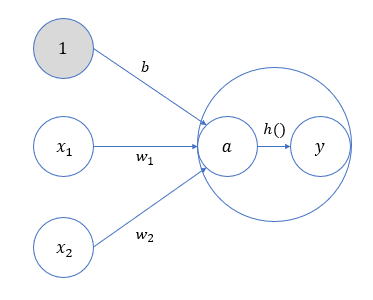
二分类模型，接受多个信号，输出一个信号，且只有0和1的取值；



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）

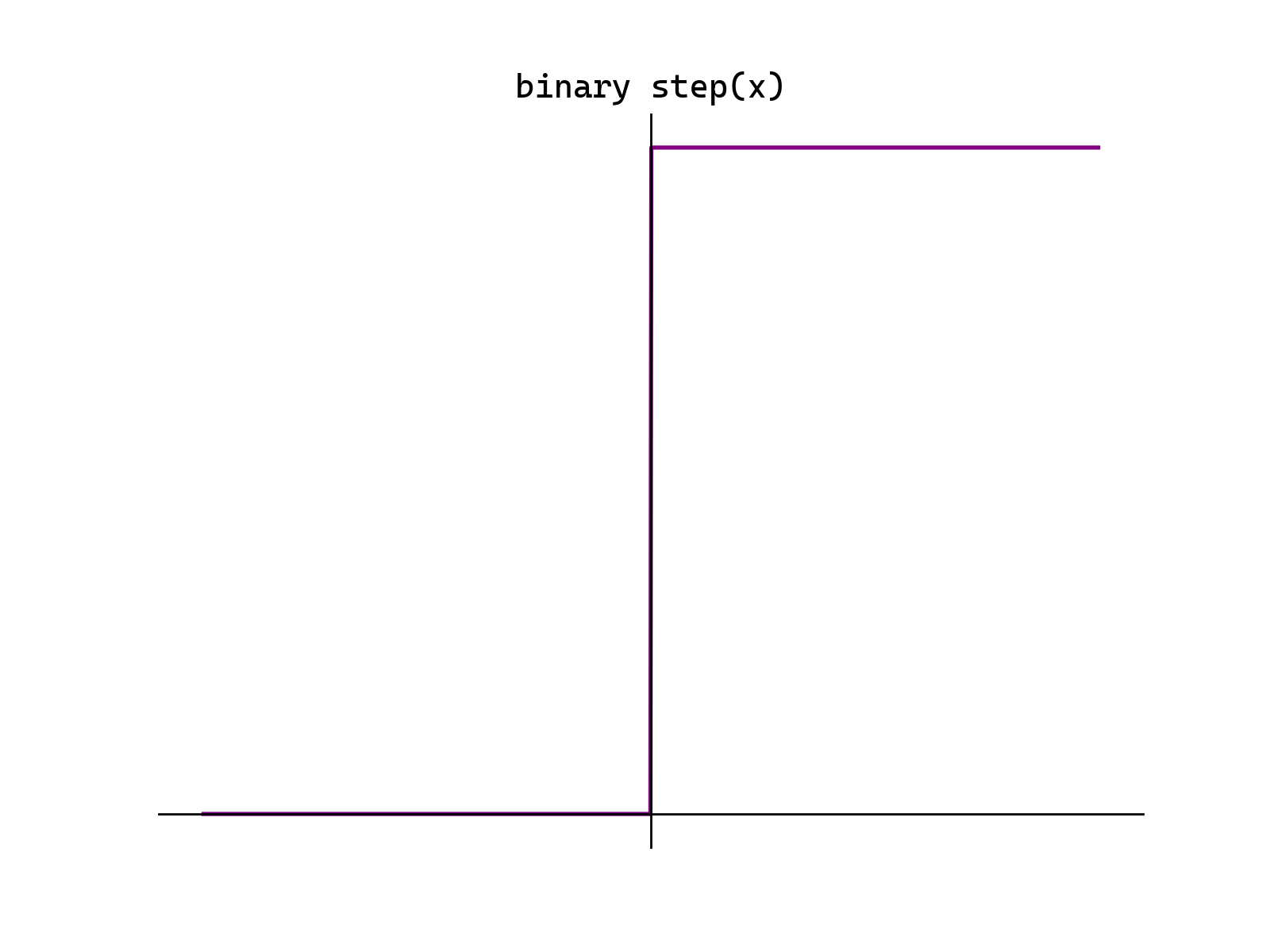
#### 激活函数的作用

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

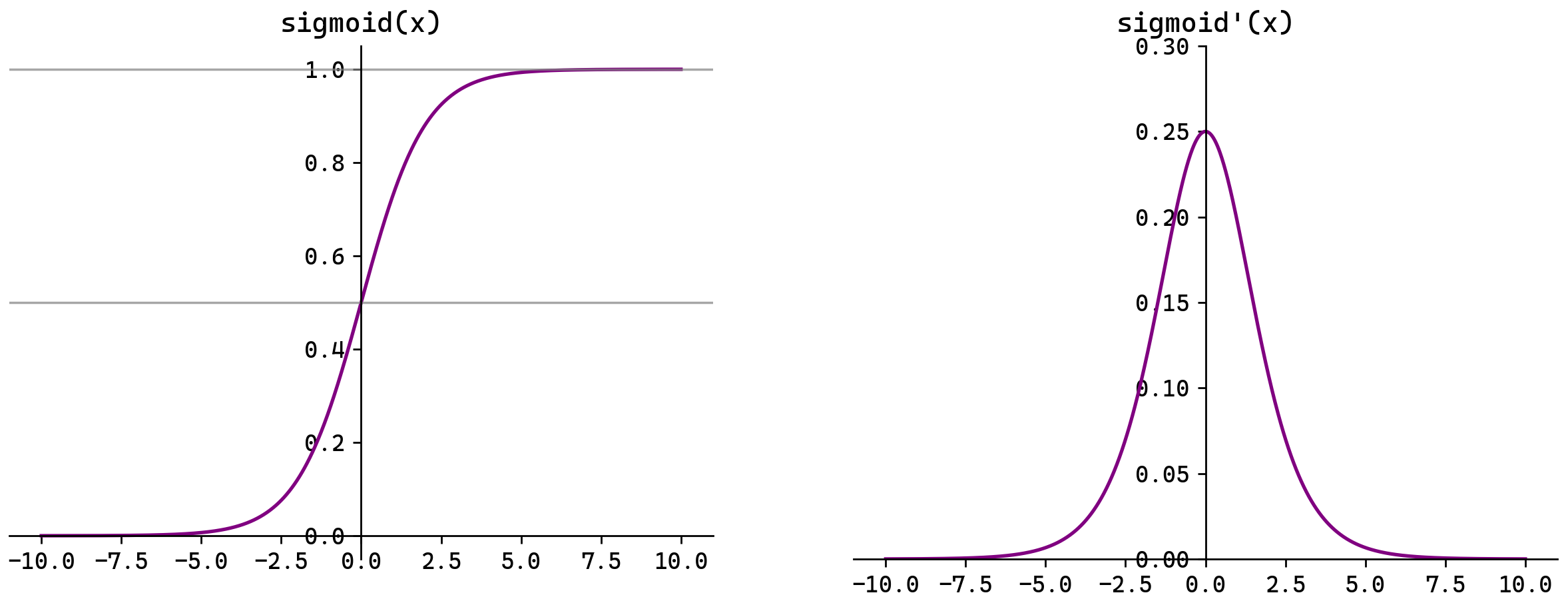
#### 阶跃函数



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



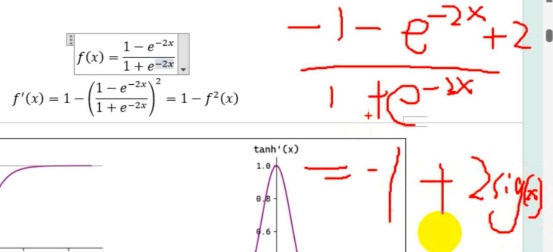
1. Sigmoid的导数范围为(0,0.25)，梯度较小。当输入在[-6,6]之外时，导数接近0，此时网络参数的更新将会极其缓慢，会出现数据丢失。使用Sigmoid作为激活函数，可能出现梯度消失（在逐层反向传播时，梯度会呈指数级衰减，），一般只作为输出层，或者只有两三层的中间层（隐藏层）；

