1 激活函数

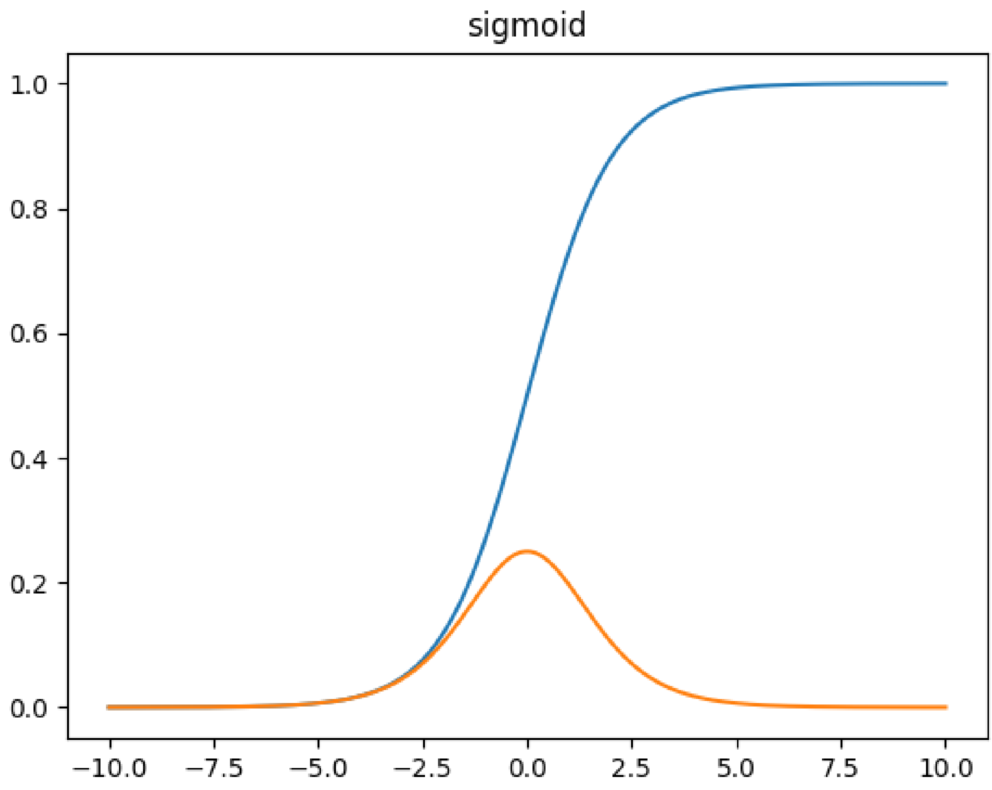
激活函数（Activation Function）是一种添加到人工神经网络中的函数，旨在帮助网络学习数据中的复杂模式。在神经元中，输入的input经过一系列加权求和后作用于另一个函数，这个函数就是这里的激活函数。

作用：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

1.1 sigmoid

（S型生长曲线）

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$
$$f'(x) = f(x)(1-f(x))$$



优点:

1. 将特征值压缩到$[0,1]$之间，在深层网络中可以保持数据幅度变化小;
2. 输入较小时，具有较大的导入，容易梯度下降;
3. 该函数适用于将**预测概率**作为输出的模型;

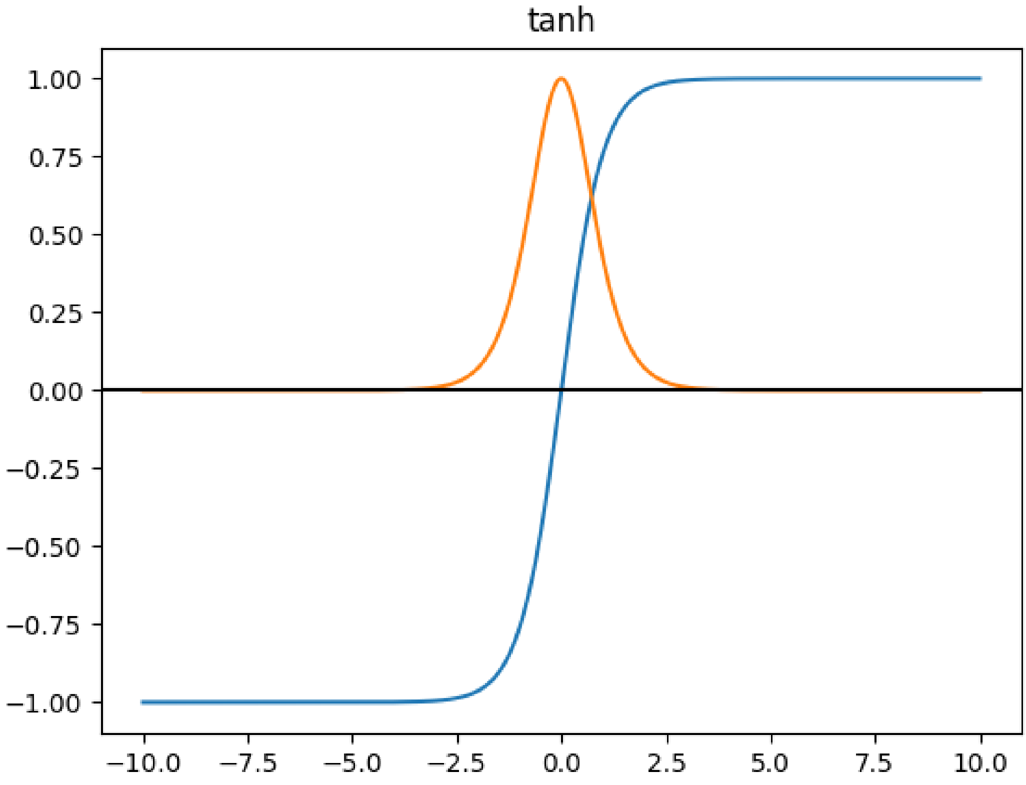
缺点:

1. **梯度消失**：当输入趋近 0 和 1的时候，变化率基本为常数，即变化非常小，进而导致梯度接近于0，无法执行反向传播
2. **收敛速度较慢**：梯度可能会过早消失，进而导致收敛速度较慢。

1.2 Tanh

(双曲正切)

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$f'(x) = 1 - f^2$$



优点：

1. 解决了的Sigmoid函数输出不是0均值的问题；
2. tanh函数的导数取值范围在$[0, 1]$之间，优于sigmoid函数的$[0, 0.25]$，一定程度上缓解了梯度消失的问题；
3. tanh函数在原点附近与$y = x$函数形式相近，当输入的激活值较低时，可以直接进行矩阵运算，训练相对容易；

缺点：

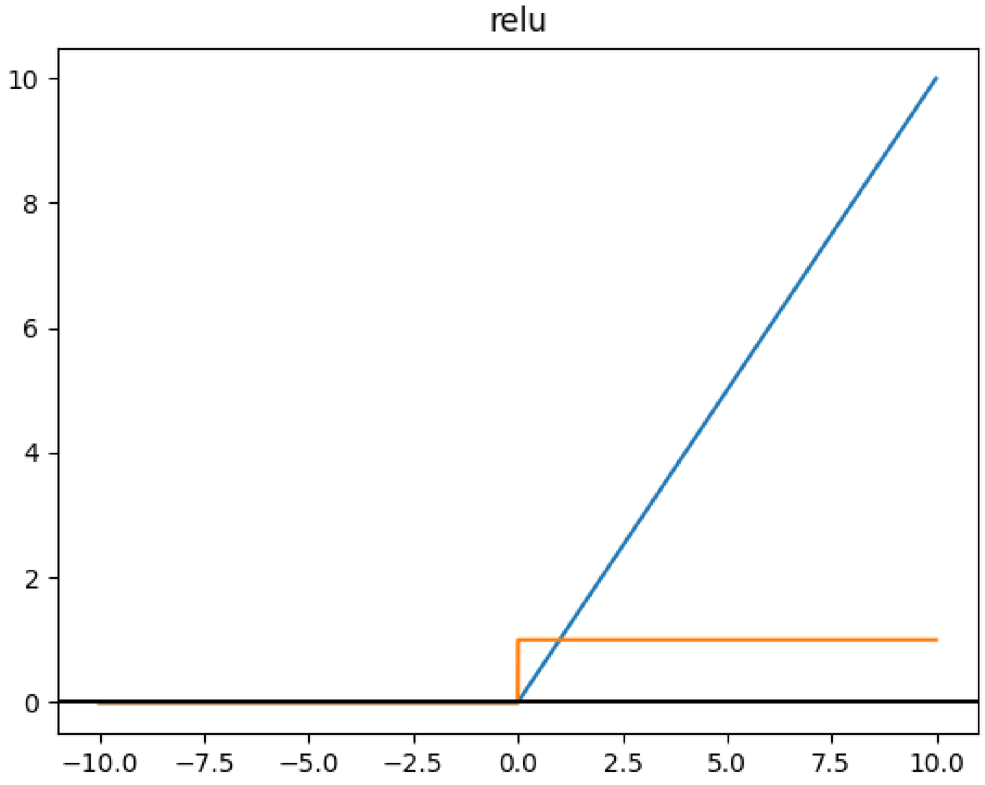
1. 梯度消失问题仍然存在

1.3 ReLU

线性整流函数(Rectified Linear Unit,ReLU)，在输入大于 0时，直接输出该值;在输入小于等于 0时，输出0。其作用在于增加神经网络各层之间的非线性关系。

$$relu(x) = max(0, x)$$

$$f'(x) = \begin{cases} 0 & x < 0 \\ 1 & otherwise \end{cases}$$



优点：

1. 相比sigmoid和tanh，**解决了梯度消失**问题，使得深层网络可训练；
2. 计算速度非常快，收敛速度快；
3. 将输出稀疏化，减少神经元冗余计算。

