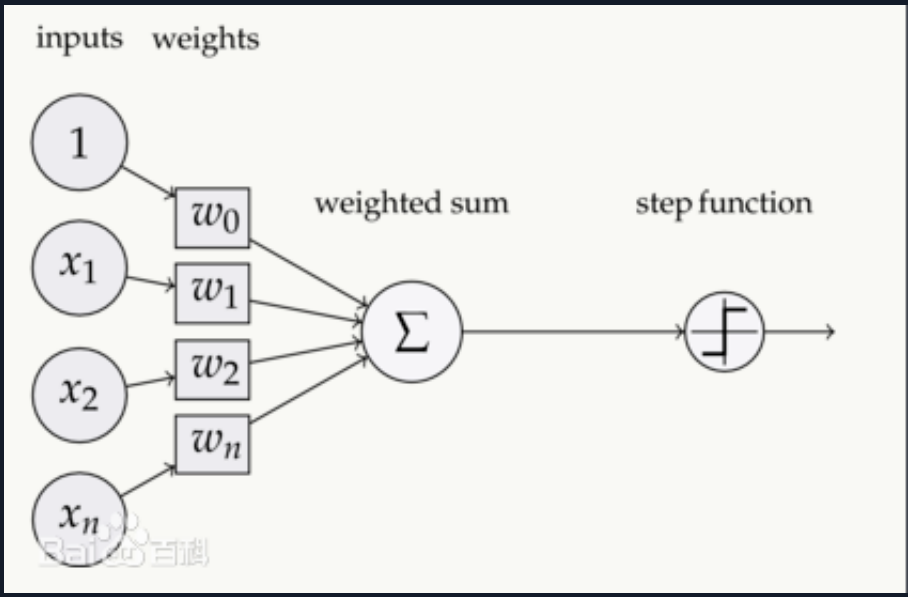
H2

# 深度学习中激活函数总结

### H3 一、激活函数的定义

> 激活函数（Activation functions）对于人工神经网络模型去学习、理解非常复杂和非线性的函数来说具有十分重要的作用。它们将非线性特性引入到我们的网络中。




> - 在神经元中，输入的 inputs 通过加权，求和后，还被作用了一个函数，这个函数就是激活函数。
> - 引入激活函数是为了增加神经网络模型的非线性。没有激活函数的煤层都相当于矩阵相乘，就算叠加若干层后，无非还是个矩阵相乘。

### H3 二、采用激活函数的原因

> - 如果不用激活函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合，这种情况就是最原始的感知机（Pperceptron）。
> - 如果使用的话，激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。

### H3 三、什么函数可用作激活函数

> 激活函数必为非线性函数，另外激活函数在神经元数足够的情况下，应尽可能使神经网络可以实现对任何一个输入向量到输出向量的连续映射函数逼近，即：万能逼近定理（universal approximation theorrm）。

### 四、如何评价激活函数的好坏

> 好的激活函数就是需要尽可能避免深度学习中的梯度消失、梯度爆炸两个问题

#### 4.1 梯度消失

> 反向传播算法在计算误差时每一层都会乘以本层激活函数的导数。如果本层激活函数的导数绝对值小于1，那么多次连乘后会导致误差项逐渐接近于0，由于参数的梯度值由误差值计算所得，因此会导致前几层的权重梯度接近于0，这样参数并不会得到更新。因此，该现象被称为梯度消失问题

#### 4.2 梯度爆炸

> 梯度爆炸可以看做与梯度消失相反，即激活函数的导数绝对值大于1，多次连乘后的误差项会趋于一个非常大的数，由此造成梯度爆炸。
>
> 梯度爆炸引发的问题 在深度多层感知机网络中，梯度爆炸会引起网络不稳定，最好的结果是无法从训练数据中学习，而最坏的结果是出现无法再更新的 NaN 权重值。

### 五、常用激活函数

#### 5.1 二元阶梯函数（Binary Step Function）

##### 5.1.1 定义

> 最简单的决定神经元是否被激活的方式是基于分类器使用一个二元阶梯函数。当分类器结果Y高于某个值得时候，神经元被激活，否则不被激活

##### 5.1.2 函数公式

$$f(x) = \begin{cases} 1\,,if & x >= 0 \\ 0\,,if & x < 0 \end{cases}$$

##### 5.1.3 使用场景

> 二元阶梯函数可以用来创建二分类器。当我们对一个东西只要简单地回答“是”或者“否”的时候，阶梯函数是很好的选择，因为它使得神经元要么被激活，要么被抛弃。
>
> 通常只是在理论上使用