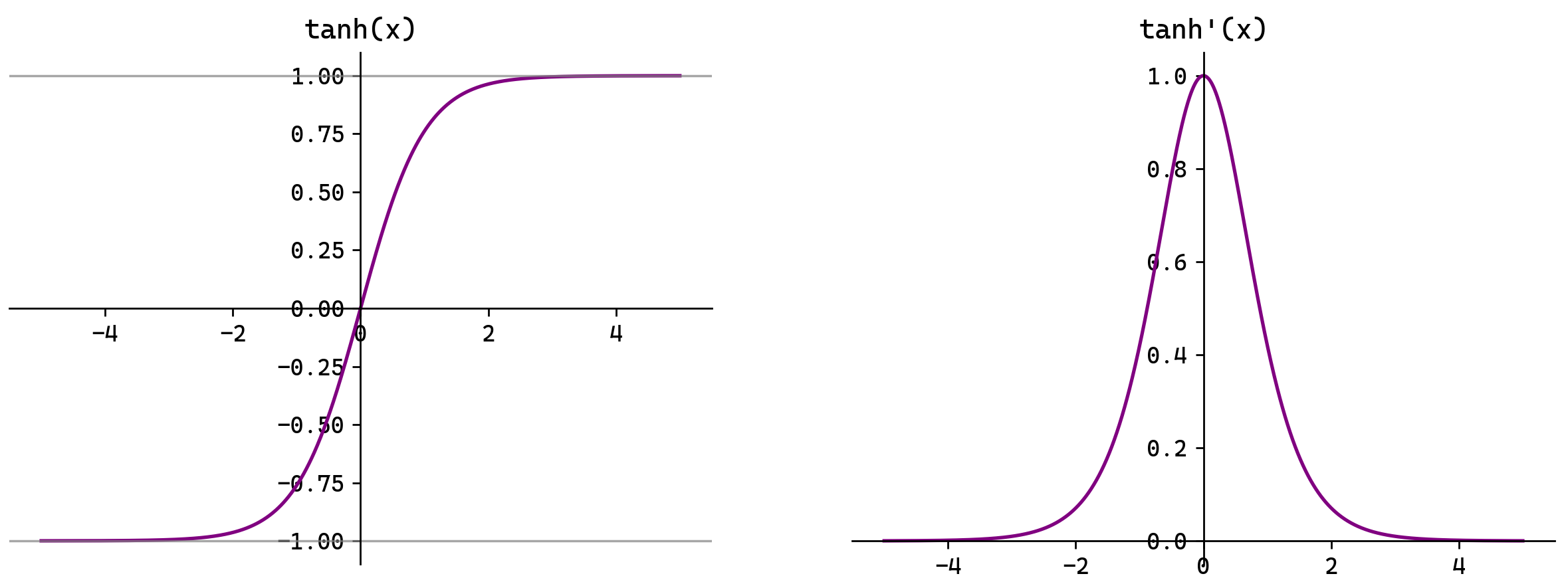
2. 并不是以圆点对称的函数；

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；

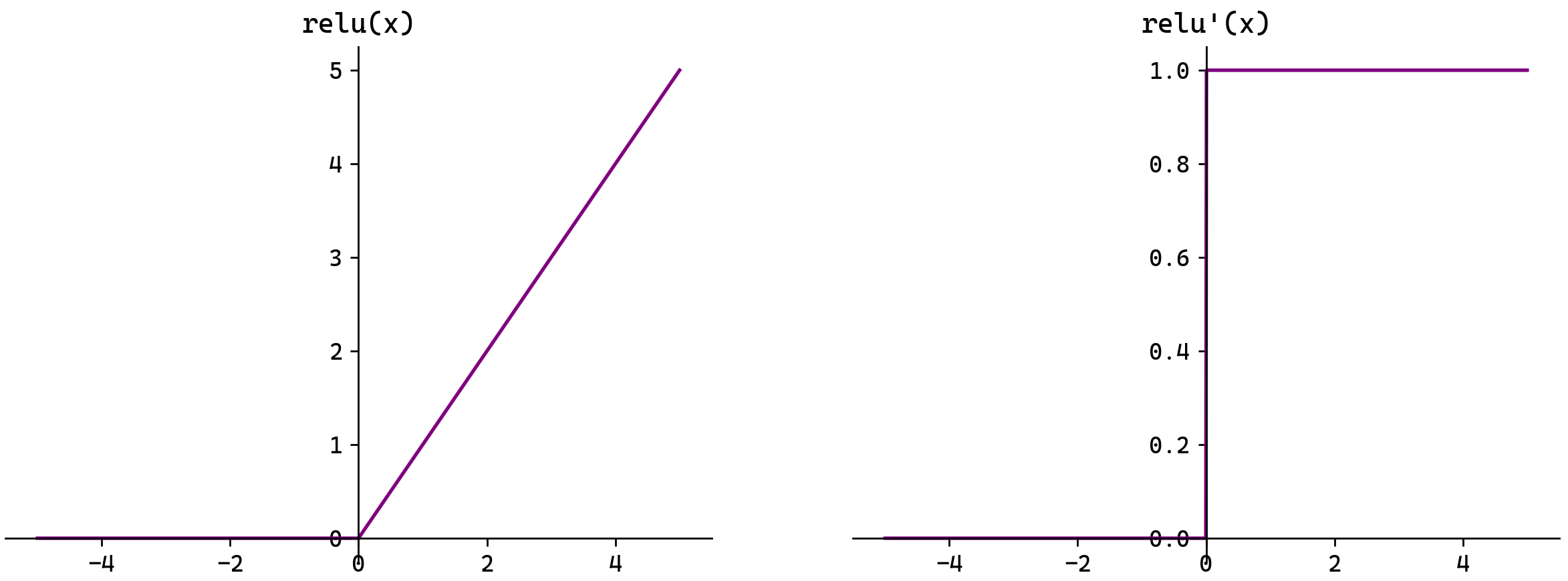
2. Sigmoid的导数范围为(0,1.0)变得更大了，但同样梯度较小

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

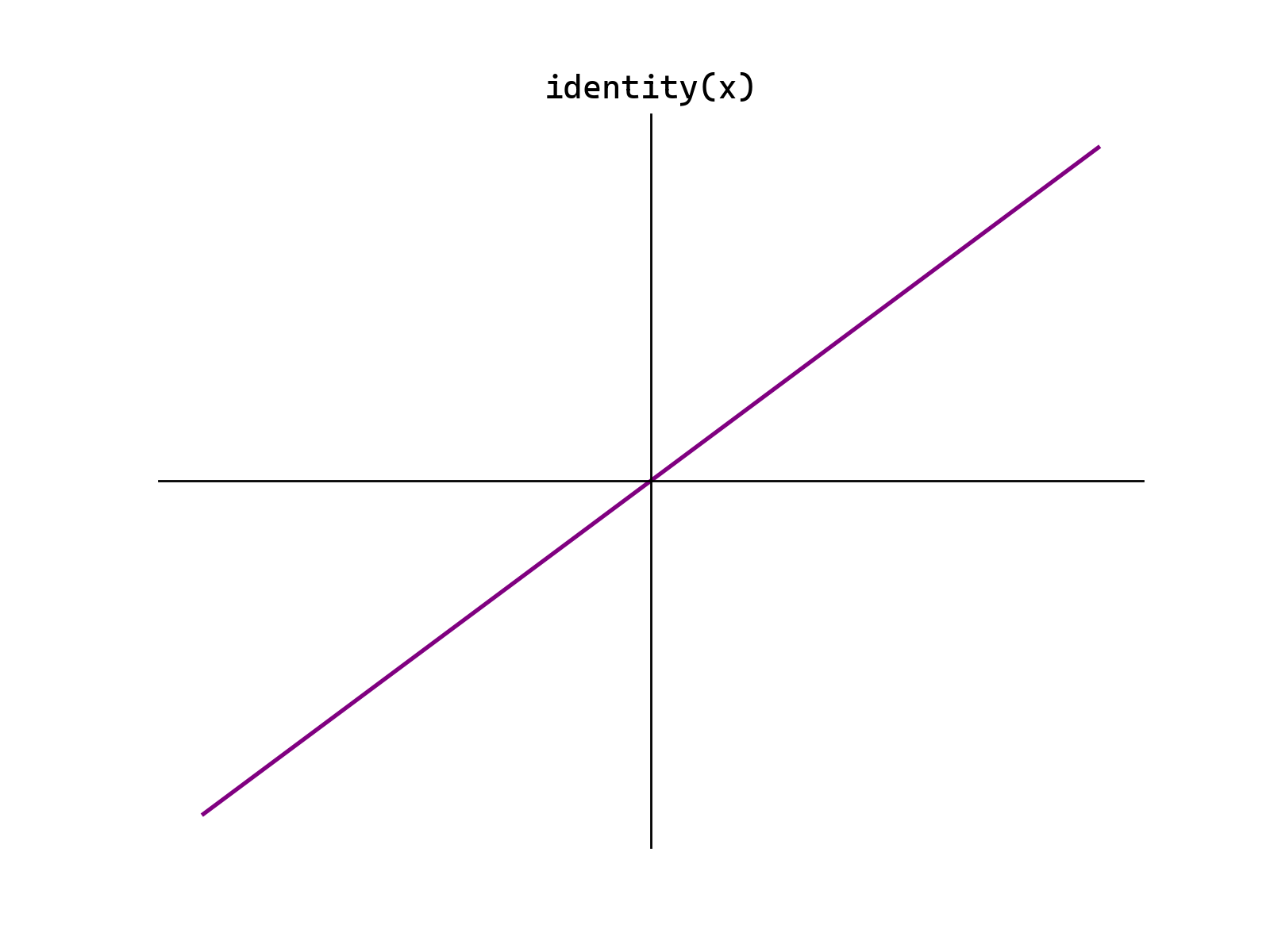
def softmax(x):  
 x = x - np.max(x) *# 溢出对策：指的是向量（分子分母）的每一个元素都减去最大的标量值，结果可能是负值，操作的同时是分母和分子，所以最终结果是不变的；* return np.exp(x) / np.sum(np.exp(x))