缺点：

1. Relu函数的输出也不是以0为均值的函数;
2. 存在**Dead Relu** 问题，某些神经元可能永远不会被激活，导致**无法更新梯度**
3. 当输入为正值，导数为1，在"链式反应"中，不会出现梯度消失，但梯度下降的强度则完全取决于权值的乘积，如此可能会导致**梯度爆炸**问题。

1.4 Leaky ReLu

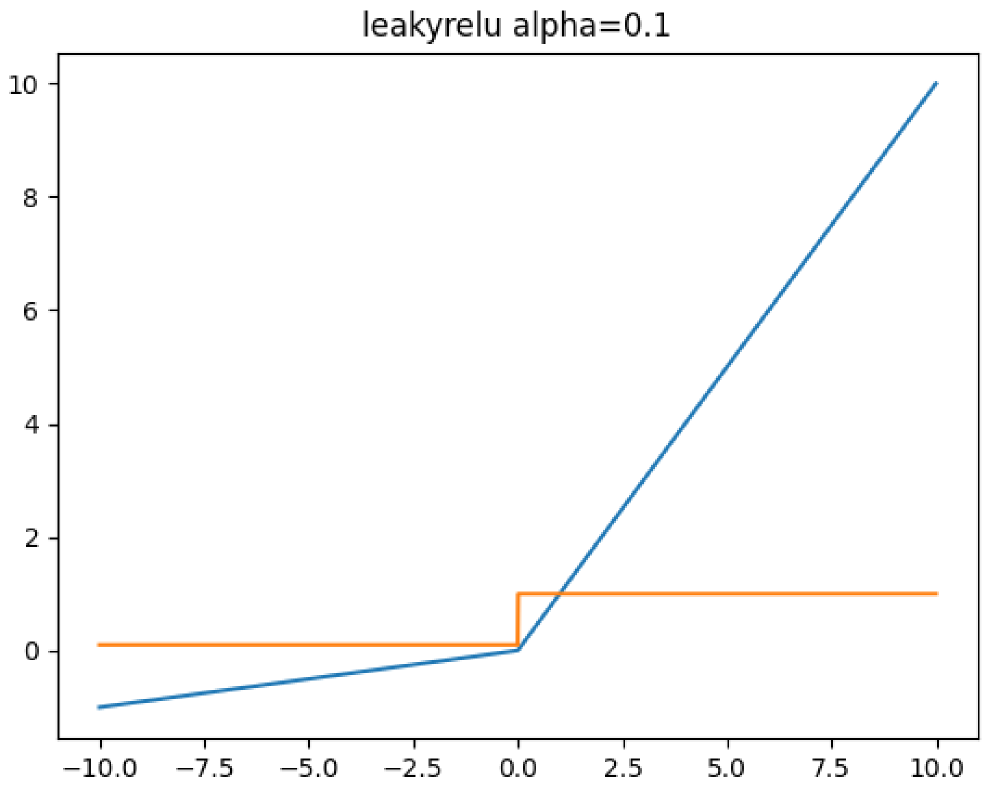
（泄漏线性整流）

LeakyReLU在神经元未激活时，它仍允许赋予一个很小的梯度α，避免ReLU死掉的问题。

值得注意的是LeakyReLU是确定的标量，不可学习。

$$LeakyReLU(x) = \begin{cases} \alpha * x & x < 0 \\ x & otherwise \end{cases}$$

$$f'(x) = \begin{cases} \alpha & x < 0 \\ 1 & otherwise \end{cases}$$



优点：与ReLU函数类似，但在输入值小于0时，导数不为0，可以避免出现“死神经元”。

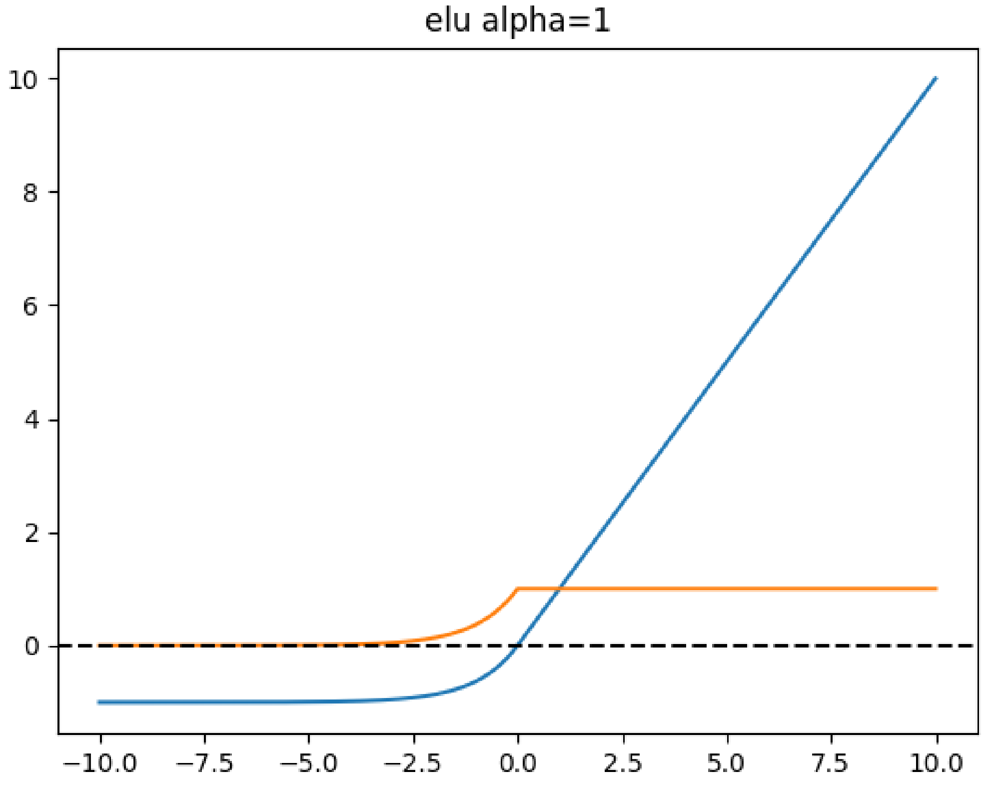
缺点：在实践中，参数α需要手动调整，较难确定其最佳值。

1.5 ELU

（指数线性单元）

$$ELU(x) = \begin{cases} \alpha * (e^x - 1) & x < 0 \\ x & otherwise \end{cases}$$

$$f'(x) = \begin{cases} \alpha * e^x & x < 0 \\ 1 & otherwise \end{cases}$$



优点：避免“死神经元”，输出的均值接近于0，并且单侧饱和可以加速收敛

缺点：计算比ReLU略慢，同样需要手动调整α

1.6 PReLU

Parametric ReLU（参数化线性整流元），与Leaky ReLU

相比于Leaky ReLU，PReLU中的α 是一个**可学习**的超参数，可以根据数据进行训练。

$$PReLU(x) = \begin{cases} \alpha * x & x < 0 \\ x & otherwise \end{cases}$$

> 如果 $\alpha = 0$，那么PReLU 就退化为ReLU;
>
> 如果α 为一个很小的常数，则PReLU 可以看作Leaky ReLU;

