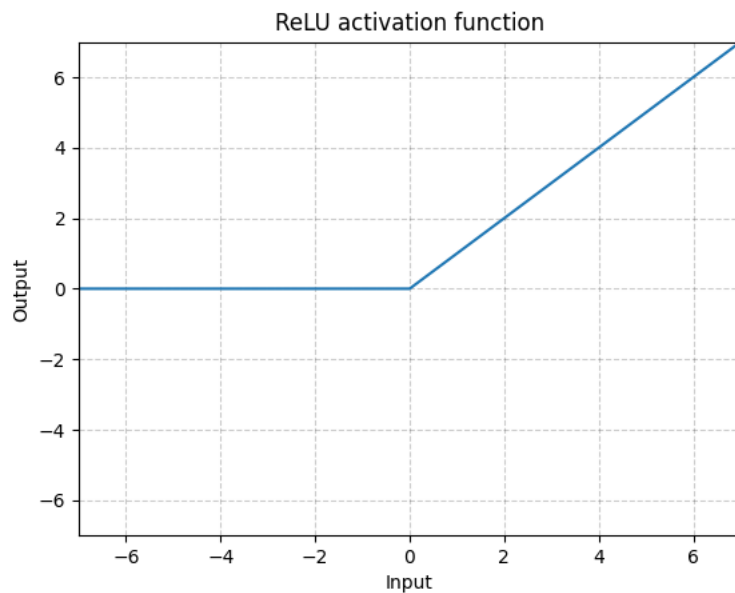


Activation Functions

In PyTorch & how to use them

ReLU – `nn.ReLU()`

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$



RReLU – nn.ReLU()

$$\text{RReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{otherwise} \end{cases}$$

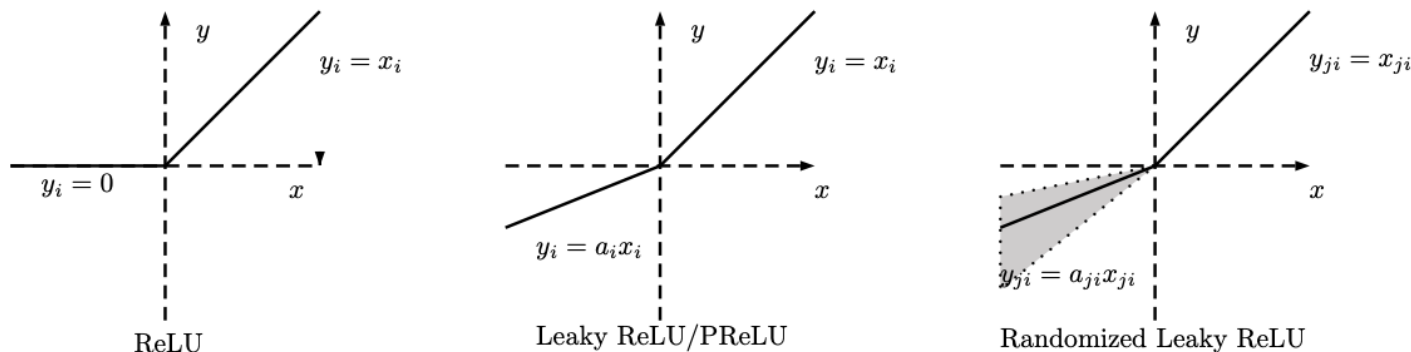
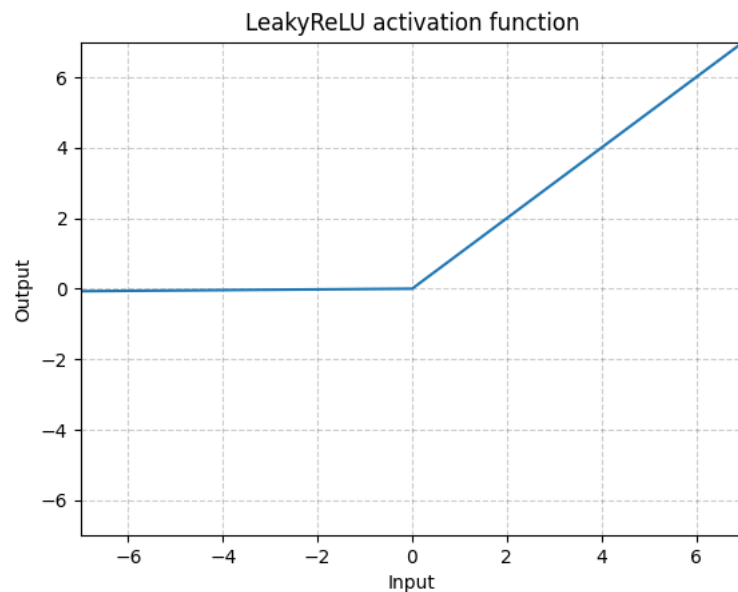


Figure 1: ReLU, Leaky ReLU, PReLU and RReLU. For PReLU, a_i is learned and for Leaky ReLU a_i is fixed. For RReLU, a_{ji} is a random variable keeps sampling in a given range, and remains fixed in testing.