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

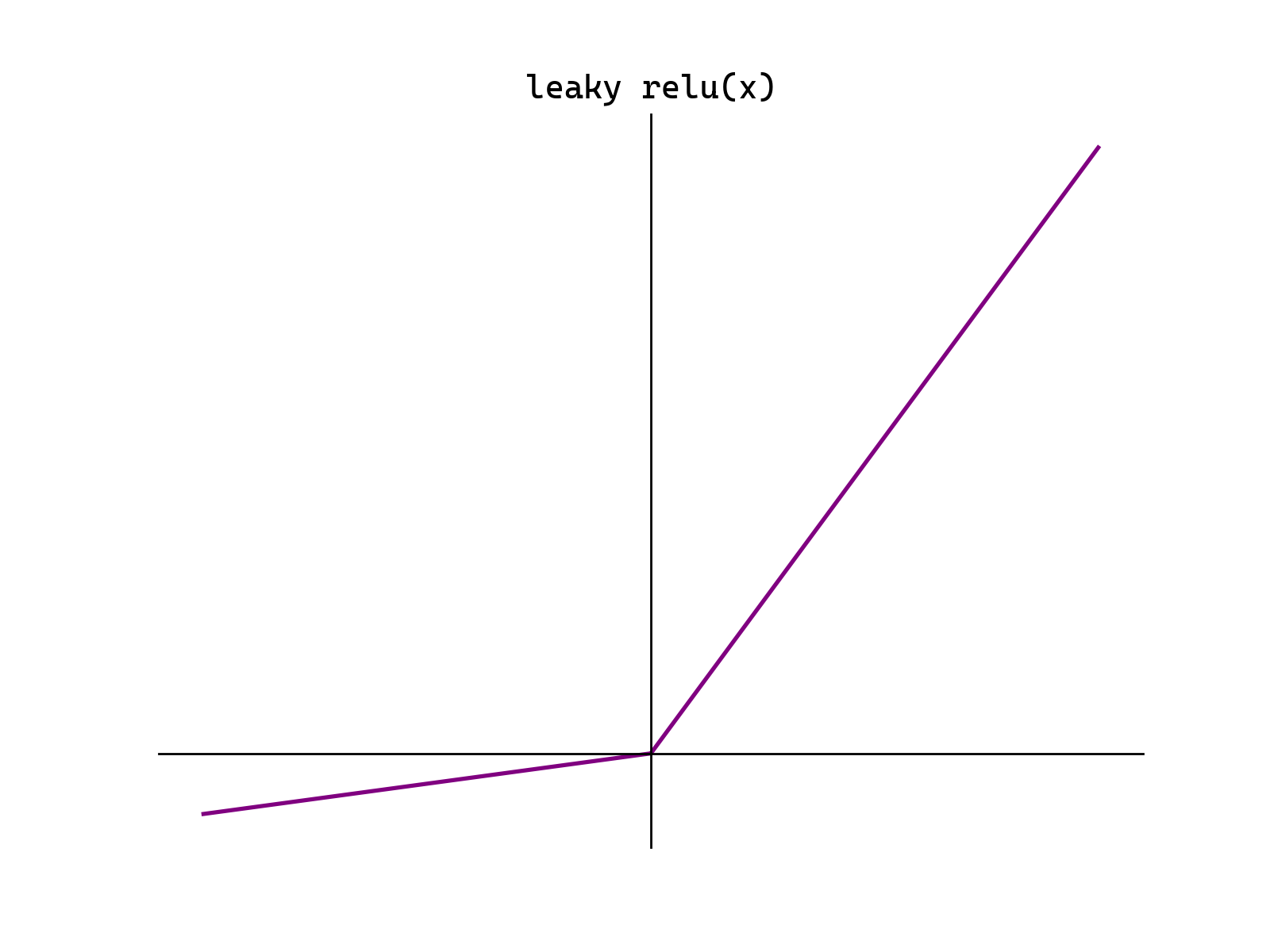
##### 恒等函数（Identity）



导数恒为1，可以作为隐藏层（中间层）；

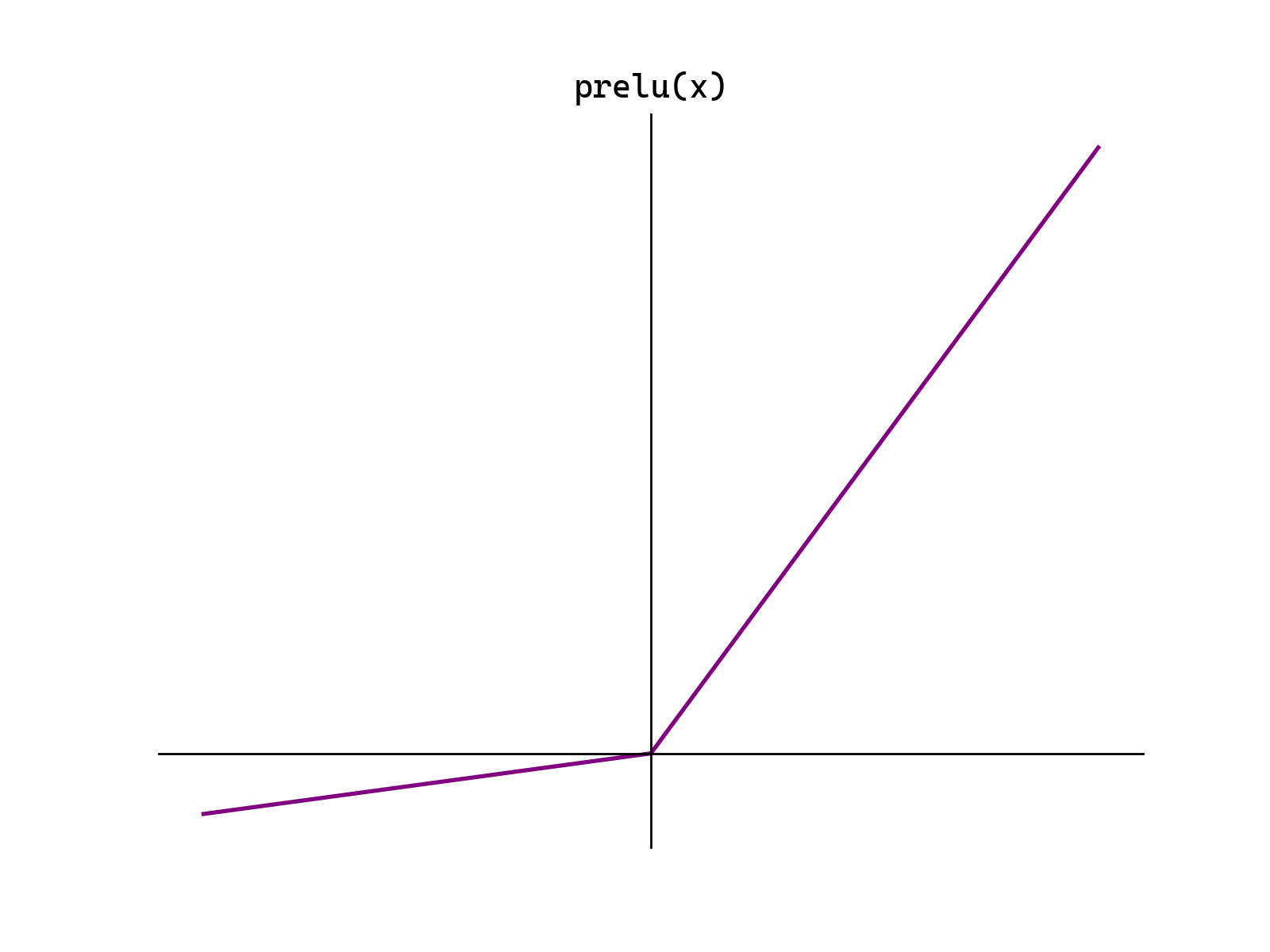
##### Leaky ReLU（Leaky Recttified Linear Unit）

ReLU函数的变体，添加了系数的固定参数



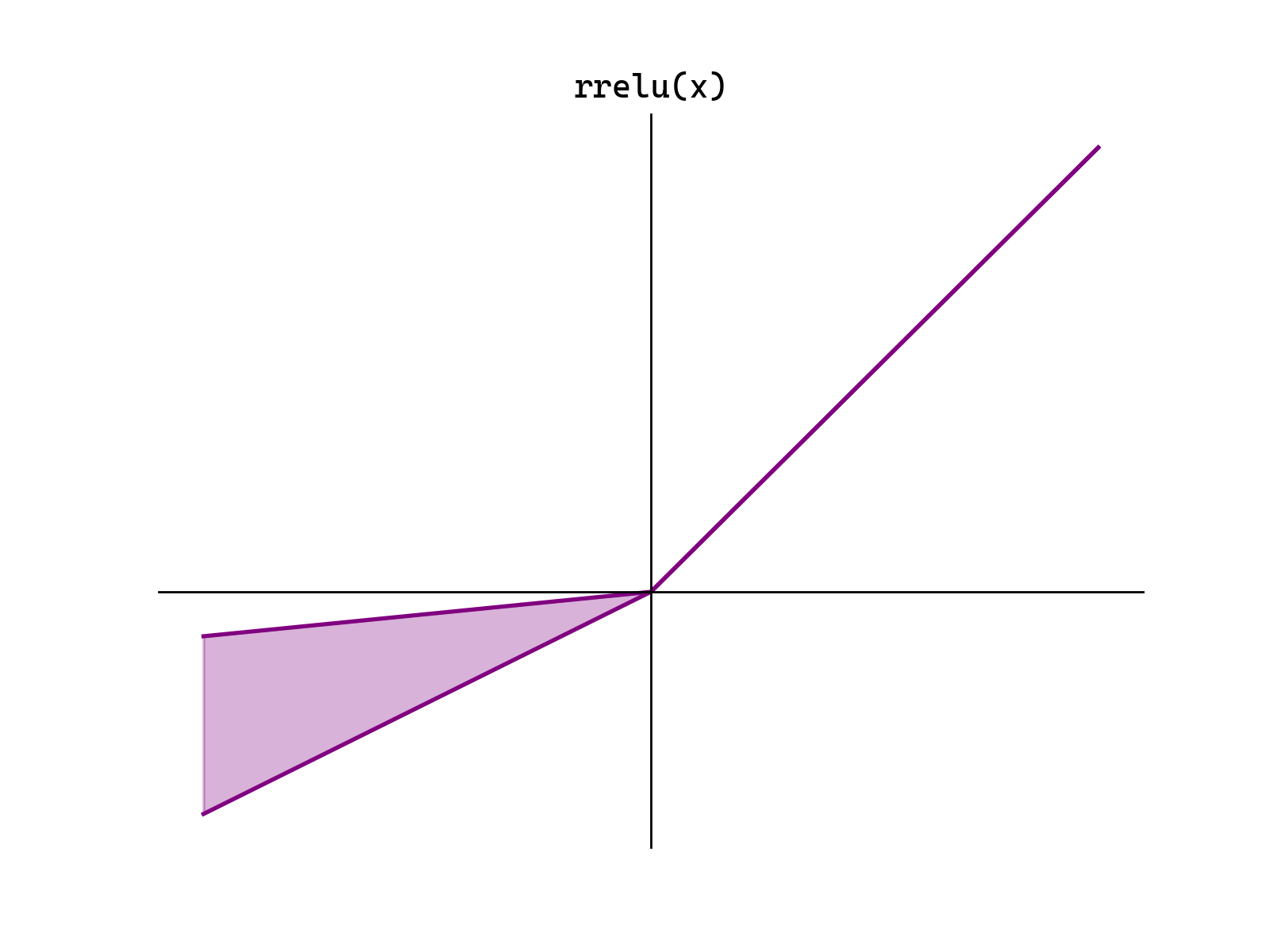
##### PreLU（Parametric Rectified Linear Unit）

