Deep Learning Technology and Application

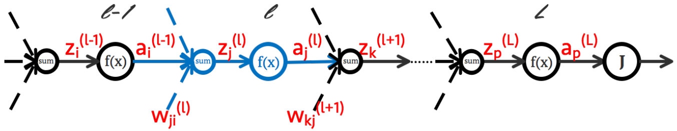
Ge Li

Peking University

Table of contents

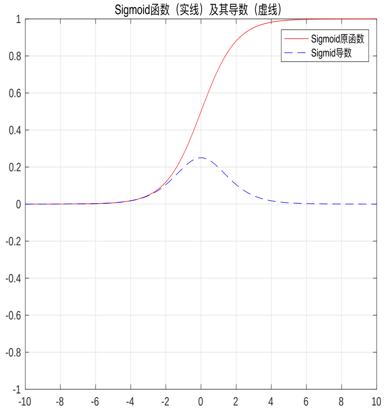
关于激活函数

反向传播核心算式



$$
\begin{aligned}
w_{ji}^{(l)} &= w_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial w_{ji}^{(l)}} \\
&= w_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} = w_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial z_j^{(l)}} a_i^{(l-1)} \\
&= w_{ji}^{(l)} - \alpha \delta_j^{(l)} a_i^{(l-1)} \\
&= w_{ji}^{(l)} - \alpha \left(\sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)}) \right) a_i^{(l-1)}
\end{aligned}
$$

Sigmoid



$$\delta(z) = \frac{1}{1+exp(-z)} \in (0,1)$$

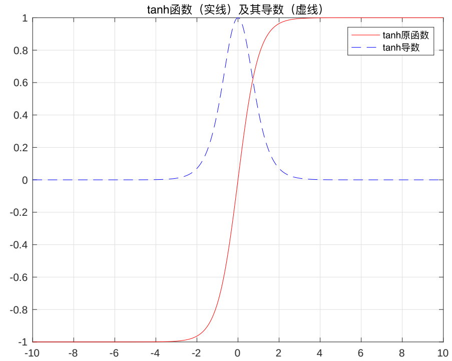
$$\begin{aligned}\delta'(z) &= \frac{-exp(-z)}{(1+exp(-z))^2} \\ &= \delta(z)(1-\delta(z))\end{aligned}$$

Sigmoid

- 优点：
    - 非常符合“激活”的含义，值从 0 到 1
    - 导数那时还算优美
- 缺点：
    - 梯度“杀手”，观察其导数曲线，当激活值“稍大一些”或“稍小一些”的时候，导数值都会逼近 0，从而导致在反向传播中与 $\nabla J(\theta)$ 相乘时，杀掉 $\nabla J(\theta)$；

      所以，使用 Sigmoid 函数时，一定要注意神经网络参数的初始化，如果参数比较大或比较小，会“帮助”提高 Sigmoid 杀伤力。
    - 降低了梯度调整的灵敏性，由于其激活值并不以 0 点为中心，而是 0 到 1，所以该激活值不会改变梯度调整的方向，梯度调整的方向完全由 $\nabla J(\theta)$ 决定，这回增加产生 Zig-zagging 问题的几率。

Tanh



$$
\begin{aligned}
tanh(z) &= \frac{exp(z) - exp(-z)}{exp(z) + exp(-x)} \\
&= 2\delta(2z) - 1
\end{aligned}
$$

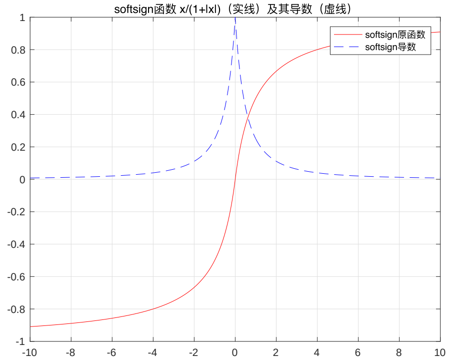
$$tanh(z) \in (-1, 1)$$

$$
\begin{aligned}
tanh'(z) &= 1 - (\frac{exp(z) - exp(-z)}{exp(z) + exp(-z)})^2 \\
&= 1 - tanh^2(z)
\end{aligned}
$$