LeakyReLU – `nn.LeakyReLU()`

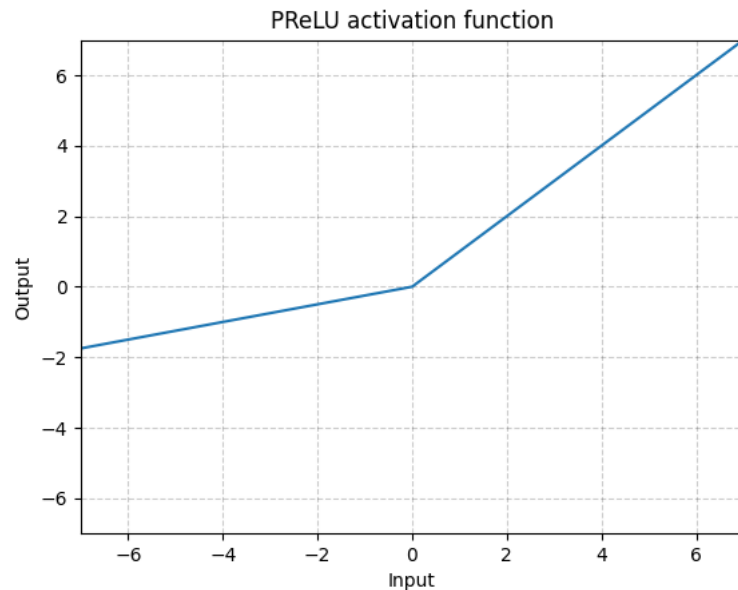
$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative_slope} \times x, & \text{otherwise} \end{cases}$$



PReLU – `nn.PReLU()`

$$\text{PReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$$

Here a is a learnable parameter. When called without arguments, `nn.PReLU()` uses a single parameter a across all input channels. If called with `nn.PReLU(nChannels)`, a separate a is used for each input channel.

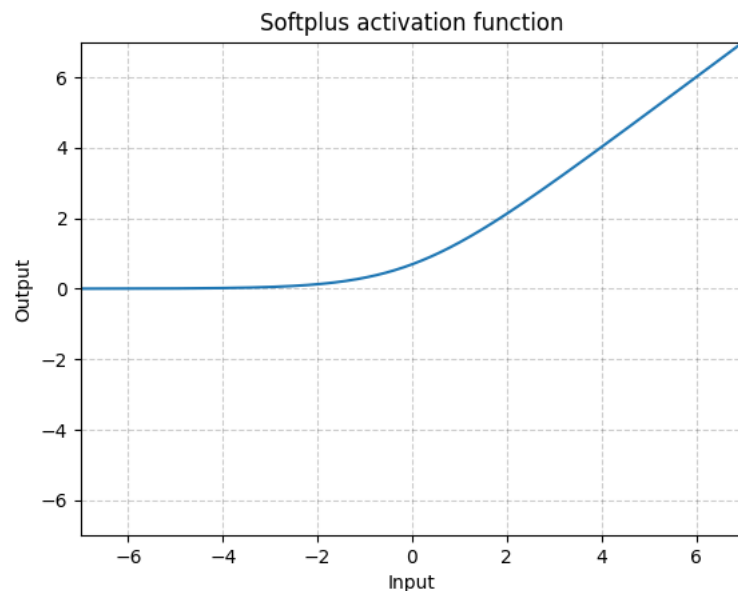


Softplus – `nn.Softplus()`

$$\text{Softplus}(x) = \frac{1}{\beta} * \log(1 + \exp(\beta * x))$$

SoftPlus is a smooth approximation to the ReLU function and can be used to constrain the output of a machine to always be positive.

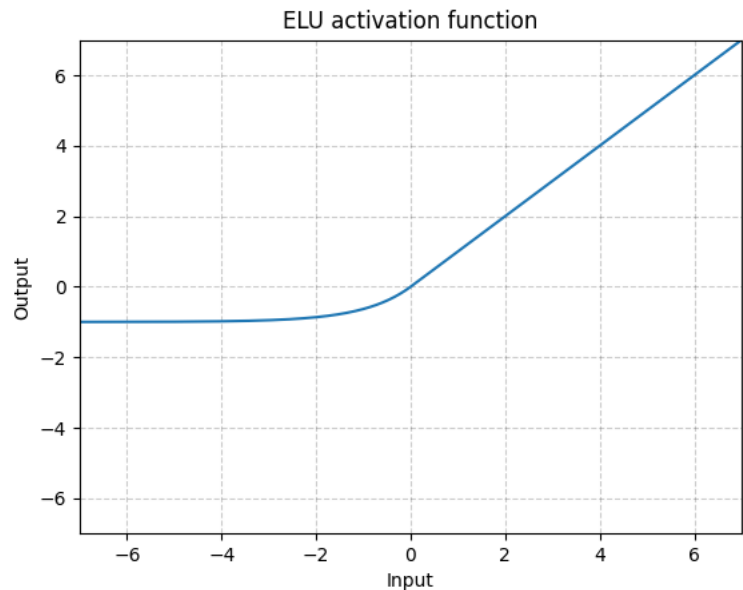
For numerical stability the implementation reverts to the linear function when $input \times \beta > threshold$.



ELU – `nn.ELU()`

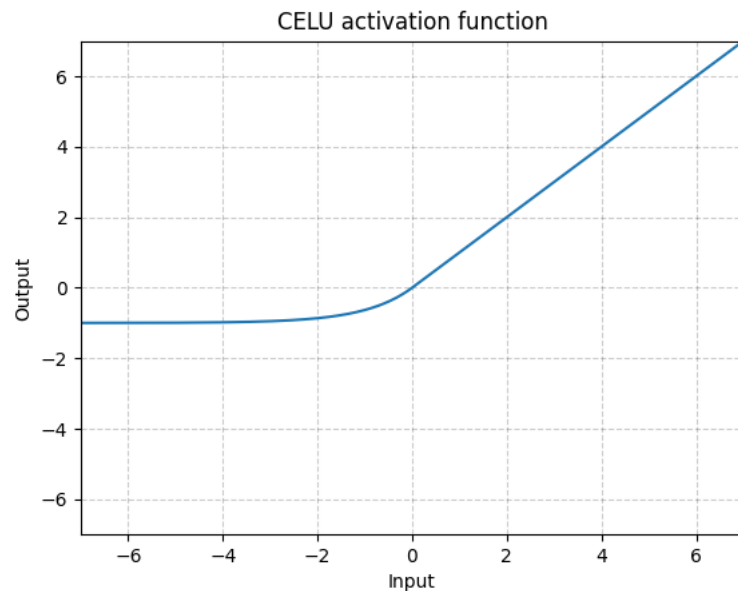
Applies the element-wise function:

$$\text{ELU}(x) = \max(0, x) + \min(0, \alpha * (\exp(x) - 1))$$



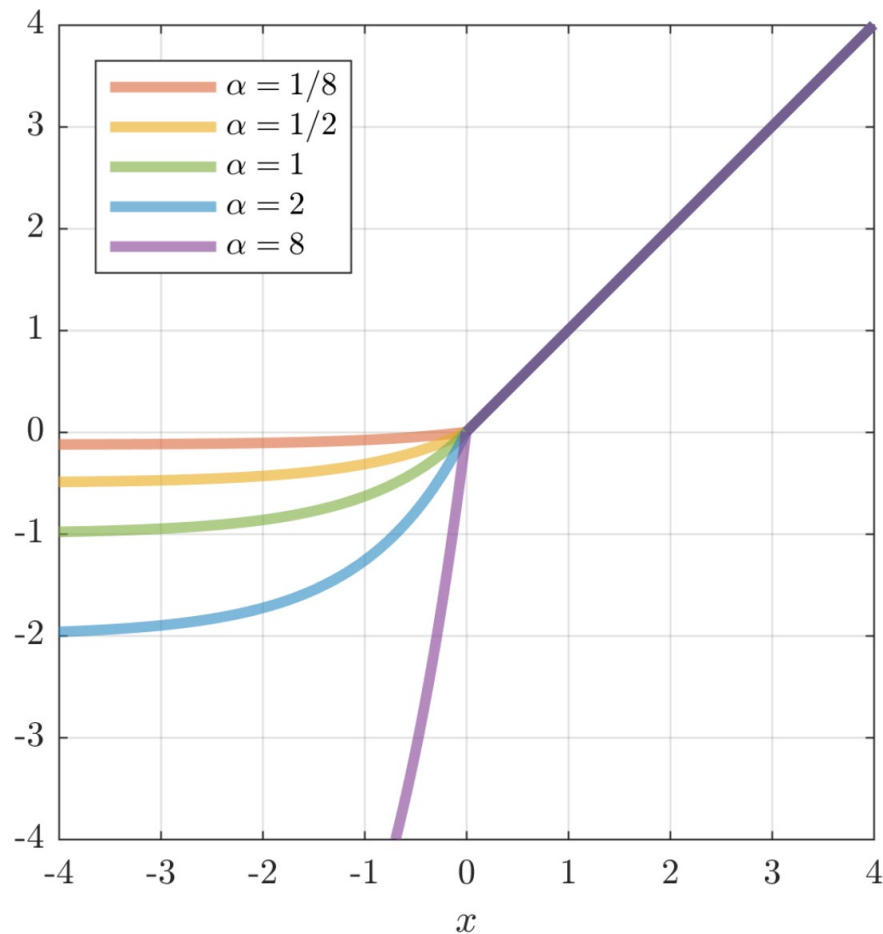
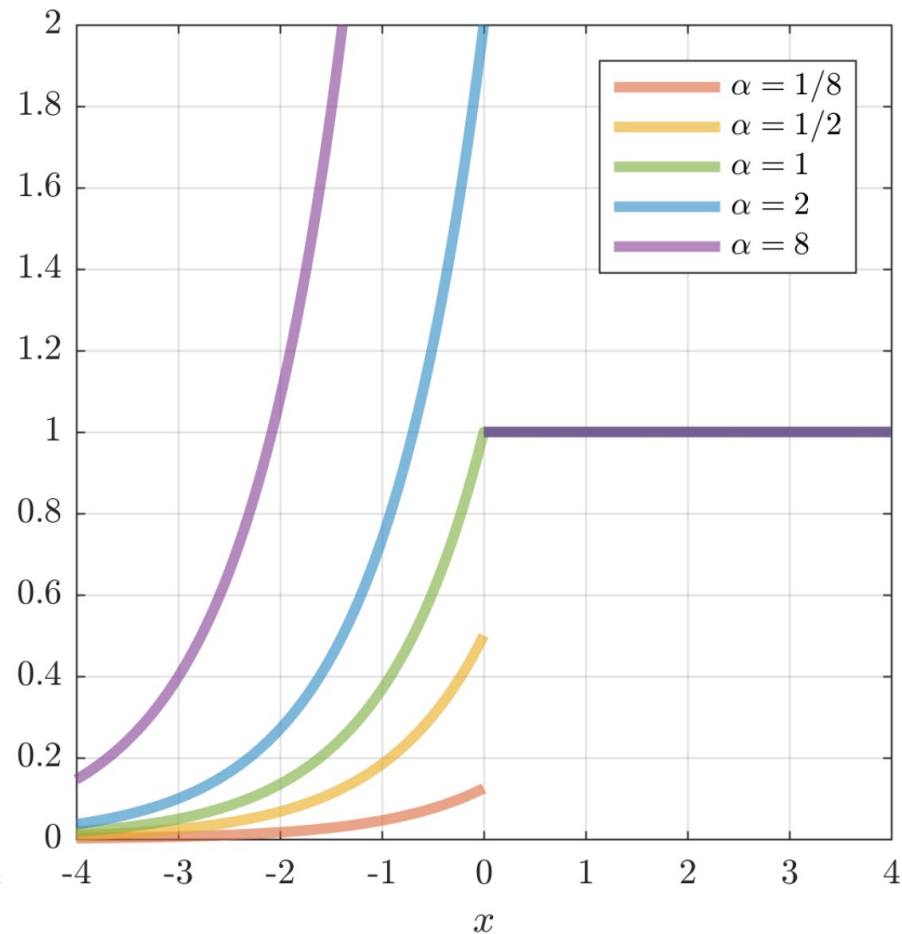
CELU – `nn.CELU()`