##### 5.1.4 优点

> - 简单

##### 5.1.5 缺点

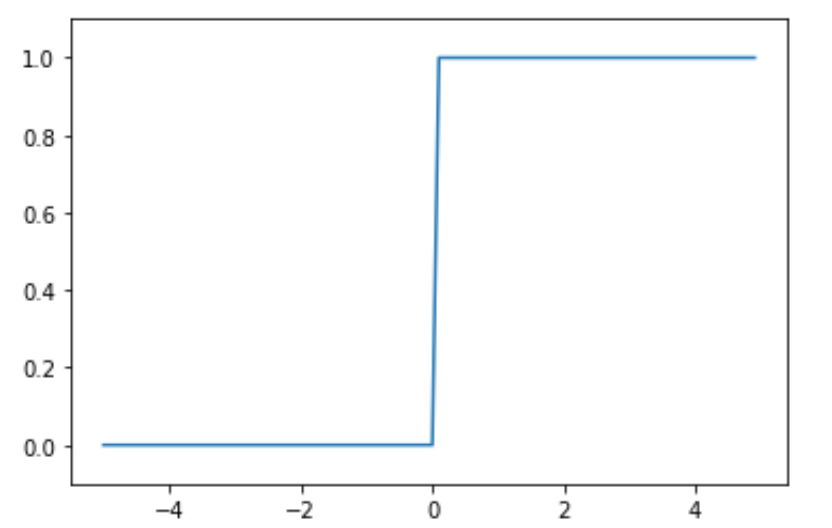
> - 无法做多分类
> - 阶梯函数的梯度是0，使得模型不会有任何实际的提升。这使得梯度函数在后向传播算法中不可行，无法进行后向损失传播。

##### 5.1.6 图形

```
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt

def step_function(x):
    return np.array(x > 0, dtype=np.int)

X = np.arange(-5.0, 5.0, 0.1)
Y = step_function(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)  # 指定图中绘制的y轴的范围
plt.show()
np.arange(-5.0, 5.0, 0.1)
```




#### 5.2 线性函数（Linear Function）

##### 5.2.1 定义

> 线性函数是指那些线性的函数

##### 5.2.2 函数公式

$$f(x) = ax$$

##### 5.2.3 使用场景

> 可以同时应用在多个不同的神经元上，可以同时激活多个神经元

##### 5.2.4 优点

> - 简单
> - 可以进行多分类（取结果最大的类别）

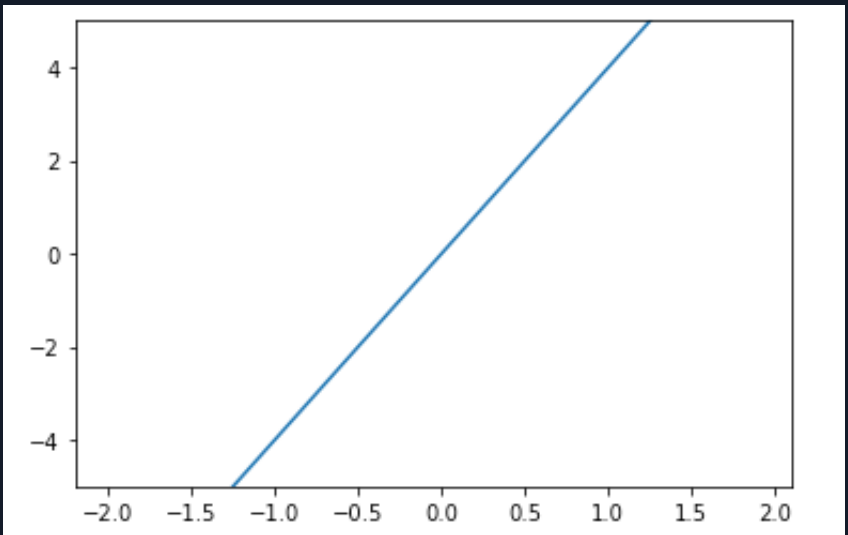
##### 5.2.5 缺点

> - 在做反向传播的时候，阶梯函数的梯度总是常量，使得模型不会有任何实际的提

> 升。而且，这是一个线性的变换，不管我们有多少个层，我们最终其实都是在做线性变换。

##### 5.2.6 图形

```
# 线性函数
X = np.arange(-2.0, 2.0, 0.1)
Y = 4 * X
plt.plot(X, Y)
plt.ylim(-5, 5)  # 指定图中绘制的y轴的范围
plt.show()
```