这里是一个可训练的参数，而非固定的参数。



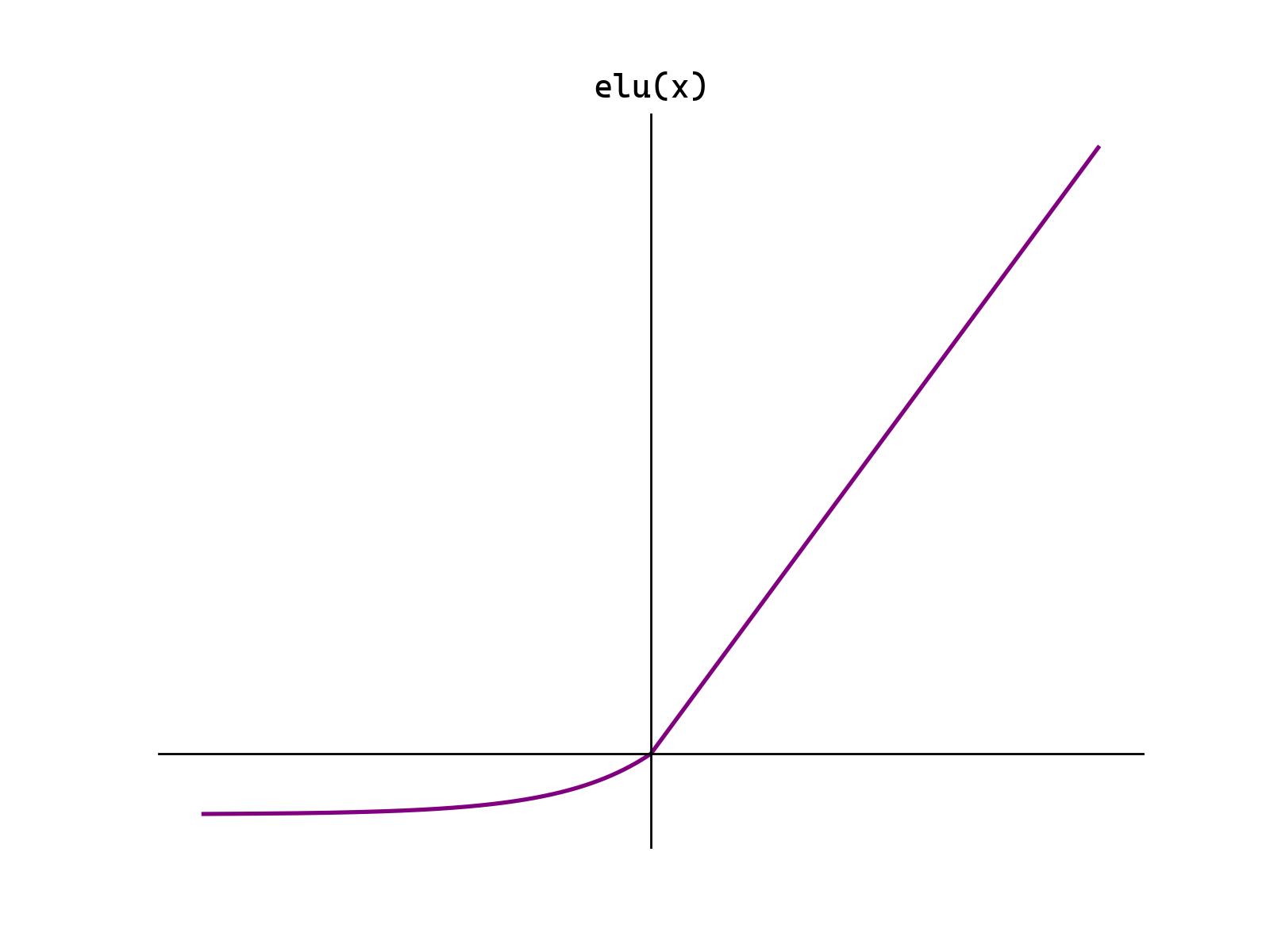
##### RReLU（Randomized Leaky ReLU）

这里是一个在训练时从一个均匀分布中随机选择的参数。



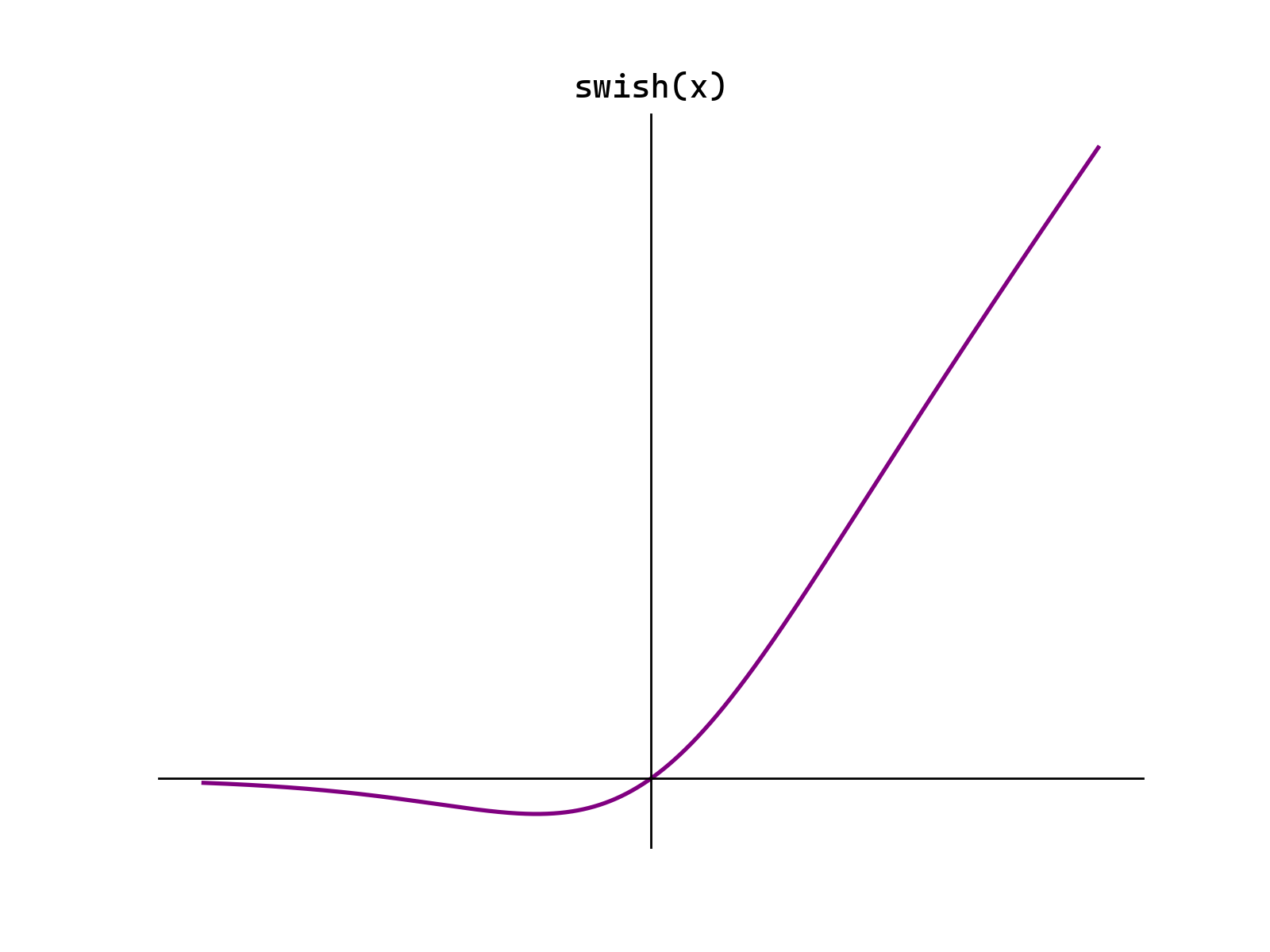
##### ELU(Exponential Linear Unit)

X小于零的函数变成了指数项



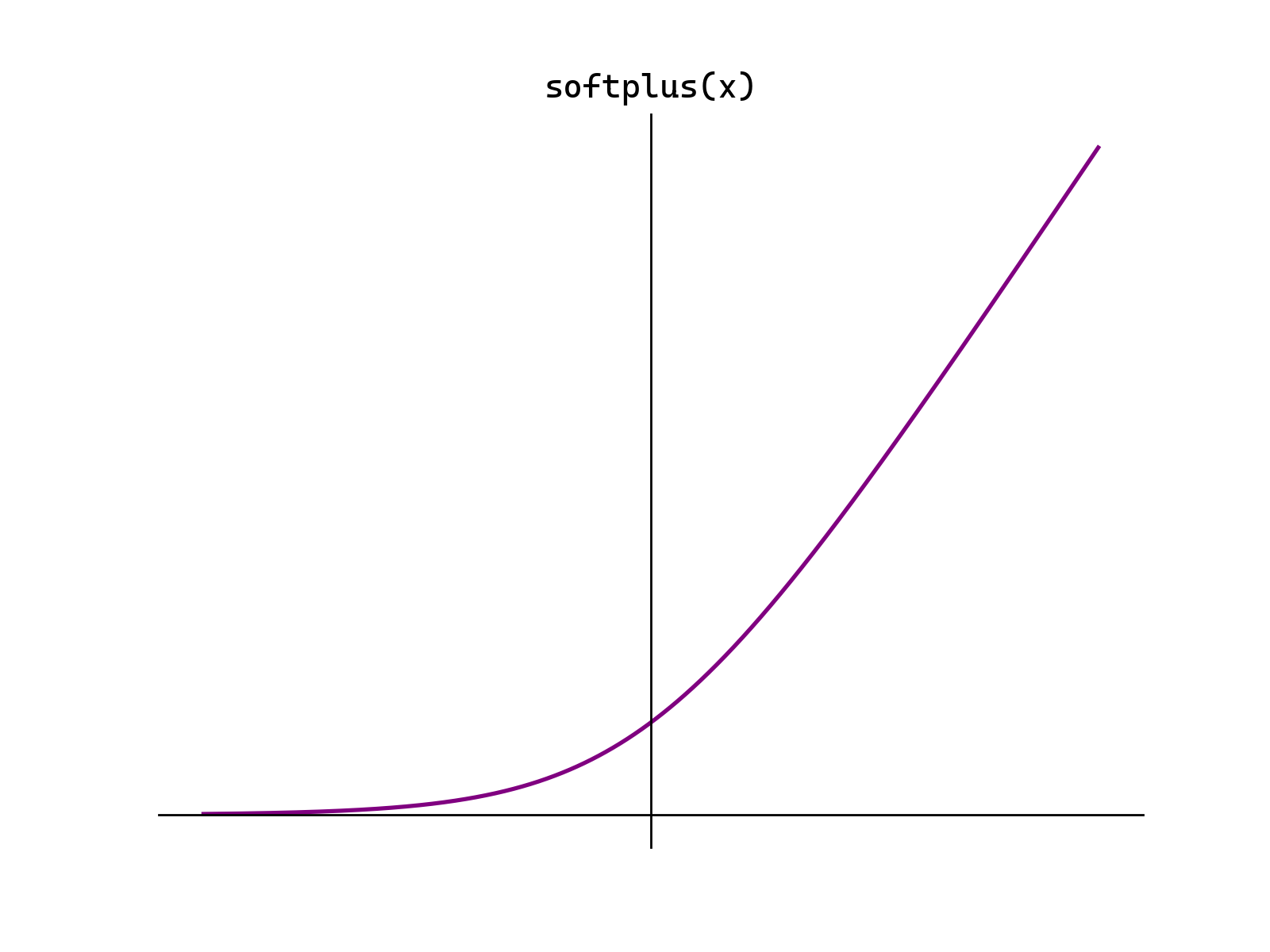
##### Swish(sigmoid Linear Unit, SiLU)

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；

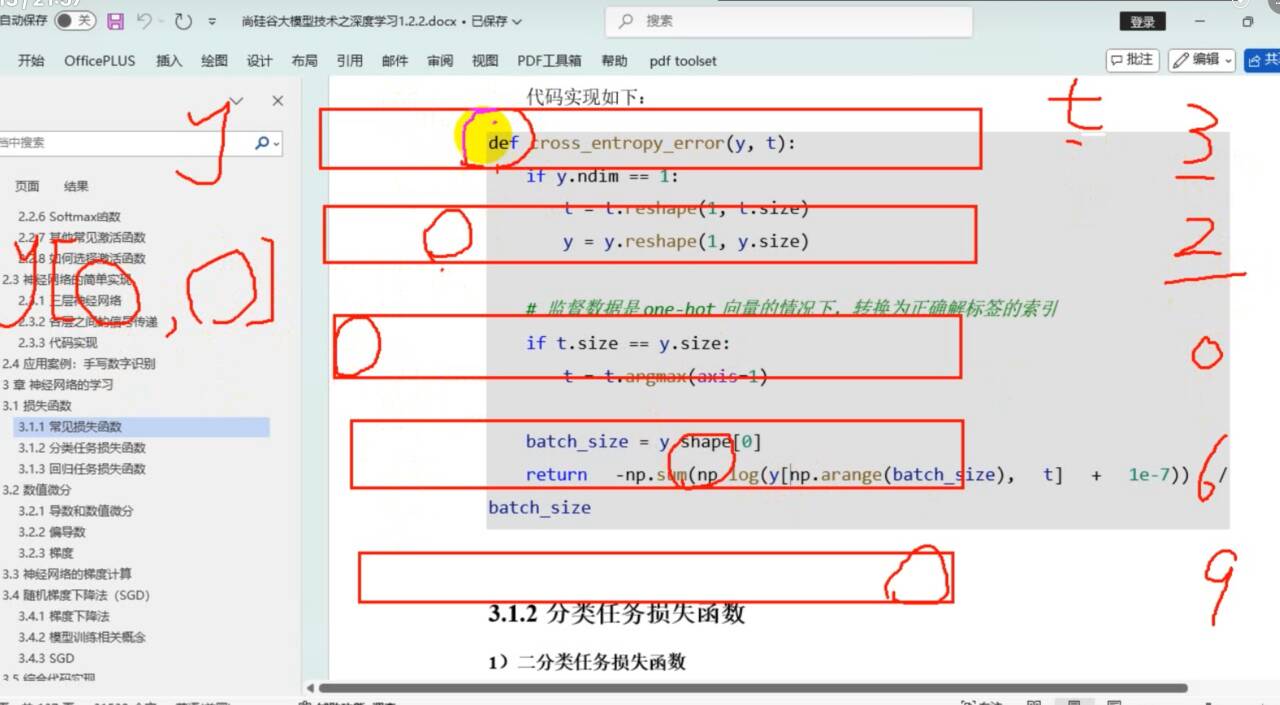
### 交叉熵误差（CEE）

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；

注释：

1. t1不能是顺序编码的原因是，如果是顺序1,2,3,4,5 ，在进行计算的时候，就会出现数值越大计算值就越大，直接影响概率值，其实1,2,3,4,5各自都是独立事件是没有关系的；