$$\text{CELU}(x) = \max(0, x) + \min(0, \alpha * (\exp(x/\alpha) - 1))$$

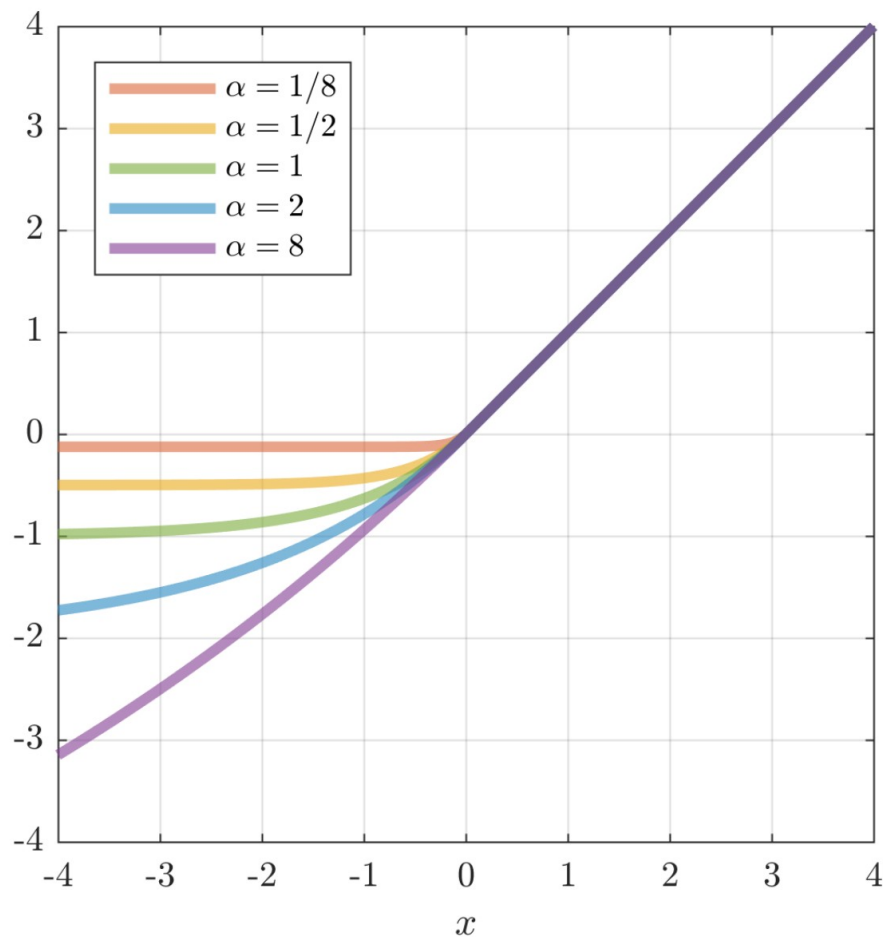
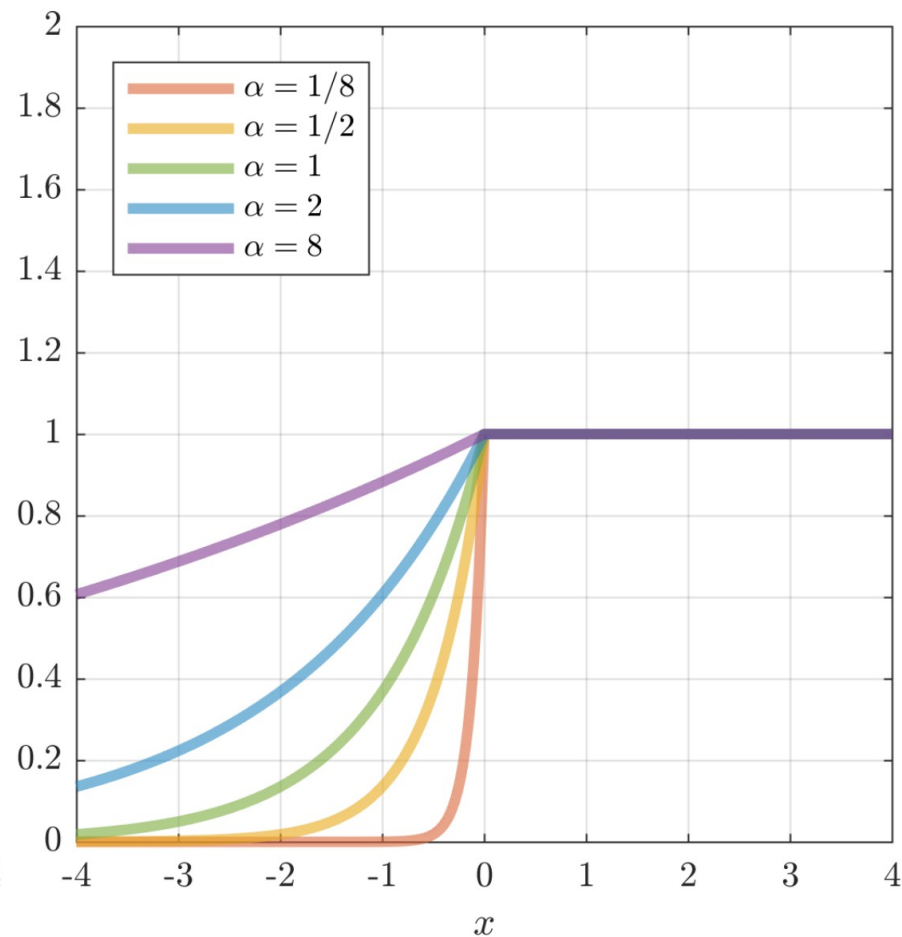