Sigmoid

- 优点：
  - 没有非 0 点问题；
  - 输入在 0 点附近时，Gradient 较大，会对权重进行较大的调整；
- 缺点：
  - 仍然是梯度“杀手”

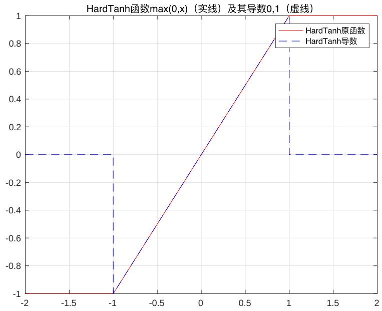
Softsign



$$softsign(z) = \frac{z}{1 + |z|}$$

$$softsign'(z) = \frac{sgn(z)}{(1 + z)^2}$$

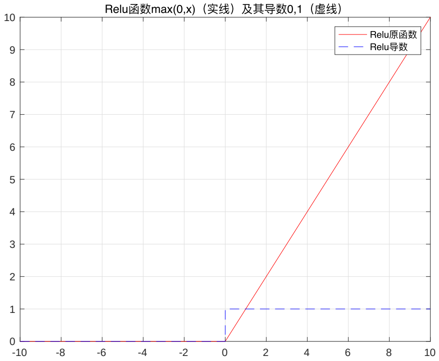
Hard Tanh



$$hardtanh(z) = \begin{cases} -1 & : z < -1 \\ z & : otherwise \\ 1 & : z > 1 \end{cases}$$

$$hardtanh'(z) = \begin{cases} 1 & : -1 \geq z \leq 1 \\ 0 & : otherwise \end{cases}$$

ReLU - Rectified Linear Unit



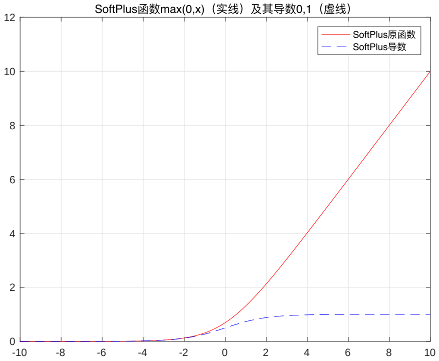
$$relu(z) = \begin{cases} 0 & : z < 0 \\ z & : z \geq 0 \end{cases}$$

$$relu'(z) = \begin{cases} 1 & : z > 0 \\ 0 & : otherwise \end{cases}$$

ReLU

- 优点：
  - 导数更简单，计算复杂度低；相比于之前的其他激活函数，ReLU 更加容易实现，并易于扩展；
  - 使用 ReLU 的神经网络更加容易收敛，其原因之一得益于 ReLU 没有最大值上届，没有饱和区；
- 缺点：
  - 会造成“死”神经元：由于 ReLU 函数没有上限，在权重调整过程中，可能出现“单次的大尺度调整”，而在参数经过该次“大尺度调整”之后，再也无法形成有效的激活，从而形成“死神经元”。
  - 如果 Learning Rate 恰好设置得比较大，有可能造成 40% 以上的“死神经元”，适当减小 Learning Rate 能够在一定程度上减少“死神经元”。

