


深度学习随笔——激活函数(Sigmoid、Tanh、ReLU、Leaky ReLU、PReLU、RReLU、ELU、SELU、Maxout、Softmax、Swish、Softplus)

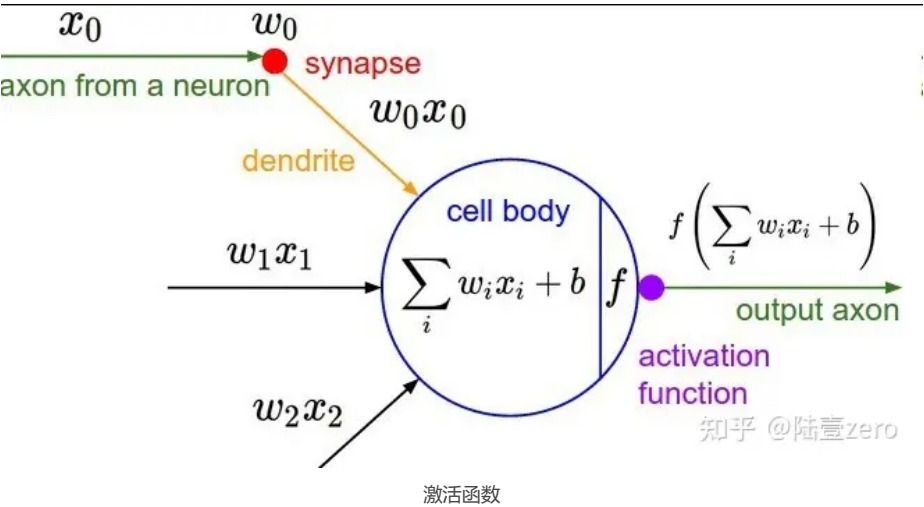


Lu1zero9

CV Rookie | Guitarist

一、什么是激活函数？

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。如下图，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。没有激活函数的每层都相当于矩阵相乘。就算你叠加了若干层之后，无非还是个矩阵相乘罢了。



二、为什么要用激活函数？

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（Perceptron）。

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。

- 收起
- 是激活函数？

么要用激活函数？

函数一览表

的激活函数？

oid激活函数

激活函数

激活函数

y ReLU激活函数

.U激活函数

.U激活函数

激活函数

激活函数

out激活函数

tmax激活函数

sh激活函数

tplus激活函数

点补充学习

传入神经网络层的数据...

消失和梯度爆炸以及解...

神经元(Saturated Neu...

l"数星"(ReLU可以解决...

饱和

解缠和对噪声的鲁棒性

性

ReLU问题

心化激活


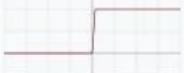

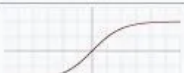
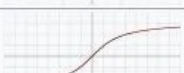



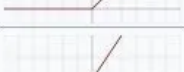







线性单元和偏置偏移

激活函数分类

三、激活函数一览表



神经网络中使用激活函数来加入非线性因素，提高模型的表达能力。

| Name | Plot | Equation |
|---|---|---|
| Identity |  | $f(x) = x$ |
| Binary step |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Logistic (a.k.a. Sigmoid or Soft step) |  | $f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}^{[1]}$ |
| TanH |  | $f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ |
| ArcTan |  | $f(x) = \tan^{-1}(x)$ |
| Softsign ^{[9][10]} |  | $f(x) = \frac{x}{1 + x }$ |
| Inverse square root unit (ISRU) ^[11] |  | $f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$ |
| Rectified linear unit (ReLU) ^[12] |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Leaky rectified linear unit (Leaky ReLU) ^[13] |  | $f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Parametric rectified linear unit (PReLU) ^[14] |  | $f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Randomized leaky rectified linear unit (RReLU) ^[15] |  | $f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}^{[3]}$ |
| Exponential linear unit (ELU) ^[16] |  | $f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$ |
| Scaled exponential linear unit (SELU) ^[17] | | $f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$ |
| S-shaped rectified linear activation unit (SReLU) ^[18] | | $f_{t_l, a_l, t_r, a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ t_l, a_l, t_r, a_r are parameters. |
| Inverse square root linear unit (ISRLU) ^[11] |  | $f(x) = \begin{cases} \frac{x}{\sqrt{1 + \alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Adaptive piecewise linear (APL) ^[19] | | $f(x) = \max(0, x) + \sum_{s=1}^S a_s^s \max(0, -x + b_s^s)$ |
| SoftPlus ^[20] |  | $f(x) = \ln(1 + e^x)$ |
| Bent identity |  | $f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$ |
| Sigmoid-weighted linear unit (SiLU) ^[21] (a.k.a. Swish ^[22]) | | $f(x) = x \cdot \sigma(x)^{[5]}$ |
| SoftExponential ^[23] |  | |

| | | |
|--------------------------|--|--|
| | | $\begin{cases} \frac{e^{\alpha x}-1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$ |
| Sinusoid ^[24] | | $f(x) = \sin(x)$ |
| Sinc | | $f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$ |
| Gaussian | | $f(x) = e^{-x^2}$ |

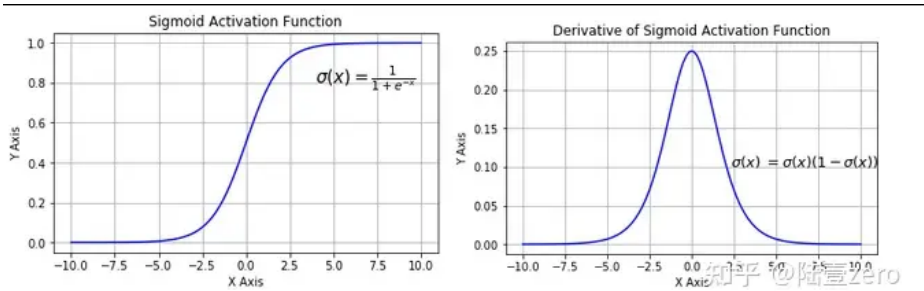
四、常用的激活函数？

1.Sigmoid激活函数

sigmoid函数也叫Logistic函数，用于隐层神经元输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。公式如下：