优点：PReLU允许网络自学习适合的α 值，使其适合不同的数据特征。

1.7 SeLU

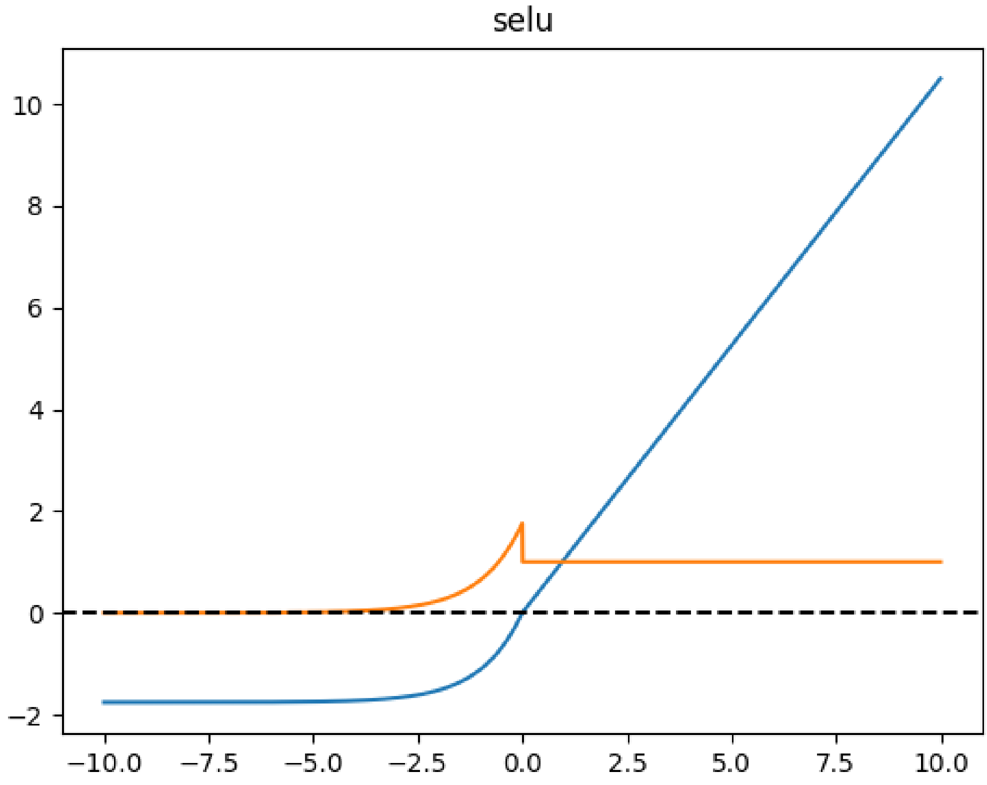
（扩展型指数线性单元激活函数）

SeLU可诱导自标准化属性（例如方差稳定化），从而避免了梯度的爆炸和消失。SeLU函数是给ELU函数乘上系数 λ , 即$SeLU(x) = \lambda * ELU(x)$

$$SeLU(x) = \lambda \begin{cases} \alpha * (e^x - 1) & x < 0 \\ x & otherwise \end{cases}$$

通过论文[Self-Normalizing Neural Networks]证明，作者给出了 λ 和 α 的值：

$$\alpha \approx 1.6732632423543772848170429916717$$
$$\lambda \approx 1.0507009873554804934193349852946$$



优点：在输入的均值为0和方差为1时，SELU函数足以自主统计的均值和方差保持在一个稳定的范围内，使数据的分配能力足够更好地适用于网络的训练。

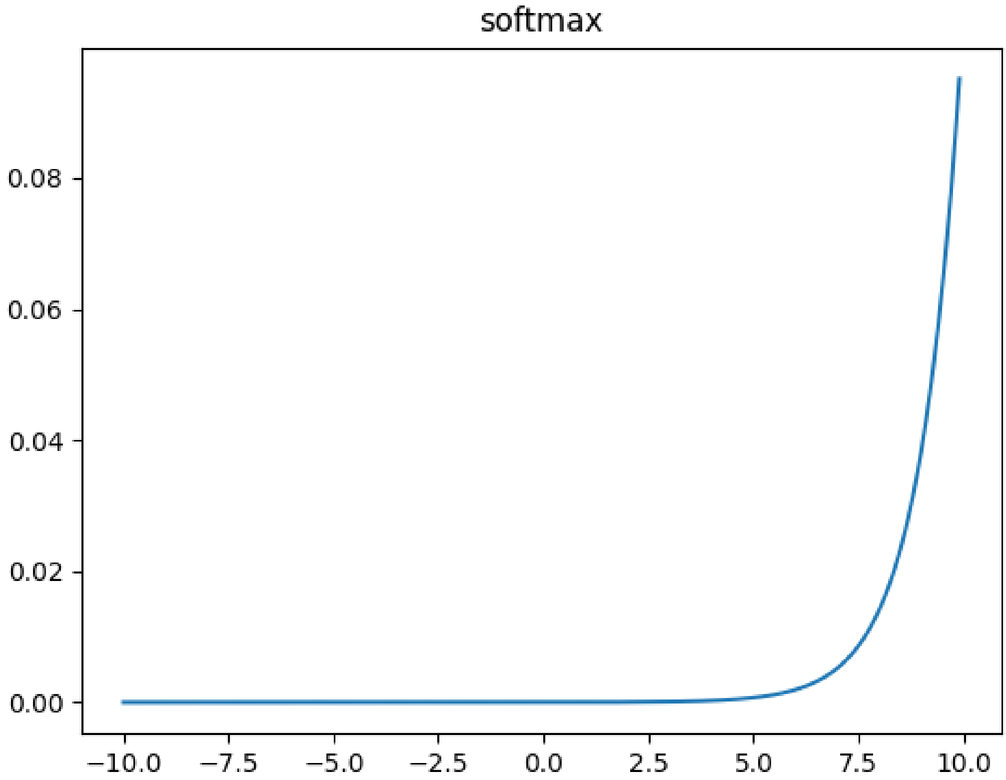
> 关于SeLU详细可参考 https://zhuanlan.zhihu.com/p/98863801