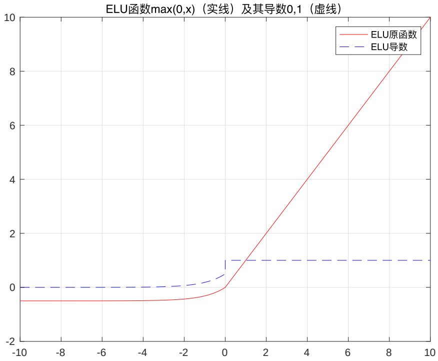
SoftPlus



$$softplus(z) = \log_e(1 + e^z))$$

$$softplus'(z) = \frac{1}{1 + e^{(-z)}}$$

ELU - Exponential Linear Unit

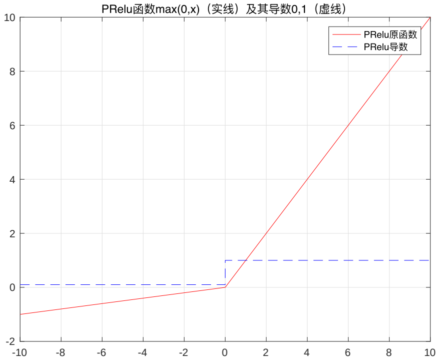


$$elu(z) = \begin{cases} \alpha(e^z - 1) & : z < 0 \\ z & : z \geq 0 \end{cases}$$

$$where\ 0 < \alpha < 1$$

$$elu'(z) = \begin{cases} elu(z) + \alpha & : z < 0 \\ 1 & : z >= 0 \end{cases}$$

PReLU - Parametric ReLU



$$prelu(z) = \begin{cases} \alpha z & : z < 0 \\ z & : z \geq 0 \end{cases}$$

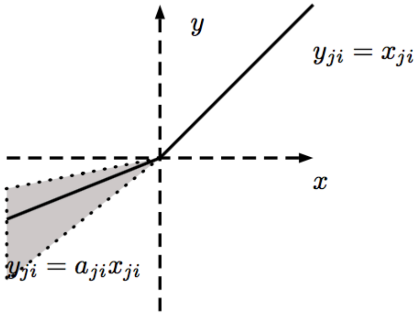
$$where\ 0 < \alpha < 1$$

$$prelu'(z) = \begin{cases} 1 & : z > 0 \\ \alpha & : otherwise \end{cases}$$

Learky ReLU PReLU

- 为了减低 ReLU 导致“死神经元”的几率，研究者提出了 Leaky ReLU 方法；
- 该方法将 $x < 0$ 情况下的输出值，修改为 αx；
- Leaky ReLU 中，α 是一个固定值，是 PReLU 的一种特殊情况。
- 遗憾的是，PReLU 方法的效果，并不稳定，在不同实验中表现不同。
- 因此，有的学者提出了 Randomized Leaky ReLU 方法。

Randomized Leaky ReLU



Randomized Leaky ReLU

$$f(x_{ji} = x_{ji}), if\ x_{ji} \geq 0$$
$$f(x_{ji}) = \alpha_{ji}x_{ji}, if\ x_{ji} < 0$$
$$\alpha_{ji} \sim U(l, u),\ l < u,\ l, u \in [0, 1)$$

训练阶段：α_{ji} 取 l 与 u 之间的随机数
测试阶段：α_{ji} 取平均数 $\frac{l+u}{2}$

Advantages vs. Problems of ReLUs

Generally, we summarize the advantages and potential problems of ReLUs:

- Positives：
  - Biological plausibility: One-sided, compared to the antisymmetry of tanh.
  - Sparse activation: For example, in a randomly initialized network, only about 50% of hidden units are activated (having a non-zero output).
  - Efficient gradient propagation: No vanishing or exploding gradient problems.
  - Efficient computation: Only comparison, addition and multiplication.
  - Scale-invariant: $max(0, \alpha x) = \alpha \dot{m}ax(0, x)$

Advantages vs. Problems of ReLUs

Generally, we summarize the advantages and potential problems of ReLUs:

- Negtives:
    - Non-differentiable at zero: however it is differentiable anywhere else, including points arbitrarily close to (but not equal to) zero.
    - Non-zero centered.
    - Unbounded: Could potentially blow up.
    - Dying Relu problem: Relu neurons can sometimes be pushed into states in which they become inactive for essentially all inputs. In this state, no gradients flow backward through the neuron, and so the neuron becomes stuck in a perpetually inactive state and "dies." In some cases, large numbers of neurons in a network can become stuck in dead states, effectively decreasing the model capacity. This problem typically arises when the learning rate is set too high.