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

函数图像：



Sigmoid作为激活函数有以下优缺点：

优点：

- 1. Sigmoid函数的输出在(0,1)之间，输出范围有限，优化稳定，可以用作输出层。
- 2. 连续函数，便于求导。

缺点：

- 1. sigmoid函数在变量取绝对值非常大的正值或负值时会出现**饱和现象**，意味着函数会变得很平，并且对输入的微小改变会变得不敏感。在反向传播时，当梯度接近于0，权重基本不会更新，很容易就会出现**梯度消失**的情况，从而无法完成深层网络的训练。
- 2. **sigmoid函数的输出不是0均值的**，会导致后层的神经元的输入是非0均值的信号，这会对梯度产生影响。
- 3. **计算复杂度高**，因为sigmoid函数是指数形式。

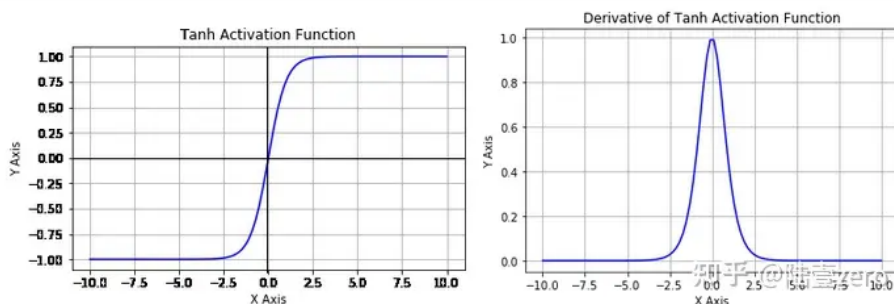
2.Tanh激活函数

Tanh的诞生比Sigmoid晚一些，sigmoid函数如上文所说有一个缺点就是输出不以0为中心，使得收敛变慢的问题。而Tanh则就是解决了这个问题。Tanh就是双曲正切函数，取值范围为[-1,1]。

Tanh函数定义如下：

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

函数图像:



实际上, Tanh函数是 sigmoid 的变形:

$$\tanh(x) = 2\sigma(x) - 1$$

Tanh函数是 0 均值的, 因此实际应用中 Tanh 会比 sigmoid 更好。但是仍然存在梯度饱和与exp计算的问题。

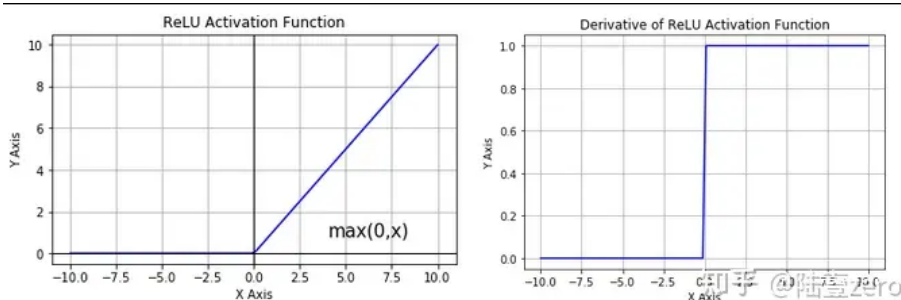
3.ReLU激活函数

整流线性单元(Rectified linear unit, ReLU)是现代神经网络中最常用的激活函数, 大多数前馈神经网络默认使用的激活函数。

ReLU函数公式:

$$f(x) = \max(0, x)$$

函数图像:



优点:

1. 使用ReLU的SGD算法的收敛速度比 sigmoid 和 tanh 快。
2. 在 $x > 0$ 区域上, 不会出现梯度饱和、梯度消失的问题。
3. 计算复杂度低, 不需要进行指数运算, 只要一个阈值就可以得到激活值。
4. 代表性稀疏, ReLU会使一部分神经元的输出为0, 这样就造成了**网络的稀疏性**, 并且减少了参数的相互依存关系, **缓解了过拟合**问题的发生。

缺点:

1. ReLU的输出**不是0均值的**。
2. **Dead ReLU Problem**
在训练的时很“脆弱”。

对任何数据有所响应，导致相应参数永远不会被更新。

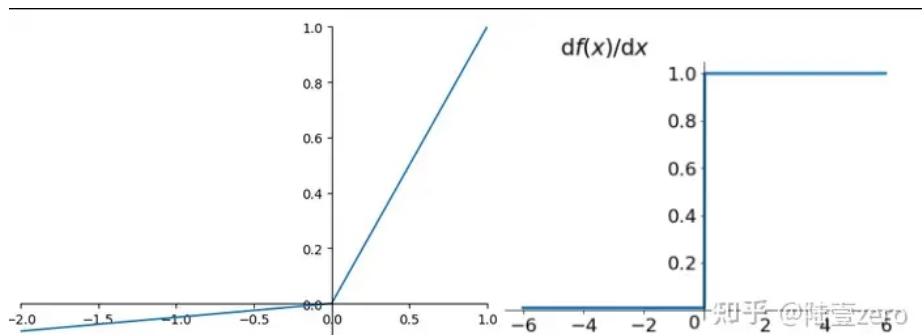
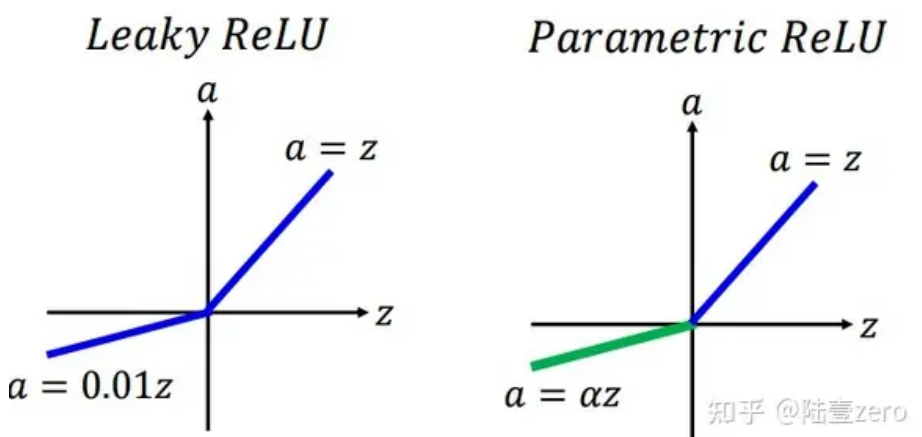
产生这种现象的两个原因：参数初始化问题；learning rate太高导致在训练过程中参数更新太大。