Barron (2017) Continuously differentiable exponential linear units

CELU – nn.CELU()

(a) $\text{ELU}(x, \alpha)$ (b) $\frac{d}{dx} \text{ELU}(x, \alpha)$

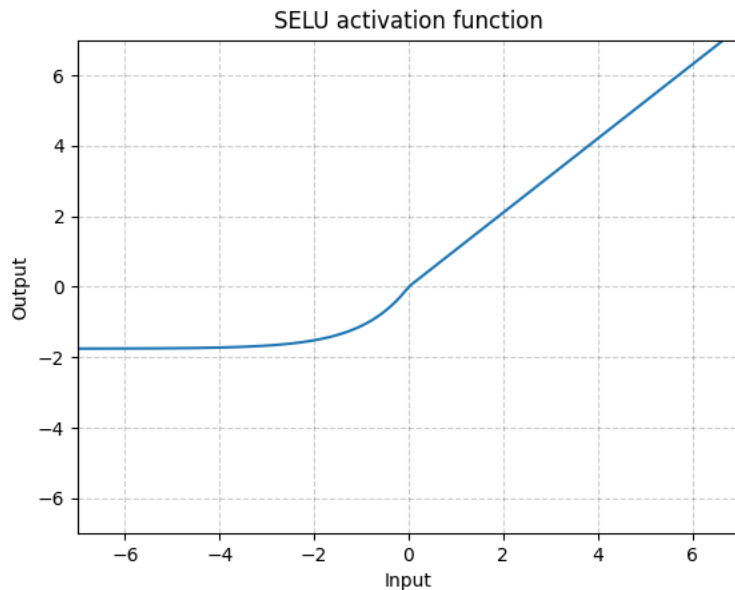
CELU – `nn.CELU()`

(c) $\text{CELU}(x, \alpha)$ (d) $\frac{d}{dx} \text{CELU}(x, \alpha)$

SELU – `nn.SELU()`

$$\text{SELU}(x) = \text{scale} * (\max(0, x) + \min(0, \alpha * (\exp(x) - 1)))$$

with $\alpha = 1.6732632423543772848170429916717$ and $\text{scale} = 1.0507009873554804934193349852946$.



SELU – nn.SELU()

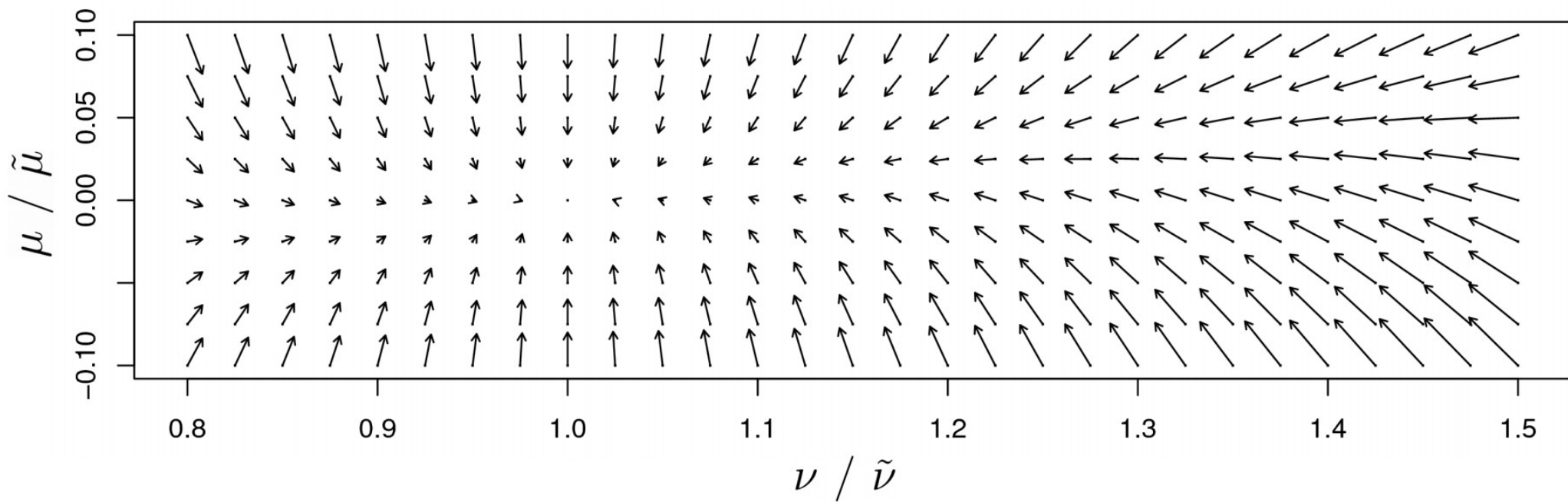
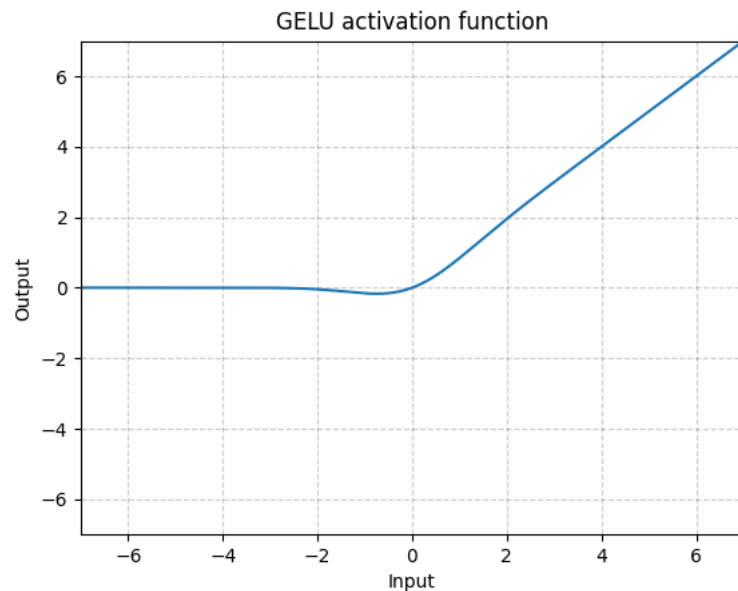