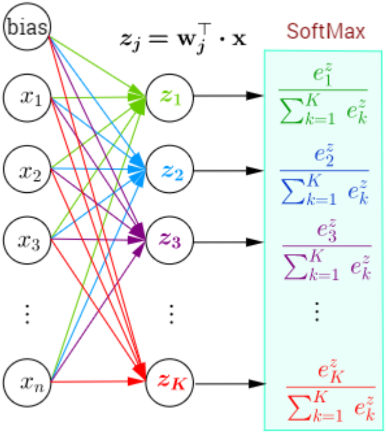
Maxout

使用类似于 ReLUs 的一类函数，例如：

$$max(w_1^T x + b_1, w_2^T x + b_2)$$

- Notice that both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have $(w_1, b_1 = 0)$
- The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU). However, unlike the ReLU neurons it doubles the number of parameters for every single neuron, leading to a high total number of parameters.

SoftMax


bias
x_1
x_2
x_3
x_n
$z_j = \mathbf{w}_j^\top \cdot \mathbf{x}$
z_1
z_2
z_3
z_K
SoftMax
$\frac{e^{z}_1}{\sum_{k=1}^K e^z_k}$
$\frac{e^{z}_2}{\sum_{k=1}^K e^z_k}$
$\frac{e^{z}_3}{\sum_{k=1}^K e^z_k}$
$\frac{e^{z}_K}{\sum_{k=1}^K e^z_k}$

- The input to the function is the result of K distinct linear functions, and the predicted probability for the j^{th} class given a sample vector x and a weighting vector w is:

$$P(y=j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^K e^{x^T w_k}}$$

- This can be seen as the composition of K linear functions $x \mapsto x^T w_1, \ldots, x \mapsto x^T w_K$ and the softmax function(where $x^T w$ denotes the inner product of x and w). The operation is equivalent to applying a linear operator defined by w to vectors x, thus transforming the original, probably highly-dimensional, input to vectors in a K-dimentional space $\mathcal{R}^K$.

回顾: 基于多项分布的 Loss Function

若可假设网络输出满足如下分布:

$$\begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)} \begin{bmatrix} f(x^{(i)};\theta_1) \\ f(x^{(i)};\theta_2) \\ \vdots \\ f(x^{(i)};\theta_k) \end{bmatrix}$$

定义函数:

$$1\{\text{表达式为真}\} = 1$$

回顾: 基于多项分布的 Loss Function

则，似然函数为：

$$\begin{aligned} L(\theta) &= p(Y|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \left(\sum_{j=1}^{k} 1\{y^{(i)} = j\} \frac{f(x^{(i)};\theta_j)}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)} \right) \end{aligned}$$

进行 log 处理，得到需要最大化的 Loss Function 为：

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} \left(\sum_{j=1}^{k} \left(1\{y^{(i)} = j\}\right) \log \frac{f(x^{(i)};\theta_j)}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)} \right) \end{aligned}$$

回顾: 基于多项分布的 Loss Function

等同于最小化:

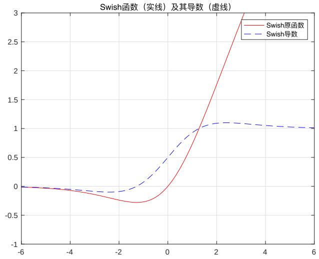
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{k}\left(1\{y^{(i)} = j\}\right)\log\frac{f(x^{(i)};\theta_j)}{\sum_{j=1}^{k} f(x^{(i)};\theta_j)}\right)$$

为便于计算，通常取 $f(x^{(i)};\theta_j) = exp(\theta_j^T x^{(i)})$，得:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{k}\left(1\{y^{(i)} = j\}\right)\log\frac{exp(\theta_j^T x^{(i)})}{\sum_{j=1}^{k} exp(\theta_j^T x^{(i)})}\right)$$

SoftMax

Swish



$$\delta(z) = \frac{x}{1+exp(-z)}$$

$$\delta'(z) = \frac{x}{1+exp(-x)} + \frac{1}{(1+exp(-x))}(1-\frac{x}{1+exp(-x)});$$

Thanks.