解决方法：采用Xavier初始化方法，以及避免将learning rate设置太大或使用adagrad等自动调节learning rate的算法。

4. Leaky ReLU激活函数

渗漏整流线性单元(Leaky ReLU)，为了解决dead ReLU现象。用一个类似0.01的小值来初始化神经元，从而使得ReLU在负数区域更偏向于激活而不是死掉。这里的斜率都是确定的。

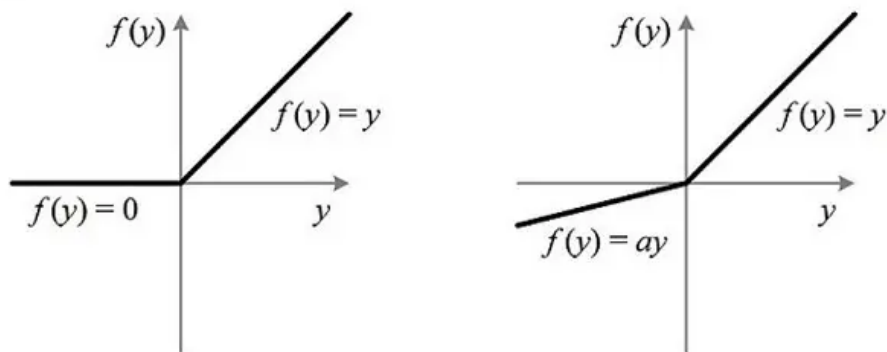
leakyrelu激活函数是relu的衍变版本，主要就是为了解决relu输出为0的问题。如图所示，在输入小于0时，虽然输出值很小但是值不为0。leakyrelu激活函数一个缺点就是它有些近似线性，导致在复杂分类中效果不好。



Leaky ReLU函数图像和求导图像

5. PReLU激活函数

PReLU(Parametric Rectified Linear Unit)，**参数化修正线性单元**(带参数的ReLU)，用来解决ReLU带来的神经元坏死的问题。二者的定义和区别如下图：



$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

$$PReLU(x_i) = \begin{cases} x_i & \text{if } x_i > 0 \\ a_i x_i & \text{if } x_i \leq 0 \end{cases}$$

i 表示不同的通道

如果 $a_i = 0$, 那么PReLU退化为ReLU; 如果 a_i 是一个很小的固定值(如 $a_i = 0.01$), 则PReLU退化为Leaky ReLU(LReLU)。有实验证明, 与ReLU相比, LReLU对最终的结果几乎没什么影响。

其中 a_i 不是固定的, 是通过反向传播学习出来的。

PReLU只增加了极少量的参数, 也就意味着网络的计算量以及过拟合的危险性都只增加了一点点。特别的, 当不同通道使用相同的 a_i 时, 参数就更少了。

BP更新 a_i 时, 采用的是带动量的更新方式, 如下图:

$$\Delta a_i := \mu \Delta a_i + \epsilon \frac{\partial \mathcal{E}}{\partial a_i}$$

上式的两个系数分别是 μ : 动量 和 ϵ : 学习率。

6.RReLU激活函数

lass="nolink">RReLU(Randomized Leaky ReLU)随机修正线性单元,RReLU也是Leaky ReLU的一个变体。RReLU训练时给定范围内随机选择 α , α 是从一个高斯分布 $U(l,u)$ 中随机出来的, 然后再测试过程中进行修正, 一般预估时固定为平均值。

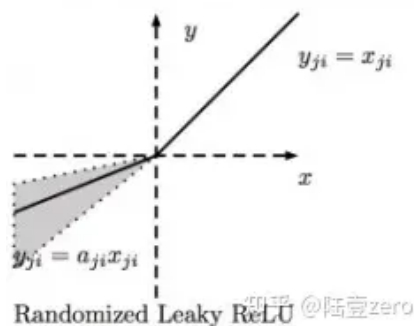
RReLU引入了随机因素, 可以减少过拟合的风险;

形式上来说, 我们能得到以下结果:

$$y_{ji} = \begin{cases} x_{ji} & \text{if } x_{ji} \geq 0 \\ a_{ji} x_{ji} & \text{if } x_{ji} < 0, \end{cases}$$

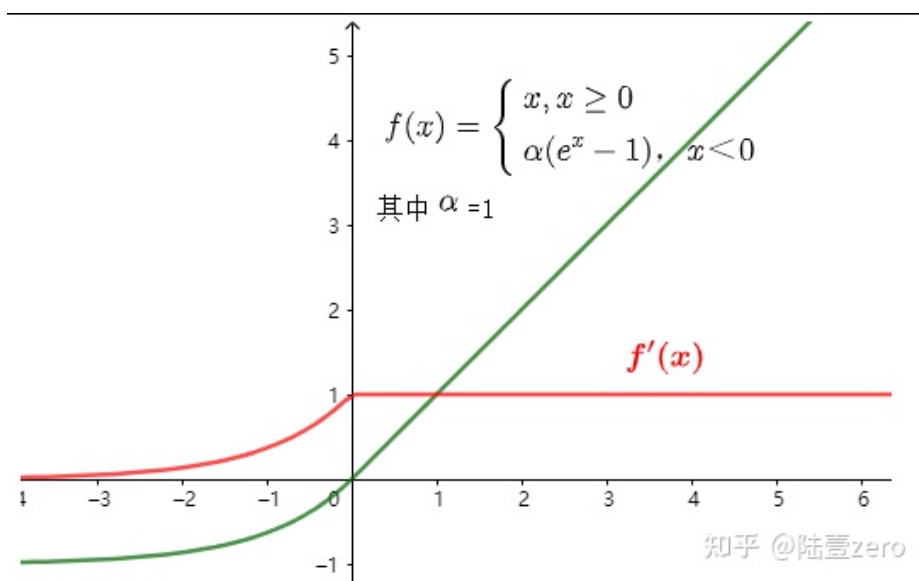
where

$$a_{ji} \sim U(l, u), l < u \text{ and } l, u \in [0, 1]$$



7. ELU激活函数

ELU (Exponential Linear Units) 指数线性单元: 具有relu的优势, 没有Dead ReLU问题, 输出均值接近0, 实际上PReLU和Leaky ReLU都有这一优点。有负数饱和区域, 从而对噪声有一些鲁棒性。可以看做是介于ReLU和Leaky ReLU之间的一个函数。当然, 这个函数也需要计算exp, 从而计算量上更大一些。