1.8 Softmax

(归一化指数函数)

用于多分类，对于长度为K的任意实向量，Softmax可以将其压缩为长度K，值在$[0, 1]$内且向量元素总和为1的实向量。

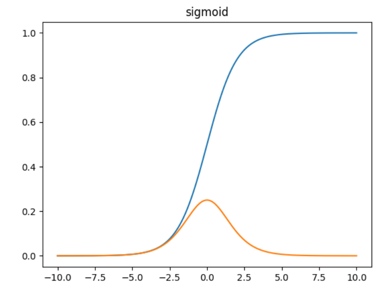
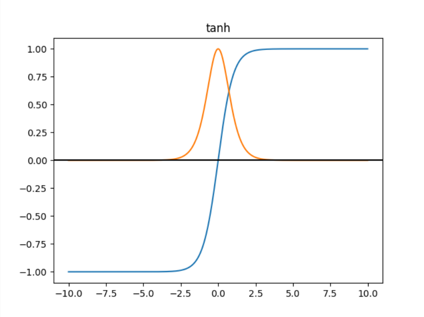
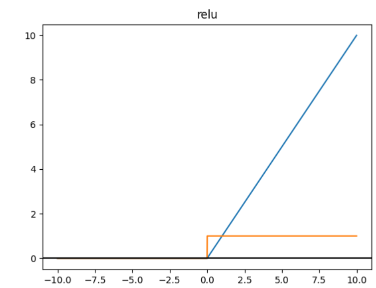
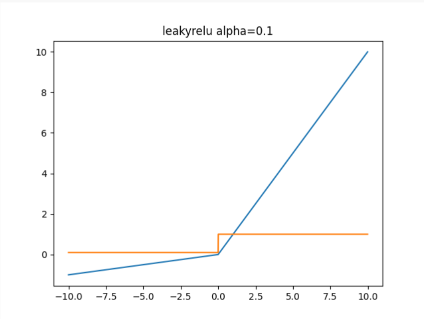
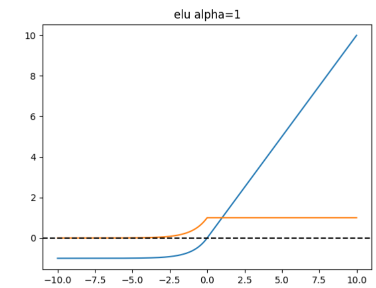
$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

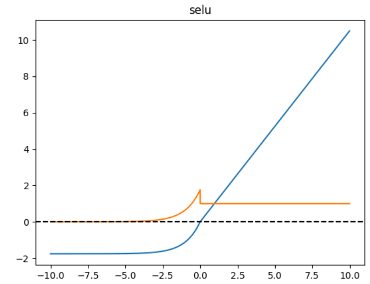
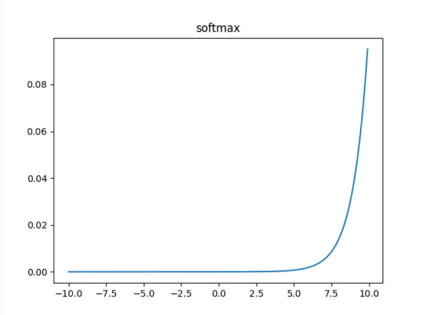


优点：将输出值转为**概率分布**，非常适合**多分类**问题

通常用于分类问题的输出层前的归一化操作。

总结对比

函数	含义	图像	优点	缺点
sigmoid	S型生长曲线		容易梯度下降	容易梯度消失
Tanh	双曲正切		比sigmoid函数具有更广的范围	容易梯度消失
Relu	线性整流函数		解决了梯度消失，计算和收敛速度快	输入值小于0时，导数为0，有Dead Relu问题
Leaky ReLU	泄漏线性整流		可以避免出现“死神经元”	参数α需要手动调整
ELU	指数线性单元		避免“死神经元”，并且单侧饱和可以加速收敛	速度略慢，α需要调整
PREeLu	参数化线性整流元	与Leaky ReLU相似	自学习适合的α 值	

函数	含义	图像	优点	缺点
SeLU	扩展型指数线性单元激活函数		自标准化属性，避免梯度爆炸和消失	
softmax	归一化指数函数		将输出值转为**概率分布**，非常适合**多分类**问题	