如下图：t是独热编码，图中表示的3、2、0、6、9，是独热编码的索引，也是y预测值对应的索引；

索引；

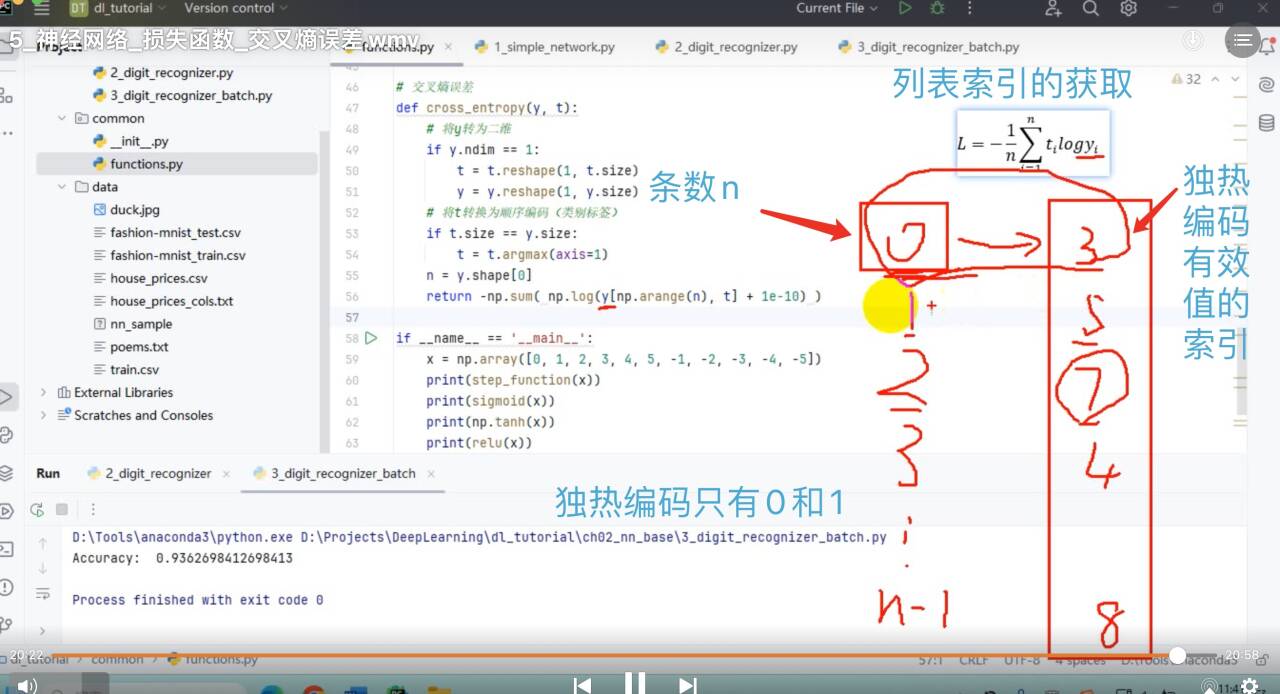
2. 公式解释

t1是独热编码的正确标签,表示为[0,0,0,1,0]，其中1的位置是正确的标签位置，

y1的值是：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 具体的可以类比下面的《多分类任务损失函数》；

3. 代码练习中： 的处理方法是，由于是独热编码eg:[0,0,1,0,0] 所以在进行乘法的时候，有效值只有1\*y，对于这种情况，我们直接取出独热编码中1对应的矩阵索引号a，然后，取出y矩阵索引号为a的值，直接进行log即可；