Figure 2: For $\omega = 0$ and $\tau = 1$, the mapping g of mean μ (x -axis) and variance ν (y -axis) to the next layer's mean $\tilde{\mu}$ and variance $\tilde{\nu}$ is depicted. Arrows show in which direction (μ, ν) is mapped by $g : (\mu, \nu) \mapsto (\tilde{\mu}, \tilde{\nu})$. The fixed point of the mapping g is $(1, 1)$.

GELU – `nn.GELU()`

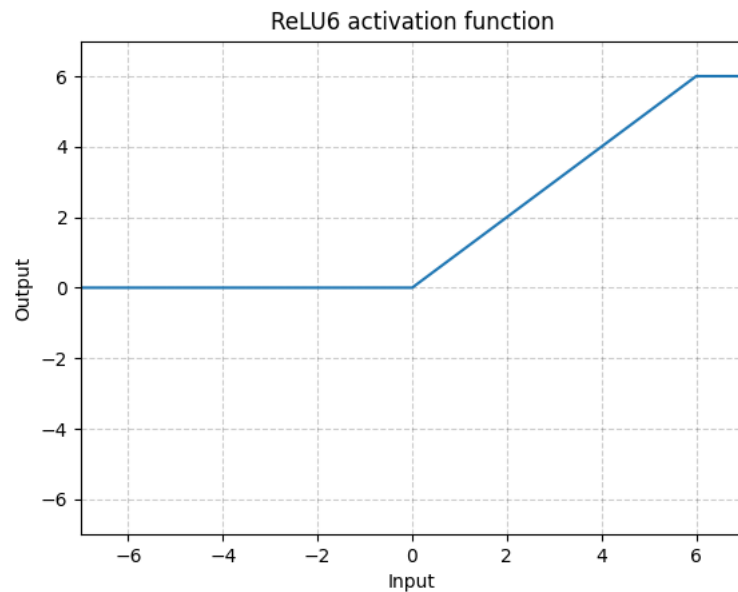
$$\text{GELU}(x) = x * \Phi(x)$$

where $\Phi(x)$ is the Cumulative Distribution Function for Gaussian Distribution.