### 5.3 sigmoid函数

#### 5.3.1 定义

> - sigmoid函数是一个连续且可微的函数。
> - sigmoid函数连续，光滑，严格单调，以(0,0.5)中心对称，是一个非常良好的阈值函数。
> - 当x趋近负无穷时，y趋近于0；趋近于正无穷时，y趋近于1；x=0时，y=0.5。当然，在x超出[-6,6]的范围后，函数值基本上没有变化，值非常接近，在应用中一般不考虑。

#### 5.3.2 函数公式

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

#### 5.3.3 使用场景

> sigmoid激活函数在特征相差比较复杂或是相差不是特别大时，使用sigmoid效果比较好。

#### 5.3.4 优点

- Sigmoid 函数的输出映射在(0,1)之间，单调连续，输出范围有限，优化稳定，可以用作输出层。
- 它在物理意义上最为接近生物神经元。求导容易。

##### H5 5.3.4 缺点

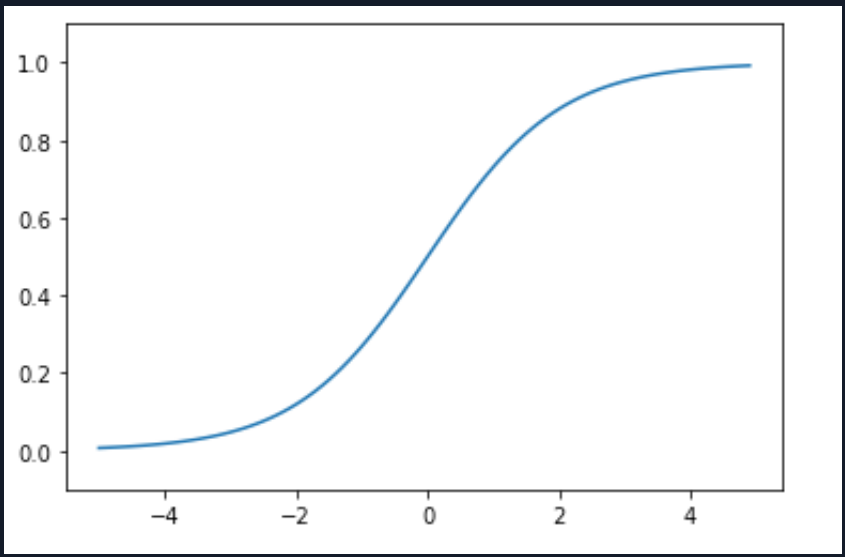
- 因为其在两端存在软饱和区，所以导数就会接近0，在深度神经网络中，容易导致梯度消失（一般来说， sigmoid 网络在 5 层之内就会产生梯度消失现象）。
- sigmod的输出不是zero-centered（0均值），称为偏移现象，会导致后一层的神经元得到上一层输出的非0值信号作为输入值，可能同时导致收敛缓慢。
- 计算过程含幂运算，求解耗时。

##### H5 5.3.5 图形

```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```




#### H4 5.4 tanh函数

##### H5 5.4.1 定义

##### H5 5.4.2 函数公式

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

### 5.4.3 使用场景

> tanh在特征相差明显时的效果会很好，在循环过程中会不断扩大特征效果。

### 5.4.4 优点

> - 比Sigmoid函数收敛速度更快。
> - 相比Sigmoid函数，其输出以0为中心。

### 5.4.5 缺点

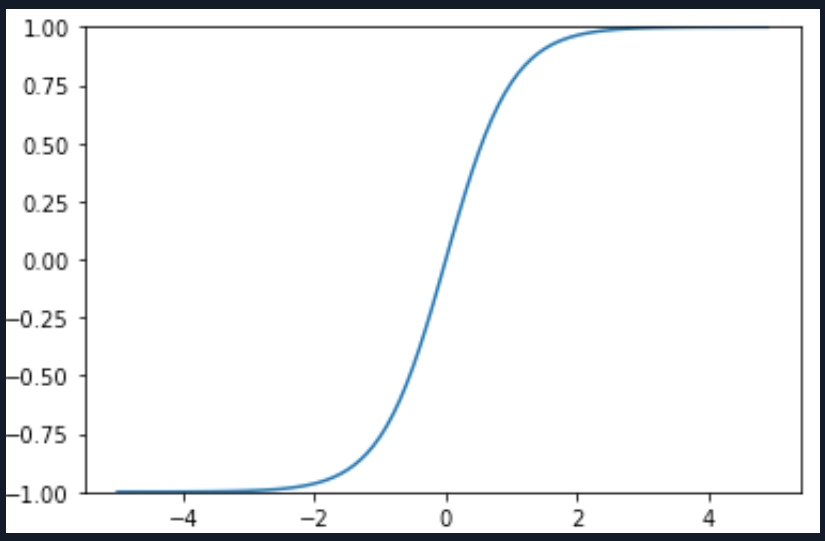
> - 还是没有改变Sigmoid函数的最大问题——由于饱和性产生的梯度消失。

### 5.4.6 图形