### 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；

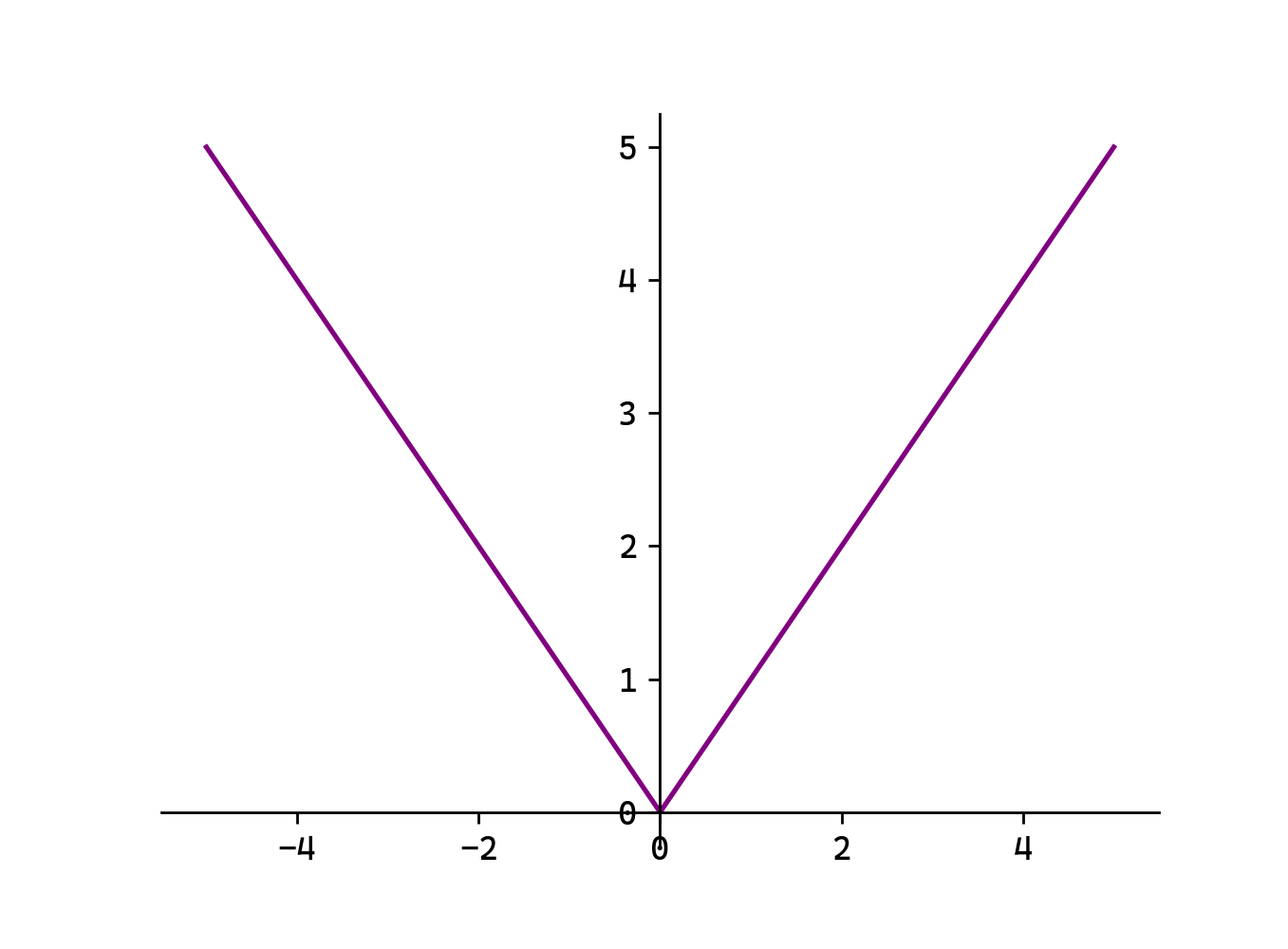
其中：

* 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

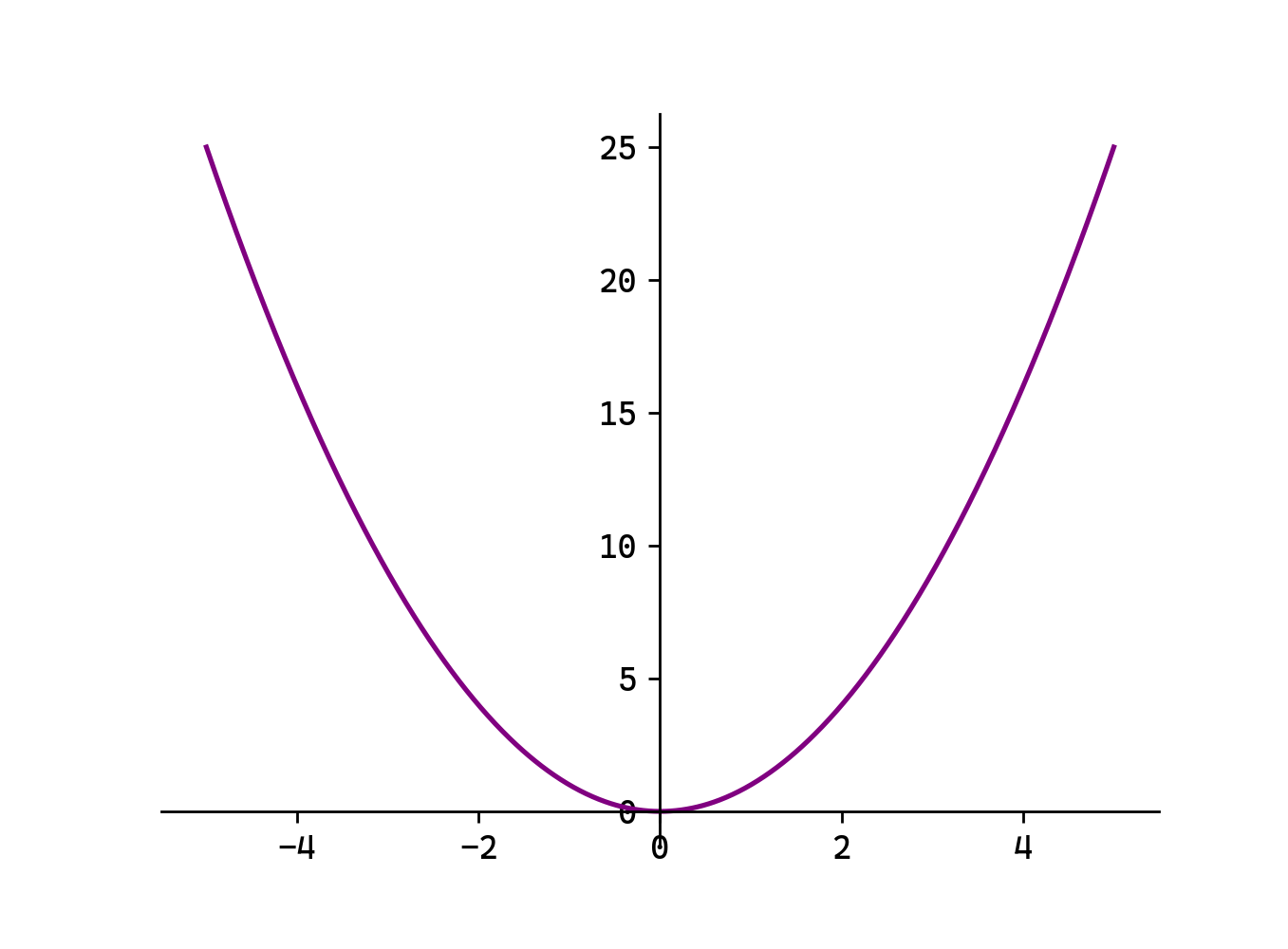
1. MAE :平均绝对误差 也成L1Loss

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。



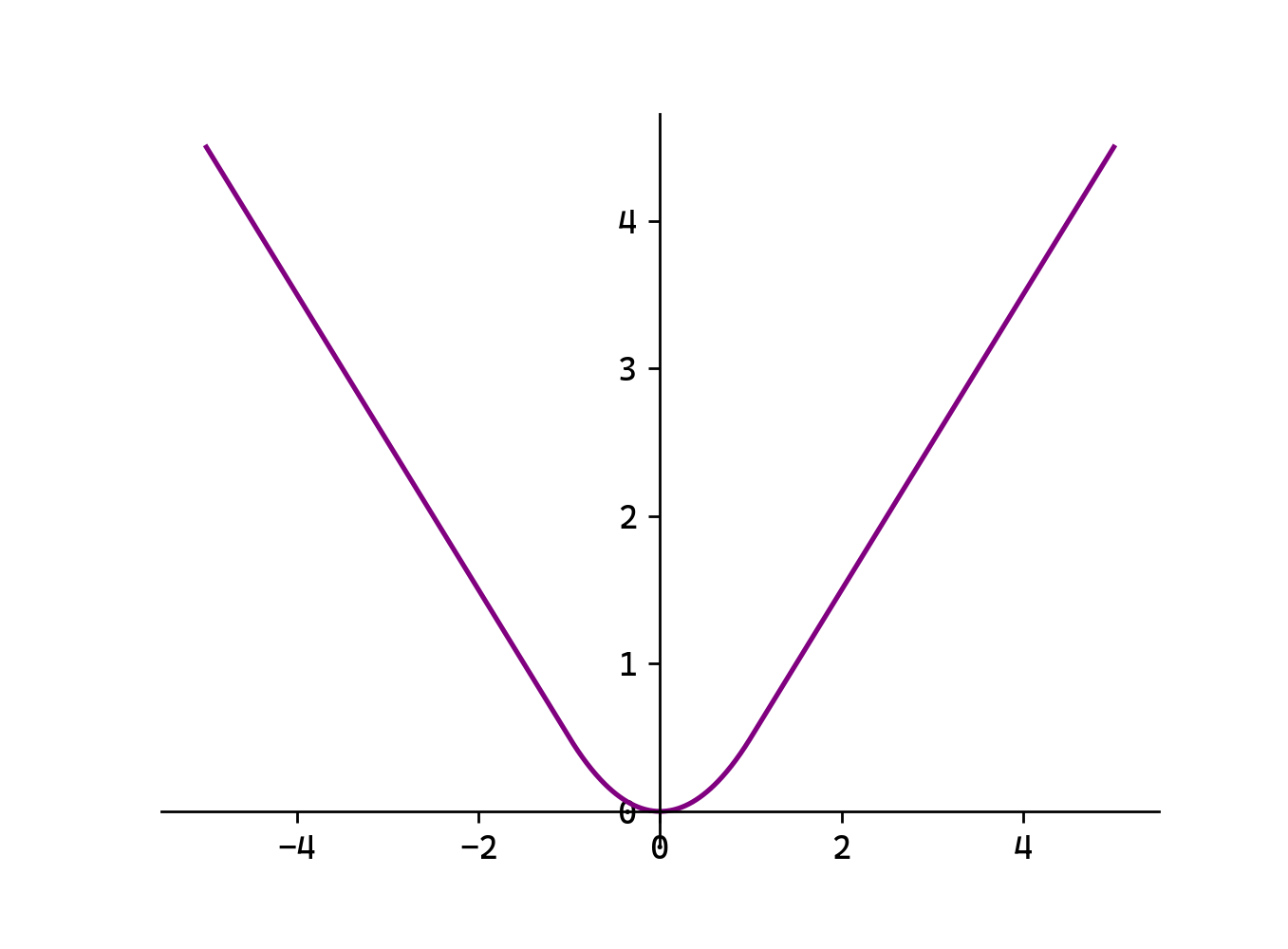
2. MSE 方差误差

会出现梯度爆炸



3. SmoothL1

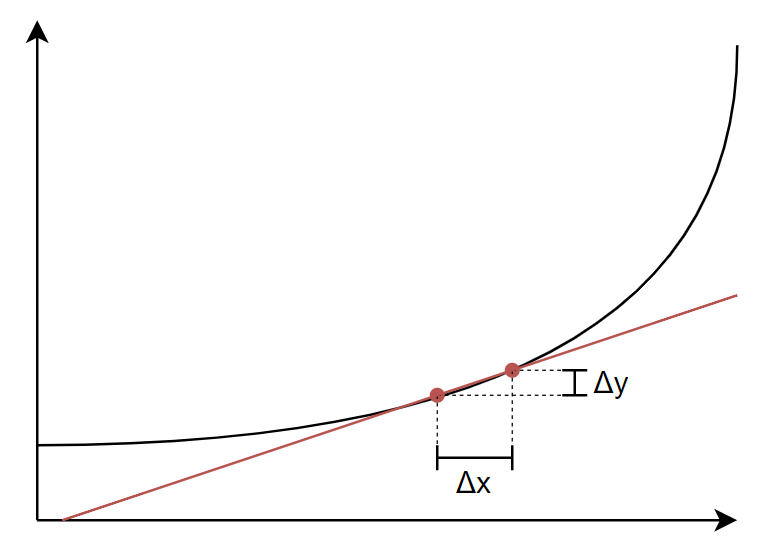
同时拥有了MAE 和 MSE的优点，即平滑处处可导，又不会出现梯度爆炸；



## 数值微分

### 导数和数值微分

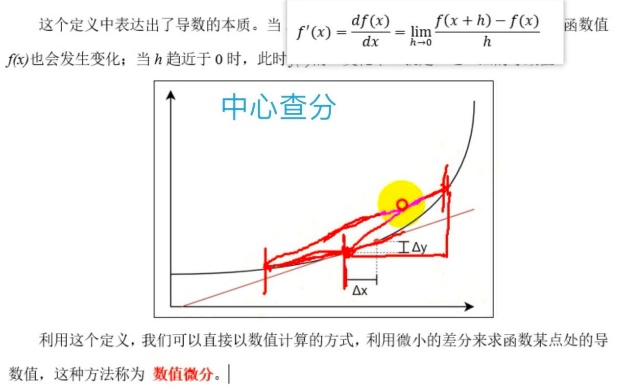
#### 导数

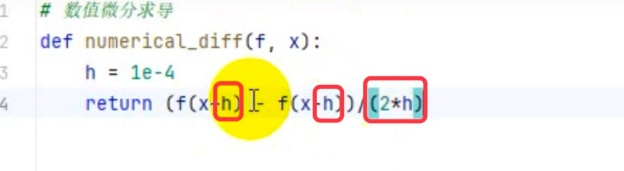


#### 数值微分

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；

### 随机梯度下降法）（SGD）

#### 梯度下降

回顾基本概念

#### 模型训练相关概念

##### Epoch

表示模型完成遍历一次整个训练数据集的过程，即所有样本数据量n

##### Batch Size

每次训练是输入的样本数量，选取数据的时候，可以是随机样本（随机选取单个样本训练BatchSize=n），小批量样本（即选一个一个的BatchSize执行）、全批量样本（BatchSize=n）；

##### Iteration

一次Itreation表示完成一个Btahc数据的正向传播或者反向传播；

##### 总结

Epoch = Batch Size \* Iteration

##### 随机梯度下降法（SGD）

1. SGD表示随机梯度梯度下降法（机器学习中的概念），在神经网络的学习中不太一样了，它表示: 先随机选择批量数据 🡪 计算梯度 🡪 更新模型参数（模型参数作为自变量） 🡪 再随机选择批量数据 🡪 …循环前面的步骤… 🡪 直到达到给定的超参数（迭代次数Iteration 和 损失函数的极限值） 🡪 最终得到了最优的模型参数；

所以在神经网路中的SGD其实就是随机梯度+小批量梯度的结合操作；

2. 有放回的随机梯度下降法，可以增加模型的泛化能力；无放回的随机梯度下降法，会减弱模型的泛化能力；

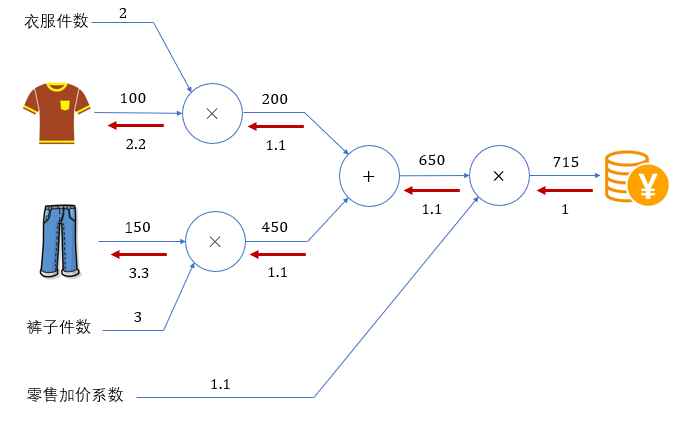
3. 工程上的是应用：多个Epoch,每个Epoch可以提前随机划分多个Iteration的BatchSize，然后在进行梯度下降计算，此操作也就是无放回的操作，但是多个Epoch的操作中，每次随机划分，也就弥补了随机性的概率，即提高了模型的泛化能力；