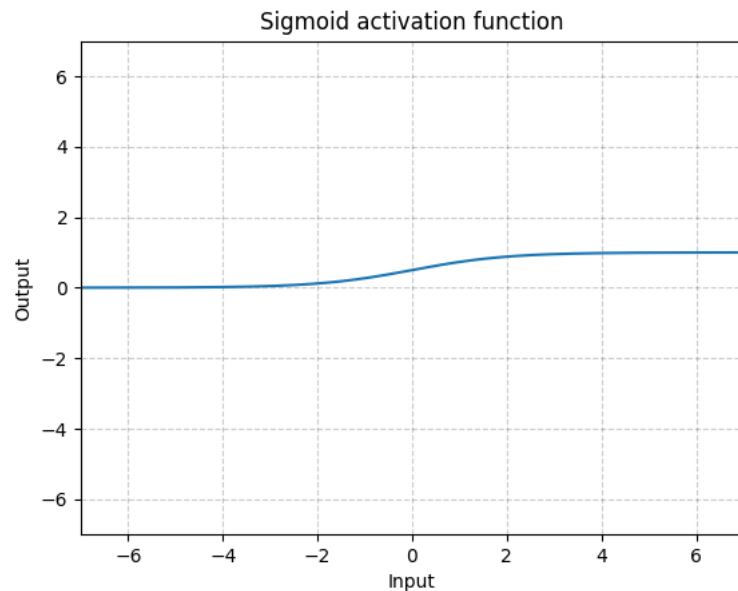
ReLU6 – `nn.ReLU6()`

$$\text{ReLU6}(x) = \min(\max(0, x), 6)$$



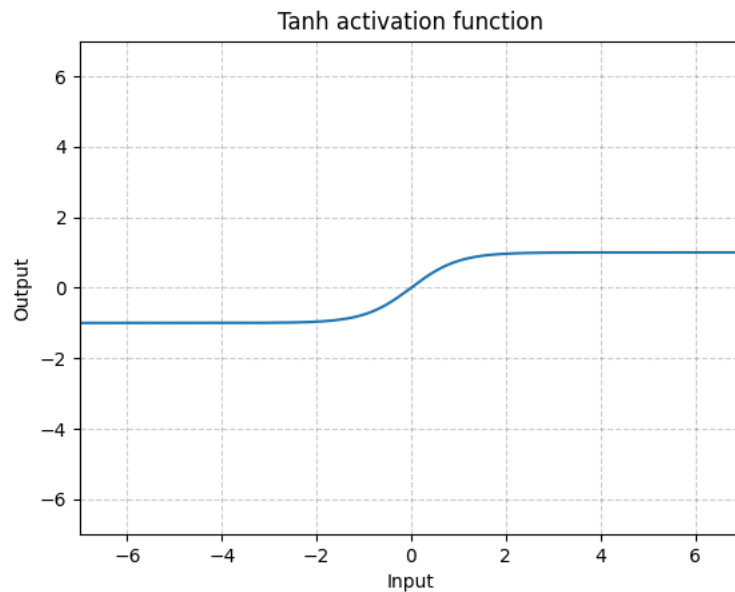
Sigmoid – `nn.Sigmoid()`

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$



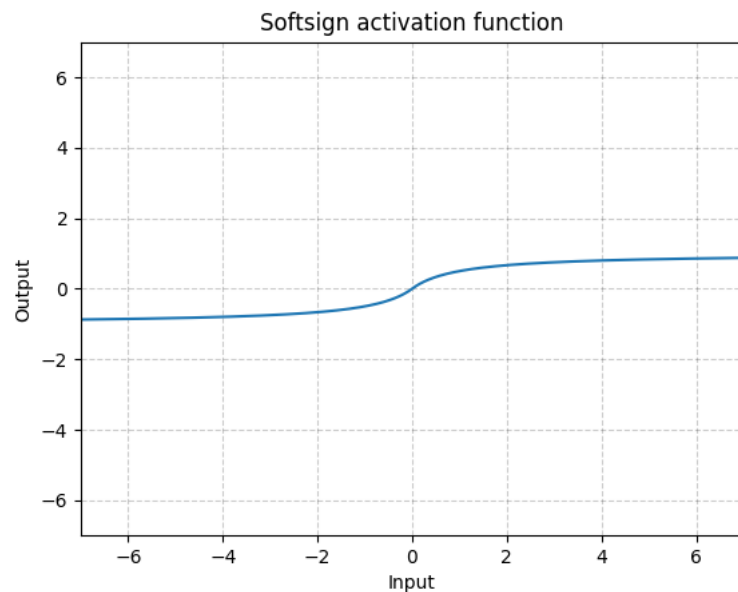
Tanh – `nn.Tanh()`

$$\text{Tanh}(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



Softsign – `nn.Softsign()`

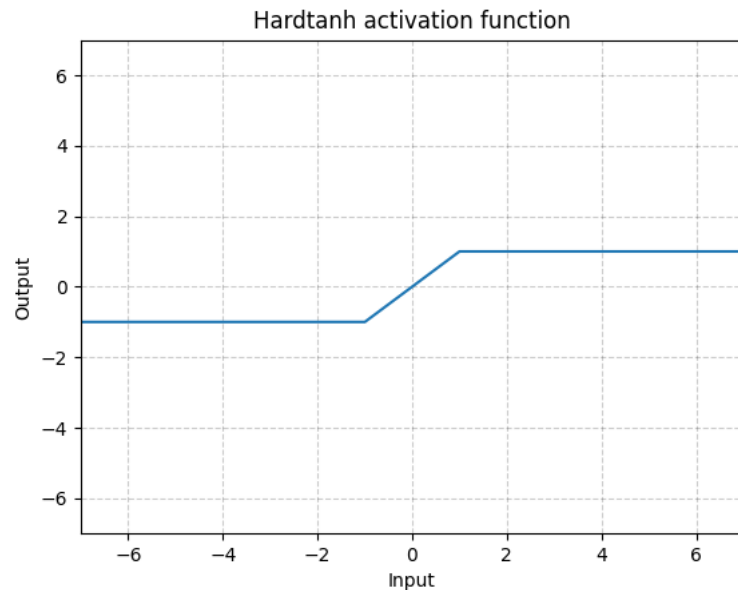
$$\text{SoftSign}(x) = \frac{x}{1 + |x|}$$



Hardtanh – `nn.Hardtanh()`

$$\text{HardTanh}(x) = \begin{cases} 1 & \text{if } x > 1 \\ -1 & \text{if } x < -1 \\ x & \text{otherwise} \end{cases}$$

The range of the linear region $[-1, 1]$ can be adjusted using `min_val` and `max_val`.



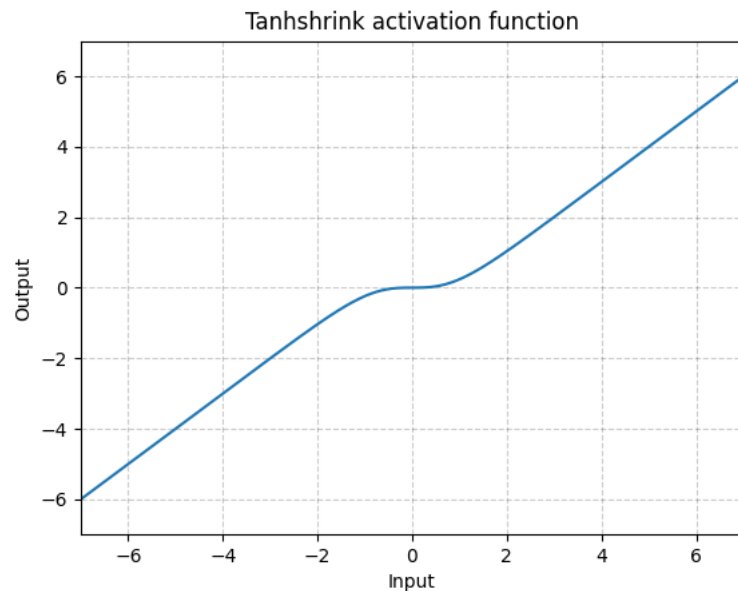
Threshold – `nn.Threshold()`

Threshold is defined as:

$$y = \begin{cases} x, & \text{if } x > \text{threshold} \\ \text{value}, & \text{otherwise} \end{cases}$$