绘制代码

```
# %% sigmoid
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10, 10, 1000)
sigmoid = 1 / (1 + np.e ** (-x))
dsigmoid = sigmoid * (1 - sigmoid)
plt.plot(x, sigmoid)
plt.plot(x, dsigmoid)
plt.title('sigmoid')
plt.show()

# %% tanh
_x = np.arange(-10, 10, 0.01)
tanh = np.tanh(_x)
dtanh = 1 - np.power(tanh, 2)
plt.plot(_x, tanh)
plt.plot(_x, dtanh)
plt.axhline(0, color='black', linestyle='-')
plt.title('tanh')
plt.show()

# %% relu
_x = np.arange(-10, 10, 0.01)
relu = np.maximum(0, _x)
drelu = [0 if x < 0 else 1 for x in _x]
plt.plot(_x, relu)
plt.plot(_x, drelu)
plt.axhline(0, color='black', linestyle='-')
plt.title('relu')
plt.show()

# %% leaky relu
```

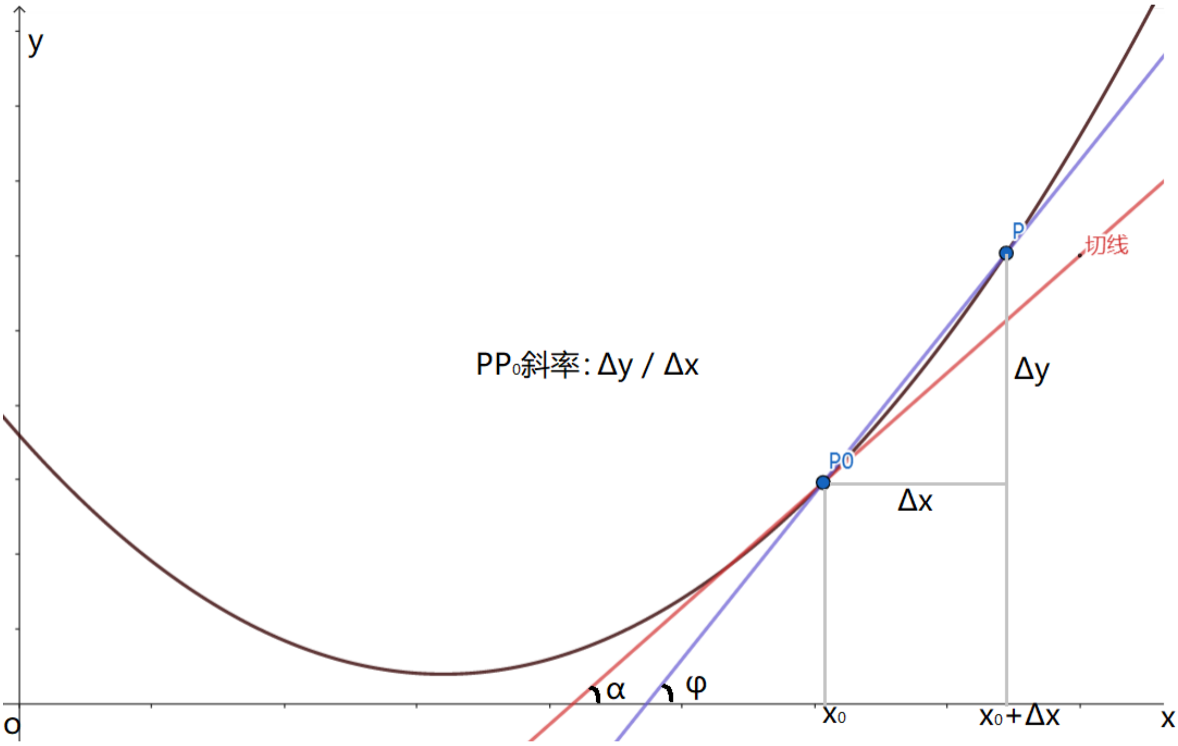
```python
alpha = 0.1
_x = np.arange(-10, 10, 0.01)
leakyrelu = [alpha * x if x < 0 else x for x in _x]
dleakyrelu = [alpha if x < 0 else 1 for x in _x]
plt.plot(_x, leakyrelu)
plt.plot(_x, dleakyrelu)
plt.title('leakyrelu alpha=0.1')
plt.show()

# %% elu
alpha = 1
_x = np.arange(-10, 10, 0.01)
elu = [alpha * (np.expm1(x)) if x < 0 else x for x in _x]
delu = [alpha * (np.exp(x)) if x < 0 else 1 for x in _x]
plt.plot(_x, elu)
plt.plot(_x, delu)
plt.axhline(0, color='black', linestyle='--')
plt.title('elu alpha=1')
plt.show()

# %% selu
alpha = 1.6733
lambda_ = 1.0507
_x = np.arange(-10, 10, 0.01)
elu = [lambda_ * (alpha * (np.expm1(x)) if x < 0 else x) for x in _x]
delu = [lambda_ * alpha * (np.exp(x)) if x < 0 else 1 for x in _x]
plt.plot(_x, elu)
plt.plot(_x, delu)
plt.axhline(0, color='black', linestyle='--')
plt.title('selu')
plt.show()

# %% softmax
_x = np.arange(-10, 10, 0.1)
softmax = np.exp(_x) / np.sum(np.exp((_x)))
plt.plot(_x, softmax)
plt.title('softmax')
plt.show()
```