### 总和代码待操作

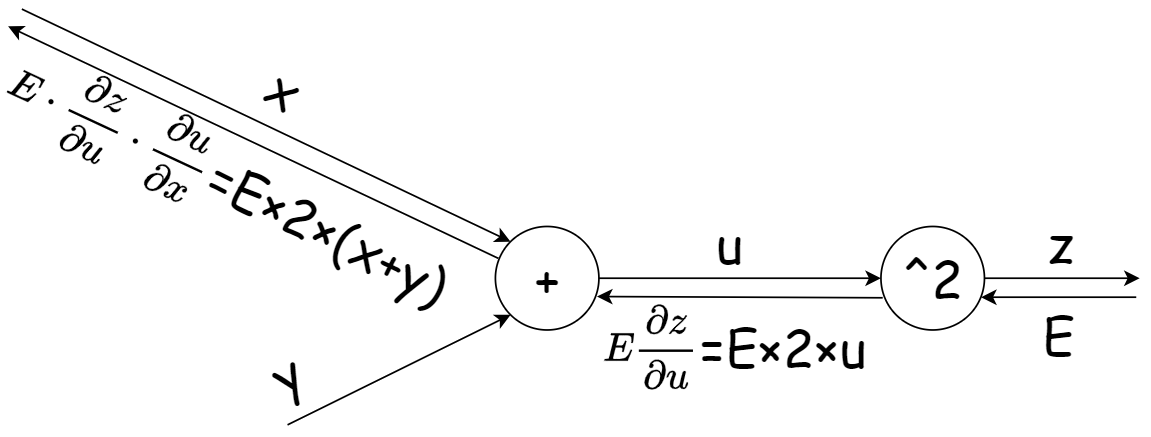
# 4. 反向传播算法

## 基本概念

根据微积分中的链式法则，按相反的顺序从输出层到输入层遍历网络。该算法存储了计算某些参数梯度时所需的任何中间变量；



## 链式法则



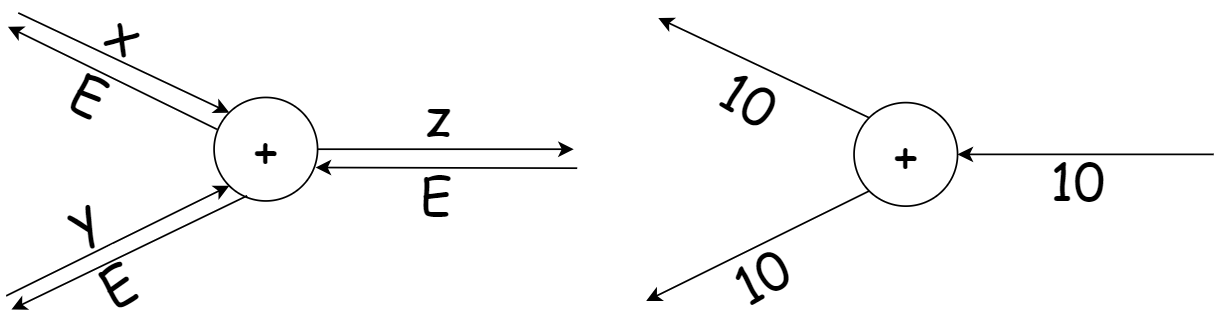
总结：

1. 前向传播的函数其实就是复合函数，那么后向传播的求导，也就是复合函数的求导（链式求导），如下

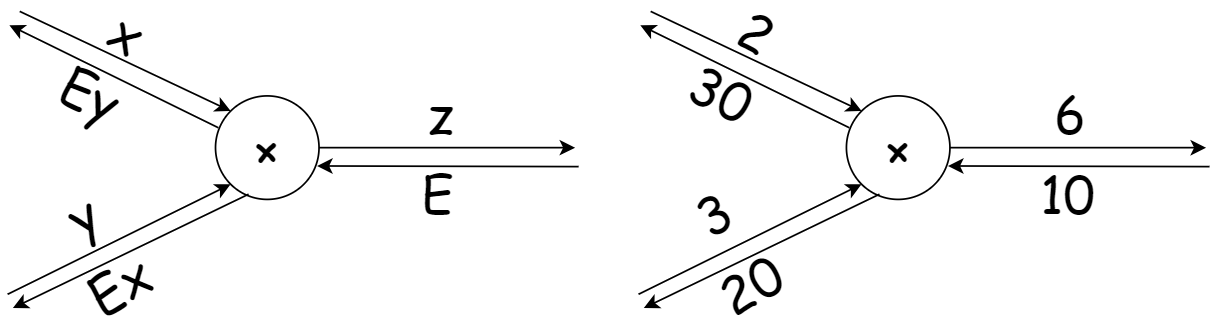
对于复合函数，令，则

## 反向传播

### 加法节点的反向传播



### 乘法节点的反向传播



总结：

1. 在反向传播之前需要先进行一次前向传播，来记录各个节点的x 和 y （这里是x y是传递给神经元的信息）；

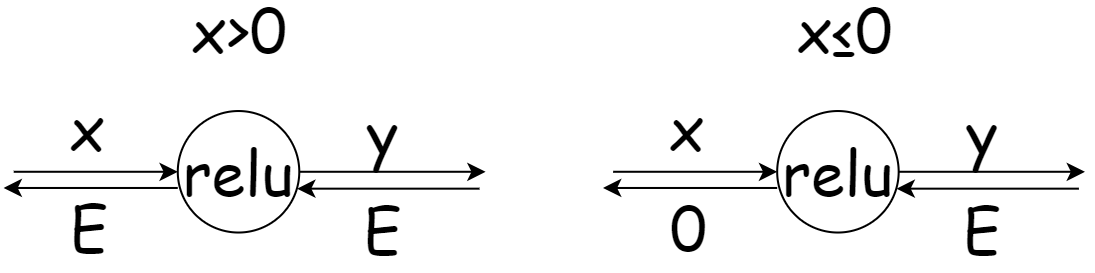
2. 反向传递本质就是复合函数的链式求导；

### 激活层的反向传播的实现

#### ReLu的反向传播

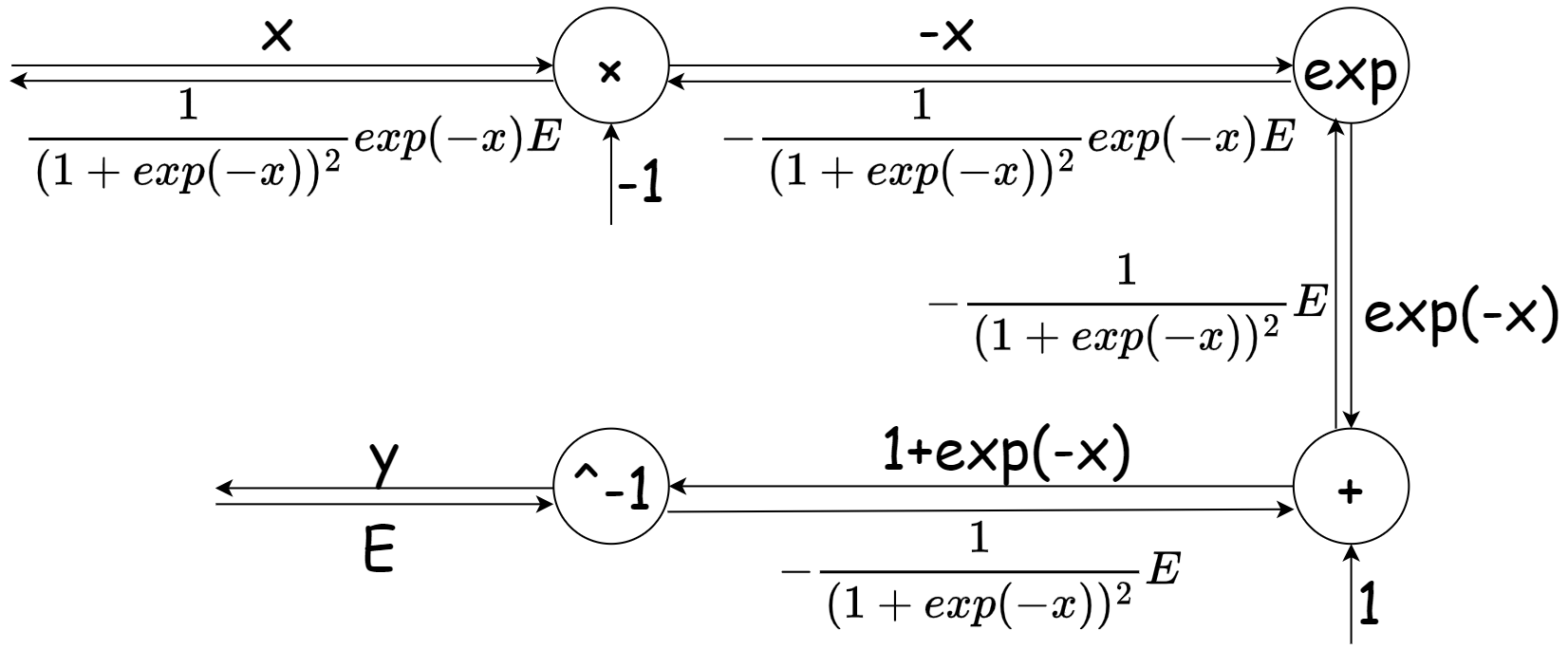
对于ReLU函数：

其导数为：



代码实现待操作

#### Sigmoid的反向传播



代码实现待操作

### Affine的反向传播和实现

### 输出层的反向传播和实现

### 反向传播的综合实现

# 5. 学习技巧

## 深度神经网络及其问题

### 深度学习

使用更多的层来分层次传递信息，有效的减少网络参数数量，从而是的学习更加高效；

2012年开始的ILSVRC学术竞赛是机器视觉领域最受追捧的竞赛之一；

### 梯度消失和梯度爆炸

当反向传播进行很多层的时候，每一层都对前一层梯度乘以了一个系数；因此当这个系数比较小（小于1）时，越往前传递，梯度就会越小、训练越慢，导致梯度消失；而如果这个系数比较大，则越往前传递梯度就会越大，导致梯度爆炸；

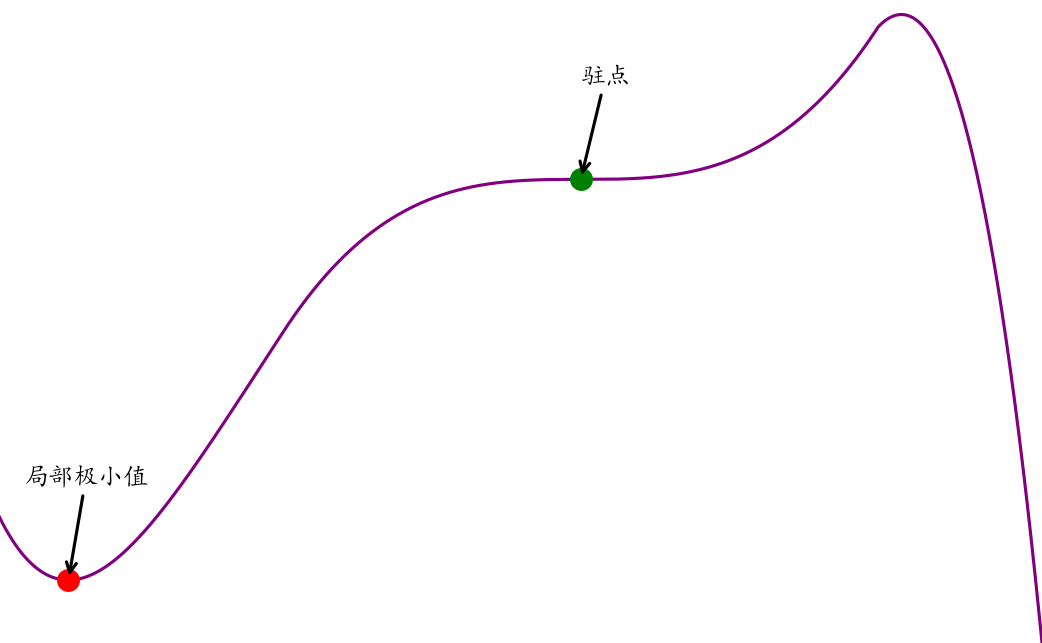
改变方法：网络架构（神经元个数或者层级）、权重参数的初始化、超参数的初始化、激活函数的选择、梯度更新算法的调整；