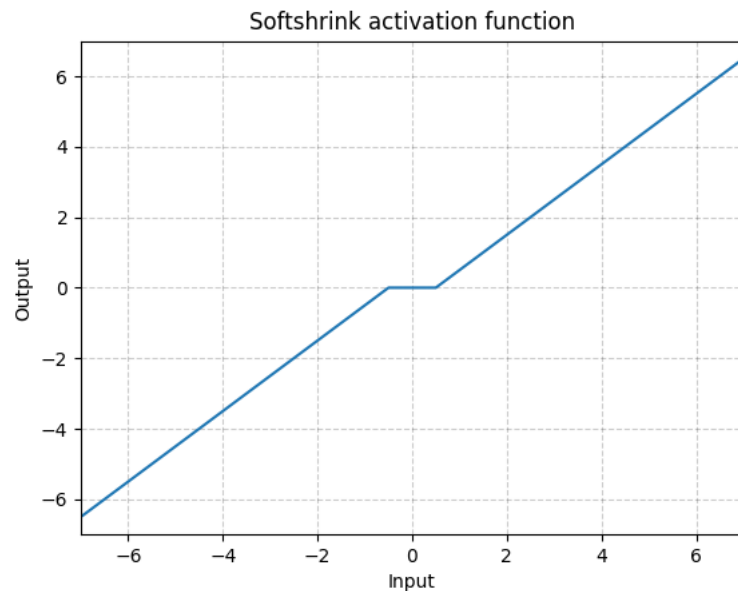
Tanhshrink – `nn.Tanhshrink()`

$$\text{Tanhshrink}(x) = x - \tanh(x)$$



Softshrink – `nn.Softshrink()`

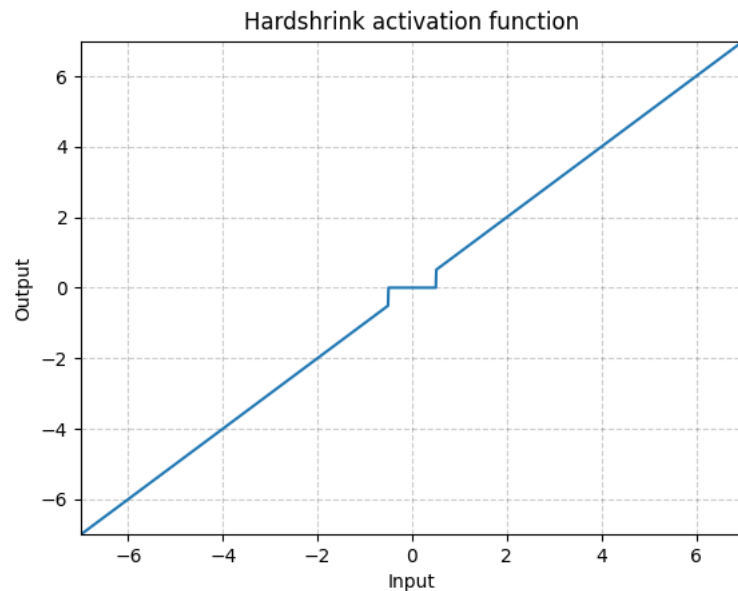
$$\text{SoftShrinkage}(x) = \begin{cases} x - \lambda, & \text{if } x > \lambda \\ x + \lambda, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$



Hardshrink – `nn.Hardshrink()`

Applies the hard shrinkage function element-wise:

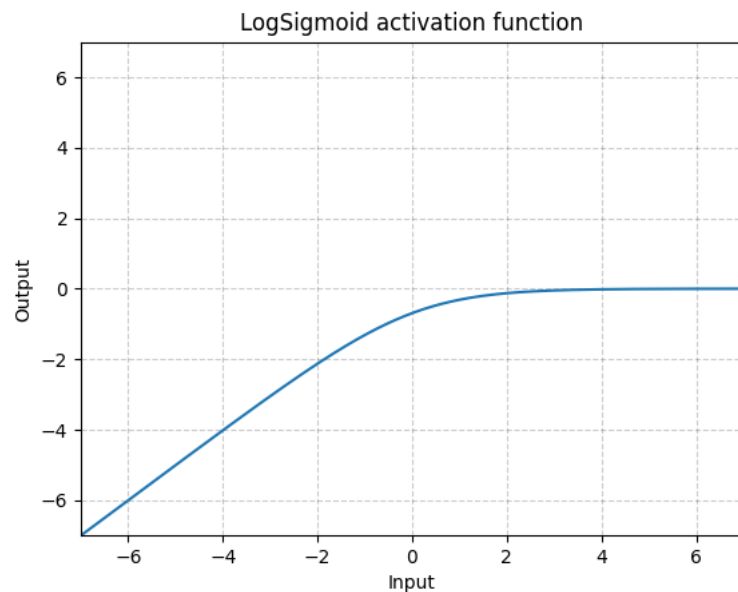
$$\text{HardShrink}(x) = \begin{cases} x, & \text{if } x > \lambda \\ x, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$



LogSigmoid – `nn.LogSigmoid()`

Applies the element-wise function:

$$\text{LogSigmoid}(x) = \log \left(\frac{1}{1 + \exp(-x)} \right)$$



Softmin – `nn.Softmin()`

Applies the Softmin function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range $[0, 1]$ and sum to 1.

Softmin is defined as:

$$\text{Softmin}(x_i) = \frac{\exp(-x_i)}{\sum_j \exp(-x_j)}$$

Softmax – `nn.Softmax()`

Applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range $[0,1]$ and sum to 1.

Softmax is defined as:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

LogSoftmax – `nn.LogSoftmax()`

Applies the $\log(\text{Softmax}(x))$ function to an n-dimensional input Tensor. The LogSoftmax formulation can be simplified as:

$$\text{LogSoftmax}(x_i) = \log \left(\frac{\exp(x_i)}{\sum_j \exp(x_j)} \right)$$