α 是一个可调整的参数, 它控制着ELU负值部分在何时饱和。右侧线性部分使得ELU能够缓解梯度消失, 而左侧软饱和能够让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

优点

1. 它在所有点上都是连续且可微的。
2. 与其他线性非饱和激活函数 (如 ReLU 及其变体) 相比, 它可以缩短训练时间。
3. 与 ReLU 不同, 它没有神经元死亡的问题。这是因为 ELU 的梯度对于所有负值都是非零的。
4. 作为非饱和激活函数, 它不会遭受梯度爆炸或消失的问题。
5. 与其他激活函数 (如 ReLU 和变体、Sigmoid 和双曲正切) 相比, 它实现了更高的准确度。

缺点

与 ReLU 及其变体相比, 它的计算速度较慢, 因为负输入涉及非线性。然而, 在训练期间, 这被 ELU 更快的收敛所补偿。但在测试期间, ELU 的执行速度会比 ReLU 及其变体慢。

理论上虽然好于ReLU, 但在实际使用中目前并没有好的证据ELU总是优于ReLU。

8. SELU激活函数

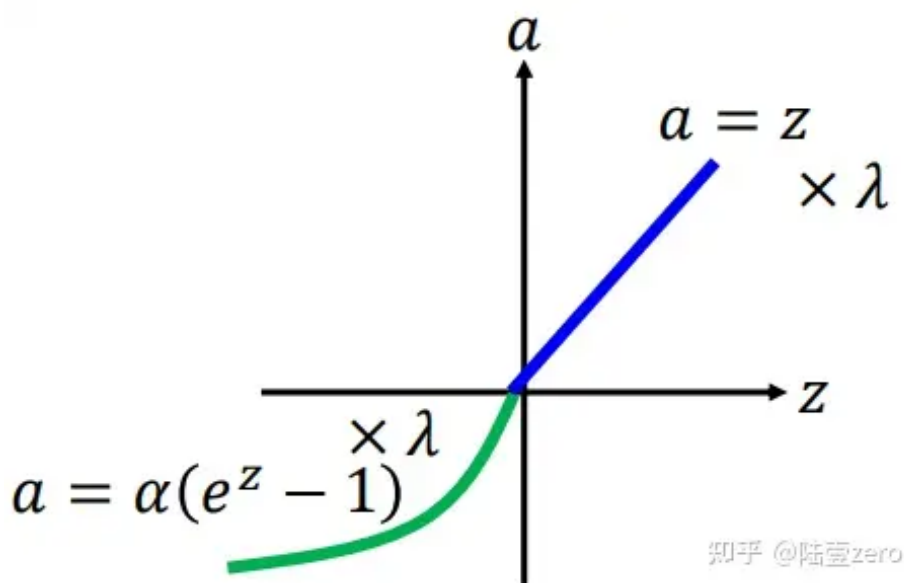
缩放指数线性单元(scaled exponential linear units, selu), 根据该激活函数得到的网络具有自归一化功能。

经过该激活函数后使得样本分布自动归一化到0均值和单位方差(自归一化, 保证训练过程中梯度不会爆炸或消失, 效果比**批归一化**Batch Normalization要好)

其实就是ELU乘了个 α 关键在于这个 α 是大于1的, 以前ReLU, PReLU, ELU这些激活函数, 都是在负半轴坡度平缓, 这样在激活函数的方差过大的时候可以让它减小, 防止了梯度爆炸, 但是正半轴坡度简单的设成了1。而SELU的正半轴大于1, 在方差过小的时候可以让它增大, 同时防止了梯度消失。这样激活函数就有一个不动点, 网络深了以后每一层的输出都是均值为0方差为1。

公式:

$$\text{SELU}(z) = \lambda \begin{cases} z & z > 0 \\ \alpha(\exp(z) - 1) & z \leq 0 \end{cases}$$



其中超参 α 和 λ 的值是证明得到的(而非训练学习得到):

$$\alpha = 1.6732632423543772848170429916717$$

$$\lambda = 1.0507009873554804934193349852946$$

即:

- 不存在死区
- 存在饱和区(负无穷时, 趋于 $-\alpha\lambda$)
- 输入大于零时, 激活输出对输入进行了放大

9.Maxout激活函数

maxout是一个函数逼近器, 对于一个标准的MLP(多层感知机)网络来说, 如果隐藏层的神经元足够多, 那么理论上我们是逼近任意的函数的。类似的, 对于maxout网络也是一个函数逼近器。

maxout函数相对于其他的激活函数有很大的区别, 它可以理解为是神经网络中的一层网络, 类似于池化层、卷积层, 可以看做是在神经网络中激活函数的地方加入一个激活函数层。maxout可以看做是一个可学习的分段线性函数, 因为可学习所以需要参数的, 而且参数是可以通过反向传播来学习的。

maxout做的事情就是第i层到第i+1层之前我们多训练一些参数, 对第i层的输出, 我们将每个输出连接k个隐藏层, 通过k个隐藏层的参数进行计算之后, 然后把这k个隐藏层求最大值, 作为最终的输出。