### 更新参数方法的优化

#### SGD(梯度下降法)：

优点：经典简单

缺点：不是很高效、陷入局部最优、鞍点问题、学习率过大导致震荡不收敛、收敛速度慢（高维或者非凸函数）；



#### Momentum(动量法)

动量法会保存历史梯度并赋予一定的权重，使其也参与到参数更新中

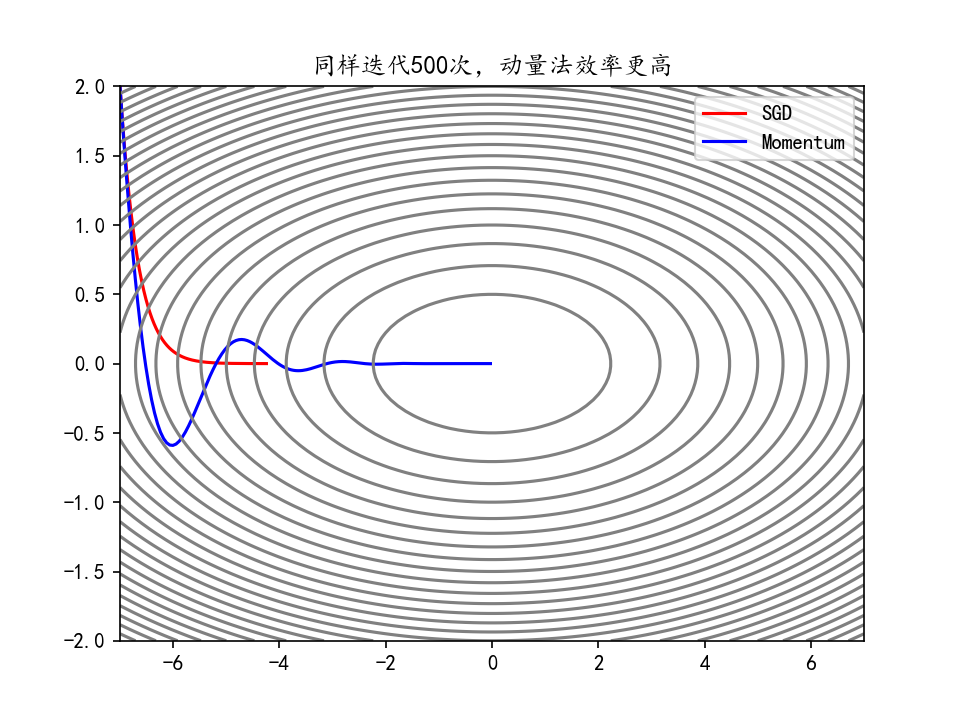
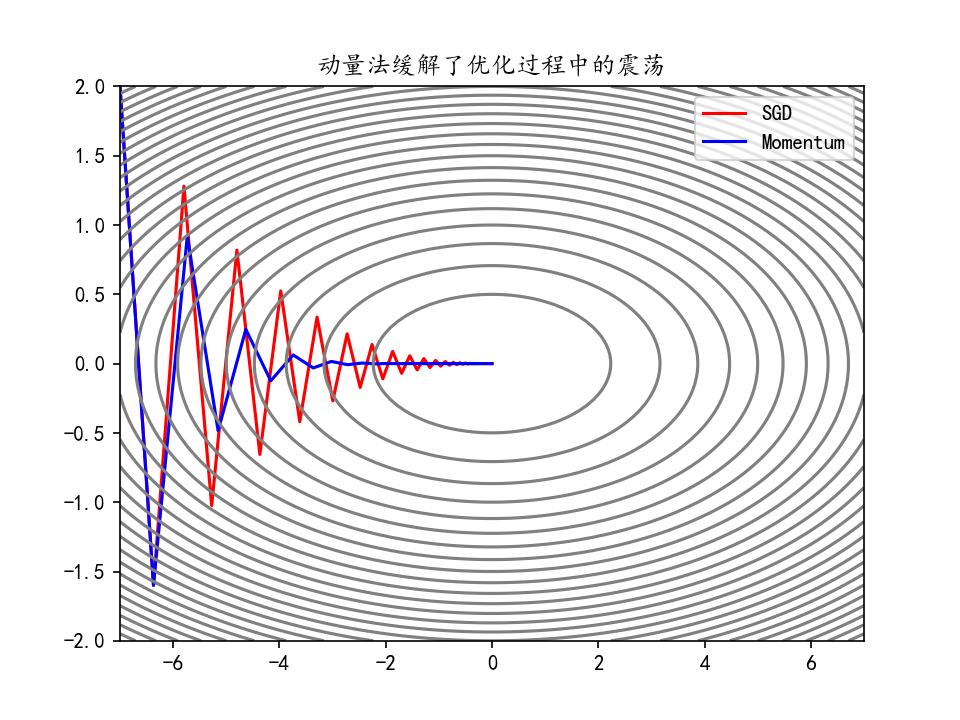
* ：历史（负）梯度的加权和 （初始值为0）
* ：历史梯度的权重
* ：当前梯度，即
* ：学习率

注：

如果历史的梯度有正有负，则可以起到抑制作用可以快速收敛；

如果都为正数，则也会加快收敛，同时在遇到鞍点或者局部最优的时候，可以再往前冲一冲；具体如下图，在x轴速度会更快，在y轴则起到振幅抑制的作用；

历史加权中梯度会被逐渐弱化（历史梯度权重（小于1）的n次方，确实很精妙）；



#### 学习率衰减

等间隔衰减、指定间隔衰减、指数衰减

注：过于固定和死板

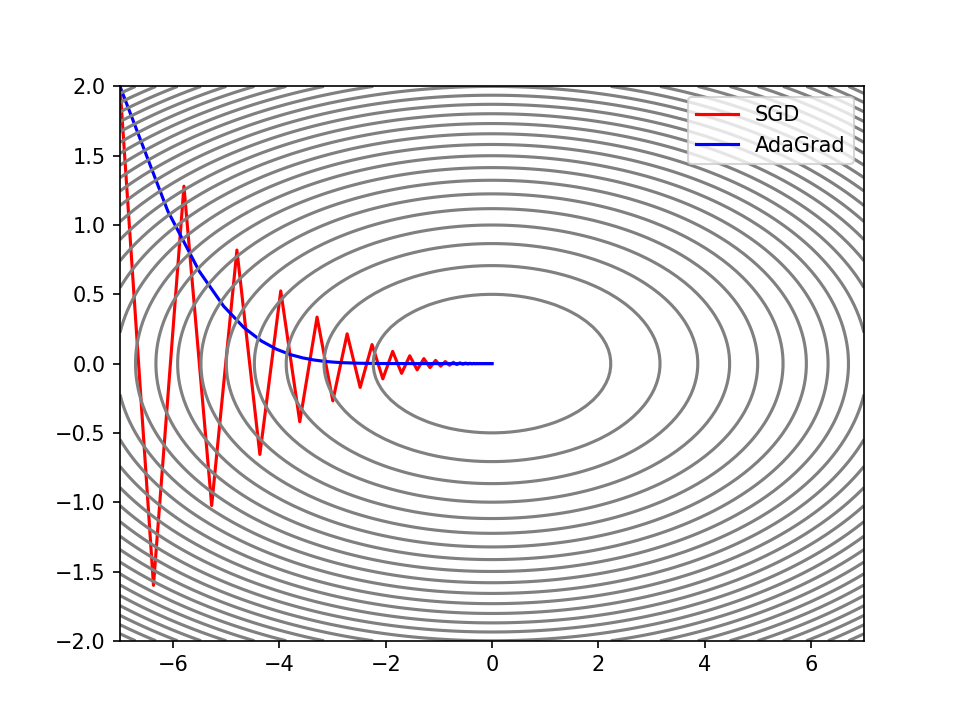
#### AdaGrad

自适应梯度，会为每个参数适当的调整学习率

* ：历史梯度的平方和

这里就表示了梯度的平方和，即 ，这里的 表示对应矩阵元素的乘法。

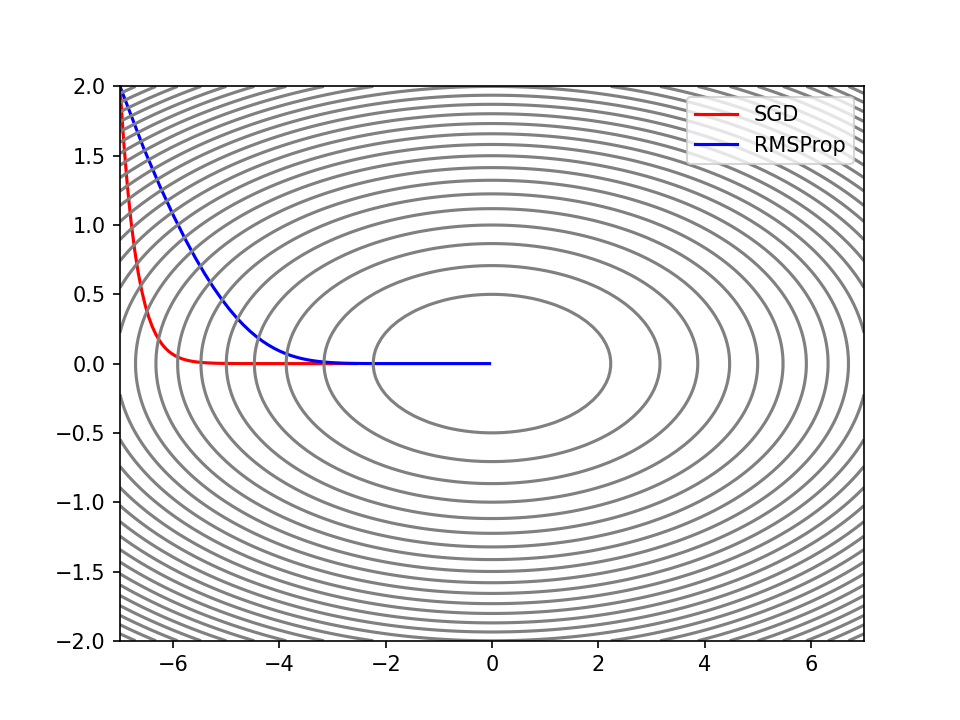
使用AdaGrad时，学习越深入，更新的幅度就越小。如果无止境地学习，更新量就会变为0，完全不再更新。



#### RMSProp

RMSProp（Root Mean Square Propagation，均方根传播）是在AdaGrad基础上的改进，采用指数移动加权平均，呈指数地减小梯度的尺度。

* *h*：历史梯度平方和的指数移动加权平均（h的n次平方）
* ：权重



#### Adam

Adam（Adaptive Moment Estimation，自适应矩估计）融合了Momentum和AdaGrad的方法

* ：学习率
* ：一次动量系数和二次动量系数
* ：迭代次数，从1开始

注：根据迭代次数来控制梯度减少的量（牛逼）

### 参数初始化

#### 常数初始化

全为1的矩阵

#### 秩初始化

单位矩阵（对角线都为1）

#### 正太分布初始化

权重参数按指定均值*μ*与标准差*σ*正态分布初始化

#### 均匀分布初始化

权重参数在指定区间内均匀分布初始化

#### Xavier初始化（也叫Glorot初始化）

输入和输出神经元数量调整权重的初始范围

Xavier正态分布初始化：均值为0，标准差为的正态分布。

Xavier均匀分布初始化：区间内均匀分布。

其中表示输入数，表示输出数。

Xavier初始化参数适用于Sigmoid和Tanh等激活函数，能有效缓解梯度消失或爆炸问题。

#### He初始化（也叫Kaiming初始化）

He初始化根据输入的神经元数量调整权重的初始范围。

He正态分布初始化：均值为0，标准差为的正态分布。

He均匀分布初始化：区间内均匀分布。

其中表示输入数。

He初始化参数主要适用于ReLU及其变体（如Leaky ReLU）激活函数

### 正则化

解决过拟合问题，提高模型泛化能力的调整，主要是对特征数据进行处理；

神经网络正则化方法:

Batch Normalization（批量标准化）、 权重衰减、Dropout、早停法；

#### Batch Normalization（批量标准化）

BN层通常放在线性层（全连接层/卷积层）之后，对线性输出的数据会先对数据进行标准化，再对数据进行缩放和平移：

* ：一个微小值，防止分母为0
* ：系数，可通过学习调整
* ：偏置，可通过学习调整

#### 权重衰减

一般会对损失函数加上一个权重的范数；最常见的就是 L2 范数的平方

* *：*表示权重的L2范数，即
* *:* 控制正则化强度的超参数。

惩罚项求导之后得到 ；所以在求权重梯度时，需要为之前误差反向传播法的结果，再加上。

#### Dropout

Dropout（随机失活，暂退法）是一种在学习的过程中随机关闭神经元的方法

注：核心思想是随机性，提高了泛化能力；