```
# coding: utf-8
# tanh函数
import numpy as np
import matplotlib.pylab as plt

def sigmoid(x):
    return (1 - np.exp(-2*x)) / (1 + np.exp(-2*x))

X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-1.0, 1.0)
plt.show()
```




## 5.5 ReLU函数

### 5.5.1 定义

> - ReLU的全称是Rectified Linear Units，是一种后来才出现的激活函数。 可以看到，当x<0时，ReLU硬饱和，而当x>0时，则不存在饱和问题。所以，ReLU 能够在x>0时保持梯度不衰减，从而缓解梯度消失问题。这让我们能够直接以监督的方

式训练深度神经网络，而无需依赖无监督的逐层预训练。

- 然而，随着训练的推进，部分输入会落入硬饱和区，导致对应权重无法更新。这种现象被称为“神经元死亡”。与sigmoid类似，ReLU的输出均值也大于0，偏移现象和 神经元死亡会共同影响网络的收敛性。

### H5 5.5.2 函数公式

$$f(x) = \begin{cases} x\ ,if & x >= 0 \\ 0\ ,if & x < 0 \end{cases}$$
$$f(x) = max(0, x)$$

### H5 5.5.3 使用场景

tanh函数在特征相差明显时的效果会很好，在循环过程中会不断扩大特征效果。

### H5 5.5.4 优点

- 相比起Sigmoid和tanh，ReLU在SGD中能够快速收敛。据称，这是因为它线性、非饱和的形式。
- Sigmoid和tanh涉及了很多很expensive的操作（比如指数），ReLU可以更加简单的实现。
- 有效缓解了梯度消失的问题。
- 在没有无监督预训练的时候也能有较好的表现。
- 提供了神经网络的稀疏表达能力。

### H5 5.5.5 缺点

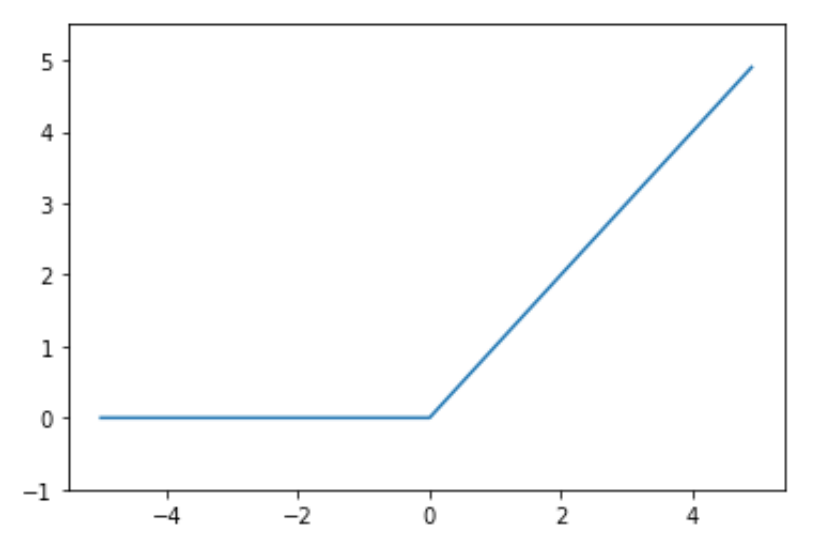
- 随着训练的进行，可能会出现神经元死亡，权重无法更新的情况。
- 如果发生这种情况，那么流经神经元的梯度从这一点开始将永远是0。也就是说，ReLU神经元在训练中不可逆地死亡了。
    - 举个例子：当在反向传播时，如果此时流进网络的梯度很大，如果权重系数被更新成很大的负数后，此时很多输入经过该神经元输出都会是0。此时就落入了硬饱和区域了。而这个神经元的梯度将一直都是0。要解决这个问题就是要在设置学习率的时候，尽量不要设置太大的学习率，这样可以有效的避免神经元的失活

### H5 5.5.6 图形