Maxout隐藏层每个神经元

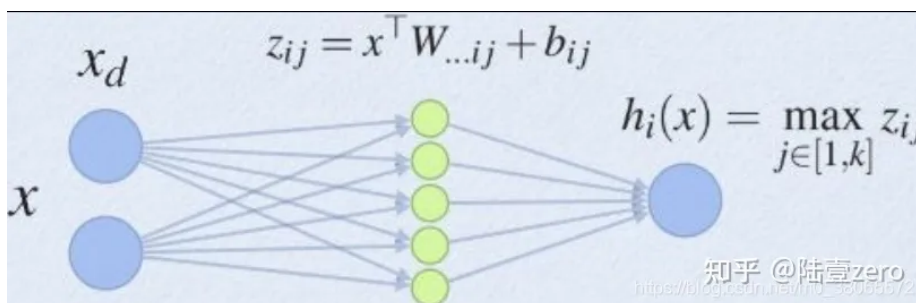
$$h_i(x) = \max_{j \in [1, k]} z_{ij}$$

上面的公式就是maxout隐藏层神经元i的计算公式。其中，k就是maxout层所需要的参数了，由我们人为设定大小。就像dropout一样，也有自己的参数p(每个神经元dropout概率)，maxout的参数是k。公式中Z的计算公式为：

$$z_{ij} = x^T W_{...ij} + b_{ij}$$

权重 \mathbf{w} 是一个大小为(d,m,k)三维矩阵(d代表的是输入节点的个数，m代表的则是输出层节点的个数，维度k就是为了产生中间的k个输出，然后将这k个输出再取一个最大值)， \mathbf{b} 是一个大小为(m,k)的二维矩阵，这两个就是我们需要学习的参数。如果我们设定参数k=1，那么这个时候，网络就类似于以前我们所学普通的MLP网络。

举个栗子，如下图所示，原本的神经网络只有输入的两个神经元x，输出的只有一个神经元 $h_i(x)$ ，普通的神经元则是直接把x和 $h_i(x)$ 连接，现在相当于在两层之间又增加了k=5个参数，然后输入先经过这个maxout的隐藏层得到输出，然后对输出取最大值，再输入到下一层当中。其中虚拟隐藏层的W和b就是所需要学习的参数。可以看到，参数量的增大是针对每一个输出神经元增加k个的。：



Maxout是通过分段线性函数来拟合所有可能的凸函数来作为激活函数的，但是由于线性函数是可学习，所以实际上是可以学出来的激活函数。具体操作是对所有线性取最大，也就是把若干直线的交点作为分段的边界，然后每一段取最大。

优点：

1. Maxout的拟合能力非常强，可以拟合任意的凸函数。
2. Maxout具有ReLU的所有优点，线性、不饱和性。
3. 不会出现神经元坏死的现象。

缺点： 增加了参数量。

10.Softmax激活函数

在多分类问题中，我们通常回使用softmax函数作为网络输出层的激活函数，softmax函数可以对输出值进行归一化操作，把所有输出值都转化为概率，所有概率值加起来等于1，softmax的公式为：

$$\text{Softmax}(x) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

Softmax函数是用于多类分类问题的激活函数，在多类分类问题中，超过两个类标签则需要类成员关系。对于长度为K的任意实向量，Softmax函数可以将其压缩为长度为K，值在[0,1]范围内，并且向量中元素的总和为1的实向量。

Softmax函数与正常的max函数不同：max函数仅输出最大值，但Softmax函数确保较小的值具有较小的概率，并且不会直接丢弃。我们可以认为它是argmax函数的概率版本或“soft”版本。Softmax函数的分母结合了原始输出值的所有因子，这意味着Softmax函数获得的各种概率彼此相关。

Softmax激活函数的特点：

- 在零点不可微。
- 负输入的梯度为零，这意味着对于该区域的激活，权重不会在反向传播期间更新，因此会产生永不激活的死亡神经元。

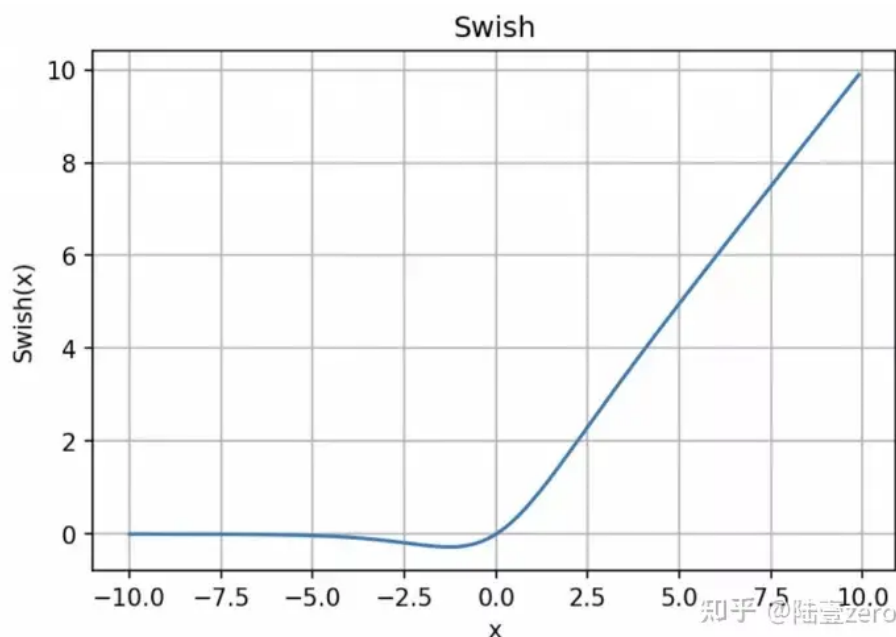
11.Swish激活函数

Swish函数表达式： $y = x * \text{sigmoid}(x)$

- Swish 的设计受到了 LSTM 和高速网络中 gating 的 sigmoid 函数使用的启发。我们使用相同的 gating 值来简化 gating 机制，这称为 self-gating。
- self-gating 的优点在于它只需要简单的标量输入，而普通的 gating 则需要多个标量输入。这使得诸如 Swish 之类的 self-gated 激活函数能够轻松替换以单个标量为输入的激活函数（例如 ReLU），而无需更改隐藏容量或参数数量。

优点：

- 「无界性」有助于防止慢速训练期间，梯度逐渐接近 0 并导致饱和；（同时，有界性也是有优势的，因为有界激活函数可以具有很强的正则化，并且较大的负输入问题也能解决）；
- 导数恒 > 0 ；
- 平滑度在优化和泛化中起了重要作用。