2 导数图

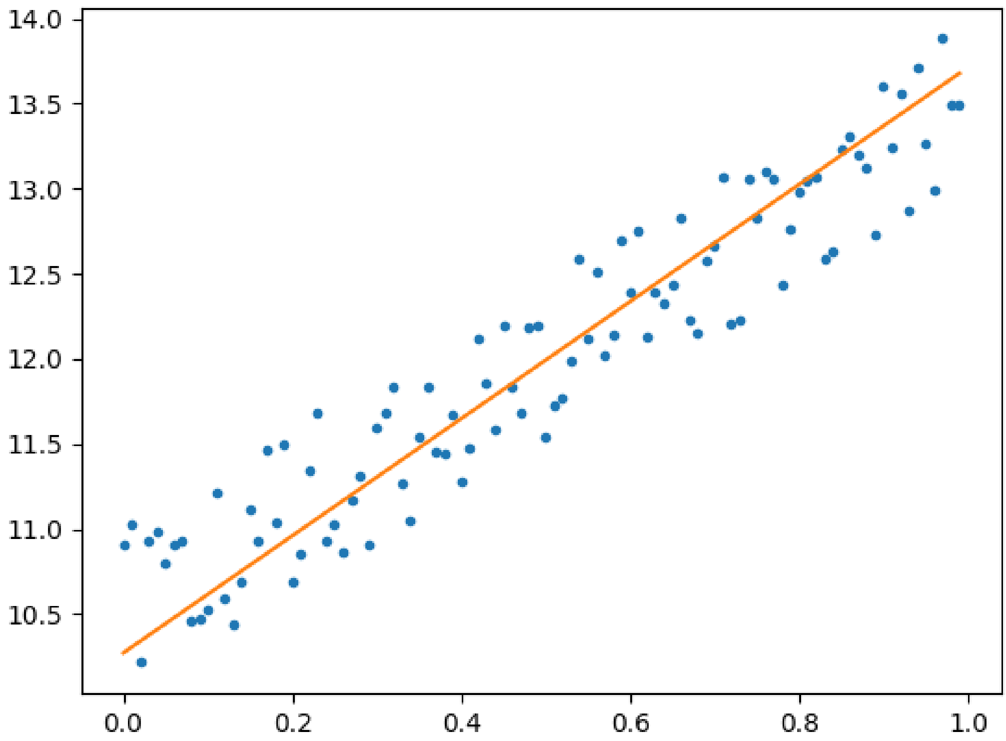


3 线性回归实例

```
import random
import matplotlib.pyplot as plt

# 输入参数x
_x = [i / 100 for i in range(100)]
# 随机数初始化权重 w,b，通过计算更新权重。
w = random.random()
b = random.random()
# 输出参数 y = wx + b 线性回归到w = 3, b = 10，生成模拟的数据集
_y = [3 * e + 10 + random.random() for e in _x]
while True:
    # zip打包成元组 (_x, _y)
    for x, y in zip(_x, _y):
        # x固定，当前w、b的值计算出的答案为 h，y是标准答案。
        h = w * x + b
        # 当前权重(wb)计算的结果和标准答案的损失
        loss = (x * w + b - y) ** 2
        # 对w，b求导数
        dw = -2 * (h - y) * x  # 对 (x * w + b - y) ** 2 求w偏导，b视为常量
        db = -2 * (h - y) * 1  # 对 (x * w + b - y) ** 2 求b偏导，w视为常量
        # w,b权值更新(学习率lr=0.01)
        lr = 0.01
        w += dw * lr
        b += db * lr
        print("w:", w, "\tb:", b)
        plt.ion()  # 开始交互模式
        plt.cla()  # 清屏
        plt.plot(_x, _y, ".")
        # 权值更新后的回归线
        y1 = [w * e + b for e in _x]
        plt.plot(_x, y1)
        plt.pause(0.01)
        # plt.ioff()  # 关闭交互模式
```

```
plt.show()
```



> 参考：
>
> https://zhuanlan.zhihu.com/p/364620596
>
> https://zhuanlan.zhihu.com/p/98863801