```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt

def relu(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.ylim(-1.0, 5.5)
plt.show()
```

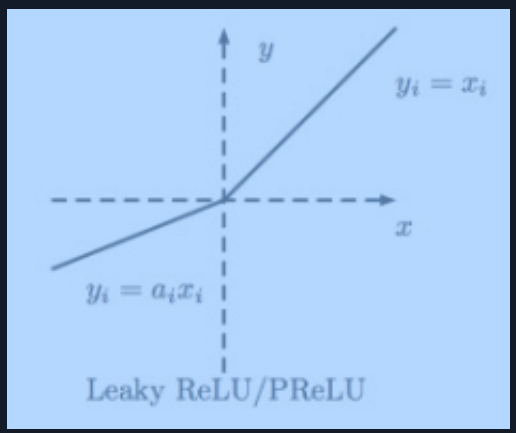



### H5 5.5.7 优化

> 针对在x<0的硬饱和问题，我们对ReLU做出相应的改进，使得

$$f(x) = \begin{cases} x\ ,\ if & x >= 0 \\ \frac{x}{a}\ ,\ if & x < 0 \end{cases}$$

> 这就是Leaky-ReLU（其中a是大于1的固定参数），而P-ReLU认为，α也可以作为一个参数来学习，原文献建议初始化a为0.25，不采用正则。




### H4 5.6 ELU

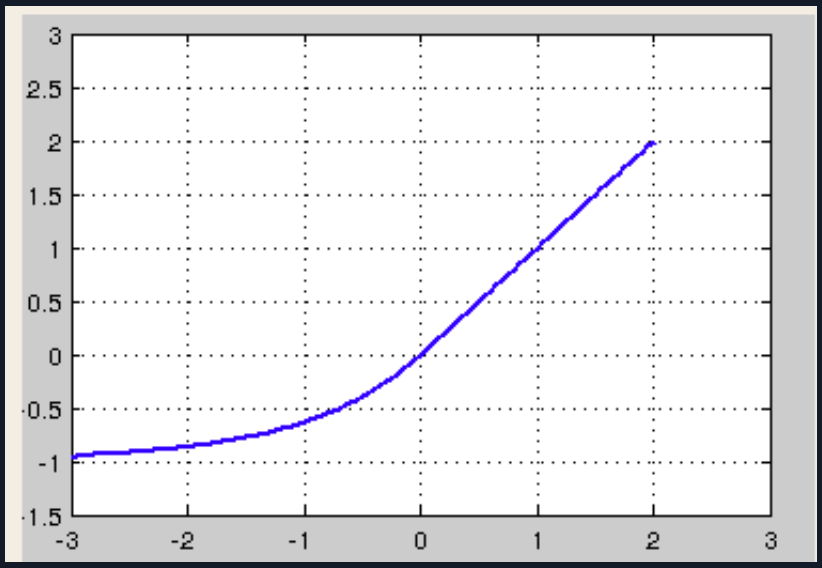
##### 5.6.1 定义

> 融合了sigmoid和ReLU，左侧具有软饱和性，右侧无饱和性。右侧线性部分使得ELU能够缓解梯度消失，而左侧软饱能够让ELU对输入变化或噪声更鲁棒。ELU的输出均值接近于零，所以收敛速度更快。在 ImageNet上，不加 Batch Normalization 30 层以上的 ReLU 网络会无法收敛，PReLU网络在MSRA的Fan-in （caffe ）初始化下会发散，而 ELU 网络在Fan-in/Fan-out下都能收敛

##### 5.6.2 函数公式

$$f(x) = \begin{cases} x\ ,\ if \quad x >= 0 \\ a(e^x - 1)\ ,\ if \quad x < 0 \end{cases}$$

##### 5.6.3 图形




#### 5.7 softmax函数

##### 5.7.1 定义

> softmax是另一种Sigmoid函数，但是它是在分类中比较容易控制的一种激活函数。

##### 5.7.2 函数公式

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



##### 5.7.3 使用场景

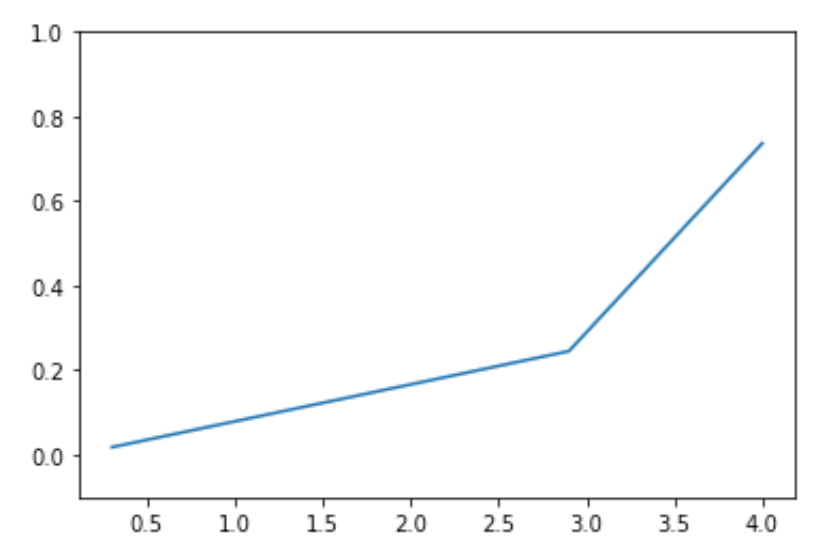
> 在多分类场景中可以用softmax也可以用多个二分类器组合成多分类，比如多个逻辑分类器或SVM分类器等等。该使用softmax还是组合分类器，主要看分类的类别是否互斥，如果互斥则用softmax，如果不是互斥的则使用组合分类器。

##### 5.7.4 图形