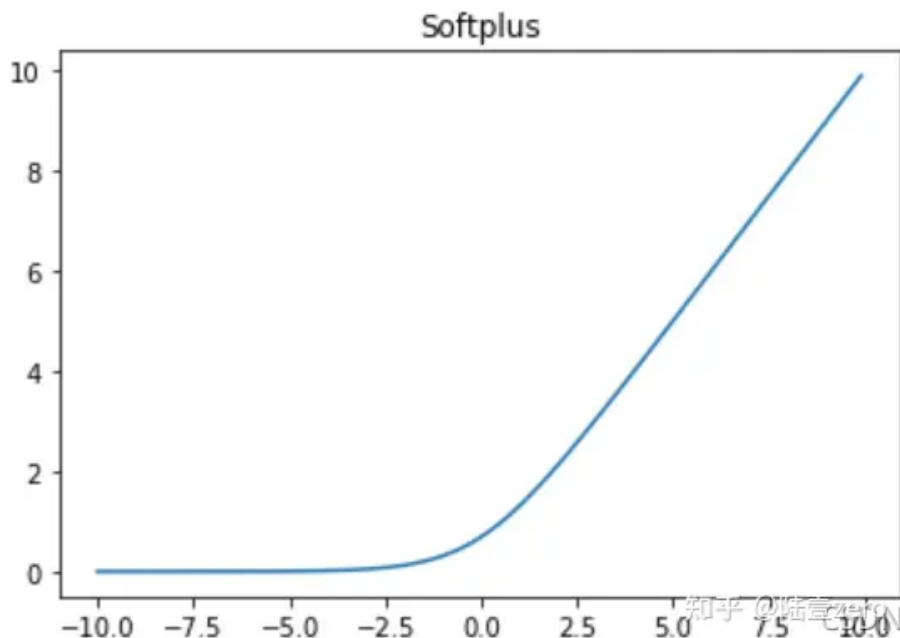
12.Softplus激活函数

Softplus函数可以看作是ReLU函数的平滑。根据神经科学家的相关研究，Softplus函数和ReLU函数与脑神经元激活频率函数有相似的地方。也就是说，相比于早期的激活函数，Softplus函数和ReLU函数更加接近脑神经元的激活模型，而神经网络正是基于脑神经科学发展而来，这两个激活函数的应用促成了神经网络研究的新浪潮。

公式：

$$\text{Softplus}(x) = \log(1 + e^x)$$

图像：



五、知识点补充学习

1.关于传入神经网络层的数据分布与特性

不同于之前的机器学习方法，神经网络并不依赖关于输入数据的任何概率学或统计学假定。然而，为了确保神经网络学习良好，最重要的因素之一是传入神经网络层的数据需要具有特定的性质。

1. 数据分布应该是**零中心化 (zero centered)** 的，也就是说，分布的均值应该在零附近。不具有这一性质的数据可能导致**梯度消失**和训练抖动。
2. 分布最好是**正态**的，否则可能导致网络**过拟合**输入空间的某个区域。
3. 在训练过程中，**不同batch和不同网络层的激活分布，应该保持一定程度上的一致**。如果不具备这一性质，那么我们说分布出现了**内部协方差偏移 (Internal Covariate shift)**，这可能拖慢训练进程。

2.梯度消失和梯度爆炸以及解决方法

- 梯度消失 (Vanishing Gradients)
在梯度下降中，随着算法反向的反馈，梯度会越来越小，最终没有变化，此时并没有收敛到比较好的解，这就是梯度消失的问题。
- 梯度爆炸
梯度爆炸原理跟梯度消失一样，反向传播时，导数大于1，导致梯度增加。

解决方法

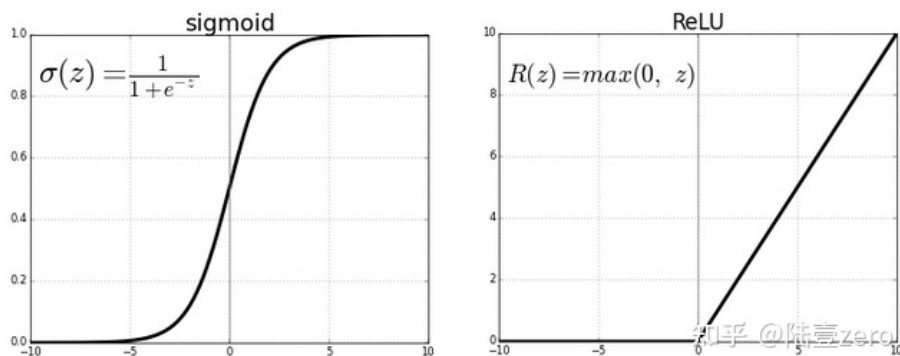
- 激活函数
- 更换激活函数：ELU > leaky ReLU > ReLU > tanh > logistic
- 添加BN(Batch Normalization)层(批归一化);作用:1.允许较大的学习率 2.减弱对初始化的强依赖性 3.保持隐藏层中数值的均值、方差不变，让数值更稳定，为后面网络提供坚实的基础 4.有轻微的正则化作用（相当于给隐藏层加入噪声，类似Dropout）
- ResNet残差网络
- LSTM结构
- 梯度裁剪；主要针对梯度爆炸问题，设置一个阈值，当梯度查过这个阈值之后将它限制在这个范围之内
- 权重正则化，L1和L2正则化；L1正则化可以产生稀疏权值矩阵，即产生一个稀疏模型，可以用于特征选择，一定程度上，L1也可以防止过拟合L2正则化可以防止模型过拟合（overfitting）

3.饱和神经元(Saturated Neurons)

饱和神经元会导致梯度消失问题进一步恶化。假设，传入带sigmoid激活的神经元的激活前数值 $\omega^T x + b$ 非常高或非常低。那么，由于sigmoid在两端处的梯度几乎是0，任何梯度更新基本上都无法导致权重 ω 和偏置 b 发生变化，神经元的权重变动需要很多步才会发生。也就是说，即使梯度原本不低，由于饱和神经元的存在，最终梯度仍会趋向于零。

4.ReLU"救星"(ReLU可以解决梯度消失问题)

在普通深度网络设定下，ReLU激活函数的引入是缓解梯度消失问题的首个尝试（LSTM的引入也是为了应对这一问题，不过它的应用场景是循环模型RNN）。



当 $x > 0$ 时，ReLU的梯度为1， $x < 0$ 时，ReLU的梯度为0。这带来了一些好处。ReLU函数梯度乘积并不收敛于0，因为ReLU的梯度要么是0，要么是1。当梯度值为1时，梯度原封不动地反向传播。当梯度值为0时，从这点往后不会进行反向传播。

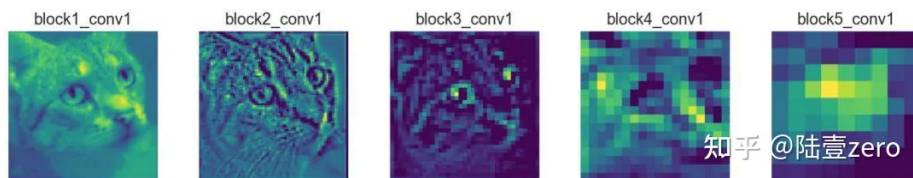
5.单边饱和

sigmoid函数是双边饱和的，也就是说，正向和负向都趋向于零。ReLU则提供单边饱和。

准确地说，ReLU的左半部分不叫饱和，饱和的情况下，函数值变动极小，而ReLU的左半部分根本不变。但两者的作用是类似的。你也许会问，单边饱和带来了什么好处？

我们可以把深度网络中的神经元看成开关，这些开关专门负责检测特定特征。这些特征常常被称为**概念**。高层网络中的神经元也许最终会专门检测眼睛、轮胎之类的高层特征，而低层网络中的神经元最终专门检测曲线、边缘之类的低层特征。

当这样的概念存在于神经网络的输入时，我们想要激活相应的神经元，而**激活的数量级则可以测量概念的程度**。例如，如果神经元检测到了边缘，它的数量级也许表示边缘的锐利程度。



然而，神经元的负值在这里就没什么意义了。用负值编码不存在的概念的程度感觉怪怪的。

以检测边缘的神经元为例，**相比激活值为5的神经元，激活值为10的神经元可能检测到了更锐利的边缘。但是区分激活值-5和-10的神经元就没什么意义了，因为负值表示根本不存在边缘。**因此，统一用零表示**概念不存在**是很方便的。ReLU的单边饱和正符合这一点。

6.信息解缠和对噪声的鲁棒性

单边饱和提高了神经元对噪声的鲁棒性。为什么？假设神经元的值是无界的，也就是在两个方向上都不饱和。**具有程度不同的概念的输入产生神经元正值输出的不同。**由于我们想要用数量级指示信号的强度，这很好。

然而，背景噪声、神经元不擅长检测的概念（例如，包含弧线的区域传入检测线条的神经元），会生成不同的神经元负值输出。**这类不同可能给其他神经元带去大量无关、无用信息。**这也可能导致单元间的相关性。例如，检测线条的神经元也许和检测弧线的神经元负相关。

而在神经元单边饱和（负向）的场景下，**噪声等造成的不同，也就是之前的负值输出数量级的不同，被激活函数的饱和元素挤压为零**，从而防止噪声产生无关信号。

7.稀疏性

ReLU函数在算力上也有优势。基于ReLU的网络训练起来比较快，因为计算ReLU激活的梯度不怎么需要算力，而sigmoid梯度计算就需要指数运算。

ReLU归零激活前的负值，这就隐式地给网络引入了稀疏性，同样节省了算力。

8.死亡ReLU问题

ReLU也有缺陷。虽然稀疏性在算力上有优势，但过多的稀疏性实际上会阻碍学习。激活前神经元通常也包含偏置项，如果偏置项是一个过小的负数，使得 $\omega^T x + b < 0$ ，那么ReLU激活在反向传播中的梯度就是0，使负的激活前神经元无法更新。

如果学习到的权重和偏置使整个输入域上的激活前数值都是负数，那么神经元就无法学习，引起类似sigmoid的饱和现象。这称为**死亡ReLU问题**。

9.零中心化激活

不管输入是什么，ReLU只输出非负激活。这可能是一个劣势。

对于基于ReLU的神经网络而言，网络层ln的权重 ω_n 的激活为

$$z_n = \text{ReLU}(\omega_n^T x_n + b_n)$$

因此，对损失函数L而言：

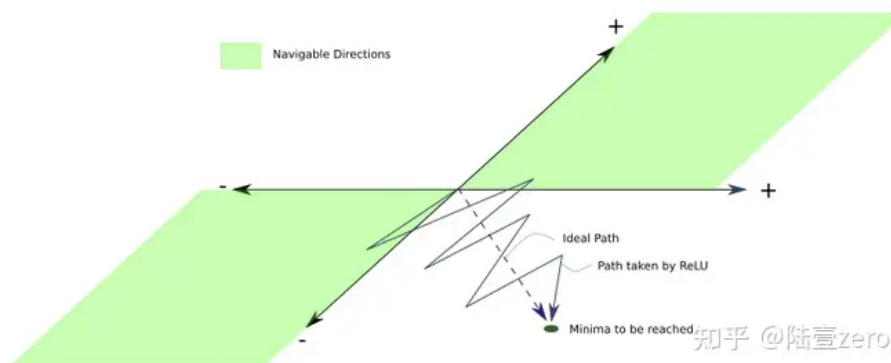
$$\frac{\partial L}{\partial \omega_n} = \frac{\partial L}{\partial (\text{ReLU}(\omega_n^T x_n + b_n))} * I(\text{ReLU}(\omega_n^T x_n + b_n)) * x_n$$

上式中的I是一个指示函数，传入的ReLU值为正数时输出1，否则输出0。由于ReLU只输出非负值， ω_n 中的每项权重的梯度更新正负都一样。

这有什么问题？问题在于，由于**所有神经元的梯度更新的符号都一样，网络层ln中的所有权重在一次更新中，要么全部增加，要么全部减少**。然而，理想情况的梯度权重更新也许是某些权重增加，另一些权重减少。ReLU下，这做不到。

假设，根据理想的权重更新，有些权重需要减少。然而，如果梯度更新是正值，这些权重可能在当前迭代中变为过大的正值。下一次迭代，梯度可能会变成较小的负值以补偿这些增加的权重，这也许会导致最终跳过需要少量负值或正值变动才能取到的权重。