```
import numpy as np
import matplotlib.pyplot as plt
```

```python
def softmax(x):
    c = np.max(x)
    exp_x = np.exp(x - c) # 解决溢出问题
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y

x = np.array([0.3, 2.9, 4.0])
y = softmax(x)
plt.plot(x, y)
plt.ylim(-0.1, 1.0)
plt.show()
```




#### H4 5.8 Maxout

##### H5 5.8.1 定义

> Maxout模型实际上也是一种新型的激活函数，在前馈式神经网络中，Maxout的输出即取该层的最大值，在卷积神经网络中，一个Maxout feature map可以是由多个feature map取最值得到。
> maxout的拟合能力是非常强的，它可以拟合任意的的凸函数。但是它同dropout一样需要人为设定一个k值。

##### H5 5.8.2 函数公式

$$f(x) = max(w_1^T x + b_1 , w_2^T x + b_2 , \dots , w_n^T x + b_n)$$

### H3 六、激活函数的选择

> - 对于激活函数的选用，可以根据神经网络的用途及其场景，再加上对于激活函数的值域的了解，大致可以选定适合对应用途以及场景的激活函数例如，对于分类器，最终输出的是输入样本，在某一类上的可能性（概率），而概率值一般在[0,1]之间，因而最后一层输出的时候，可以选用值域在[0,1]之间的激活函数，比如说sigmoid函数。诸如此类的问题，可以根据值域来选择激活函数的运用

- 如果使用 ReLU，要小心设置 learning rate，注意不要让网络出现很多 “dead” 神经元，如果不好解决，可以试试 Leaky ReLU、PReLU 或者 Maxout.
- Sigmoid函数以及它们的联合通常在分类器的中有更好的效果
- 由于梯度崩溃问题，在某些时候需要避免使用Sigmoid和Tanh激活函数
- ReLU函数是一种常见的神经元，我们可以使用Leaky ReLU函数
- 记住，ReLU永远只在隐藏层中使用
- **根据经验，一半可以从ReLU激活函数开始，但是如果ReLU不能很好的解决问题，再去尝试其他的激活函数**

---

激活函数

激活函数简介

深度学习中常见激活函数以及什么时候该使用激活函数

深度学习笔记（十三）---多分类任务中的softmax以及各激活函数的适用场景

深度学习：激活函数的比较和优缺点，sigmoid，tanh，relu