这可能导致搜寻最小值时出现之字模式，拖慢训练速度。



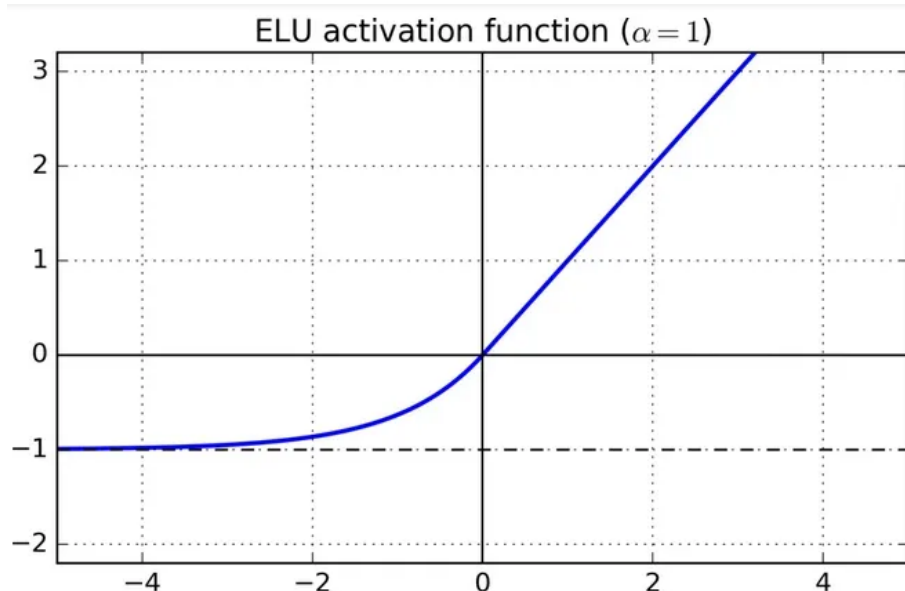
10.指数线性单元和偏置偏移

一个完美的激活函数应该同时具备以下两个性质：

1. 产生零中心化分布，以加速训练过程。
2. 具有单边饱和，以导向更好的收敛。

Leaky ReLU和PReLU（参数化ReLU）满足第一个条件，不满足第二个条件。而原始的ReLU满足第二个条件，不满足第一个条件。

同时满足两个条件的一个激活函数是指数线性单元（ELU）。



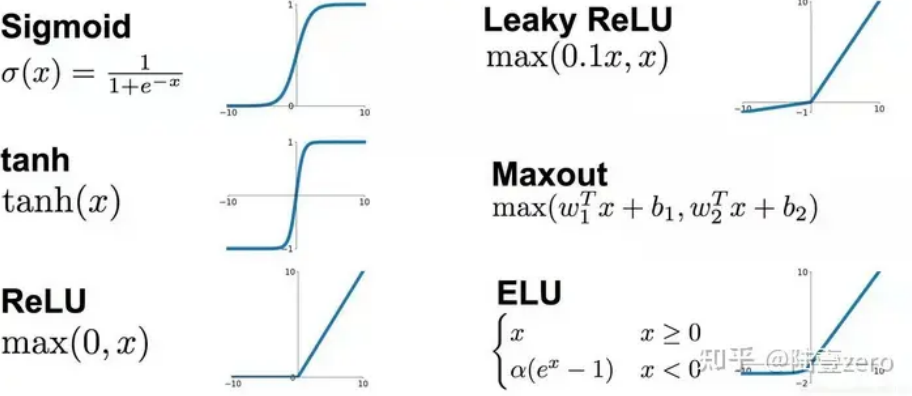
$$f(x) = \begin{cases} x & \text{when } x > 0 \\ \alpha(e^x - 1) & \text{when } x \leq 0 \end{cases}$$

$x > 0$ 部分，ELU的梯度是1， $x < 0$ 部分的梯度则是 $\alpha \times e^x$ 。ELU激活函数的负值区域趋向于 $-\alpha$ 。 α 是一个超参数，通常取1。

11.常用激活函数分类

激活函数可以分为两大类：

- 饱和激活函数：sigmoid、tanh
- 非饱和激活函数：ReLU、Leaky Relu、ELU【指数线性单元】、PReLU【参数化的ReLU】、RReLU【随机ReLU】



相对于饱和激活函数，使用“非饱和激活函数”的优势在于两点：

- 首先，“非饱和激活函数”能解决深度神经网络【**层数非常多**】的“**梯度消失**”问题，浅层网络【**三五层那种**】才用sigmoid 作为激活函数。
- 其次，它能**加快收敛速度**。

12.如何选择激活函数

- 首先尝试ReLU激活。尽管我们上面列出了ReLU的一些问题，但很多人使用ReLU取得了很好的结果。根据奥卡姆剃刀原则，先尝试更简单的方案比较好。**相比ReLU的有力挑战者，ReLU的算力负担最轻**。如果你的项目需要从头开始编程，那么ReLU的实现也特别简单。
- 如果ReLU的效果不好，我会接着尝试Leaky ReLU或ELU。我发现能够产生零中心化激活的函数一般要比不能做到这点的函数效果好得多。ELU看起来很有吸引力，但是由于负的激活前值会触发大量指数运算，基于ELU的网络训练和推理都很缓慢。**如果算力资源对你而言不成问题，或者网络不是特别巨大，选择ELU，否则，选择Leaky ReLU**。LReLU和ELU都增加了一个需要调整的超参数。
- 如果算力资源很充沛，时间很充裕，你可以将上述激活函数的表现与PReLU和随机ReLU做下对比。**如果出现了过拟合，那么随机ReLU可能会有用**。参数化ReLU加入了需要学习的一组参数，所以，**只在具备大量训练数据的情况下才考虑选用参数化ReLU**。

六、参考资料

[kevin：一文详解激活函数](#)

[深度学习（二十三）Maxout网络学习_hjimce的博客-CSDN博客](#)

[factor v leiden,selu激活函数](#)

[深度学习中的激活函数 - 走看看](#)

[机器学习中的数学--激活函数（七）： Softmax函数_von Neumann的博客-CSDN博客](#)

[将为帅：一文搞懂激活函数\(Sigmoid/ReLU/LeakyReLU/PReLU/ELU\)](#)

[深度学习优化算法入门：三、梯度消失和激活函数](#)

编辑于 2022-12-05 11:50 · IP 属地江西