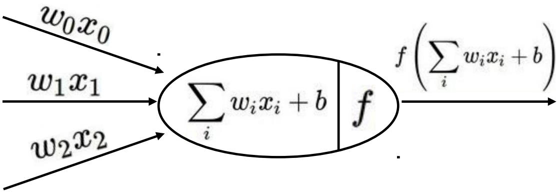
Activation Functions

Alvaro Soto

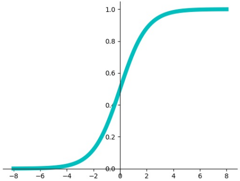
Computer Science Department, PUC

Activation Function



What activation function did we use for Multi Layer Perceptrons (MLPs)?

Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$



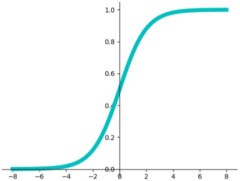
Good properties:

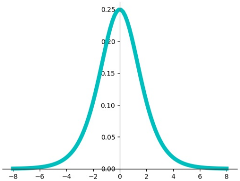
- Smooth and differentiable.
- Pushes output values towards extremes, good for classification.

Sigmoid function:

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

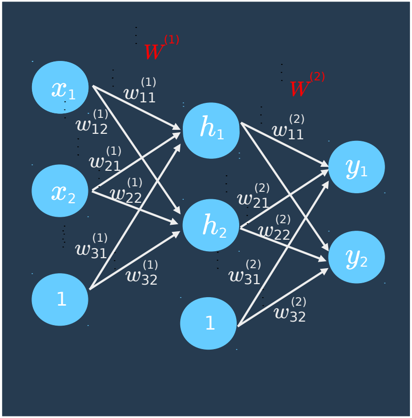
$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$





Bad property 1:

- Saturated neurons have gradients close to zero.
- Weight update uses gradient descent: $w_i^{new} = w_i^{old} - \eta \frac{\partial E}{\partial w_i}$
- Any problem?



$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \sigma \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \sigma \left(W^{(1)} X \right)$$

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \sigma \begin{pmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \sigma \left(W^{(2)} H \right)$$

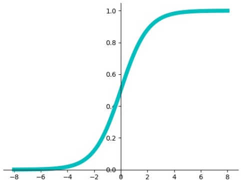
$$Y = \sigma \left(W^{(2)} \sigma \left(W^{(1)} \vec{X} \right) \right)$$

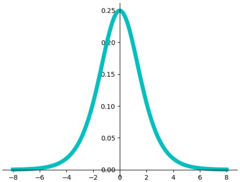
Backprop chains weight updates across the network. Any problem?.

Activation Function: Sigmoid

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

$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$





Bad property 1:

- Saturated neurons have gradients close to zero.
- Weight update uses gradient descent: $w_i^{new} = w_i^{old} - \eta \frac{\partial E}{\partial w_i}$
- Any problem?
  - Slow convergence.
  - In deep learning this can cause vanishing gradient problem. This is a big problem for Recurrent Neural Networks (RNNs), we will talk about this later in the course.

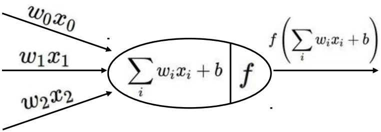
Bad property 2:

- Sigmoid outputs are not zero-centered.
- Let's L be the loss function and o_n the output of an intermediate neuron:

$$o_n = f\left(\sum_i w_i x_i + b\right)$$

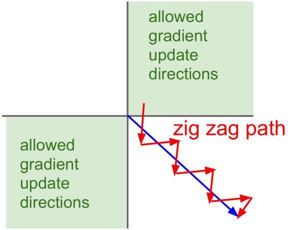
$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial w_i} = \frac{\partial L}{\partial f} x_i$$

- If $x_i > 0$ then the gradient $\frac{\partial L}{\partial w_i}$ has always the same sign as $\frac{\partial L}{\partial f}$.
- Therefore all weight gradients are positive or negative, why?.



Bad property 2:

- Sigmoid outputs are not zero-centered.
- Therefore all weight gradients are positive or negative.
- This produces a zig-zag effect:



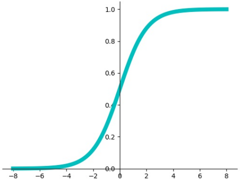
- This is also why one should use zero mean data.

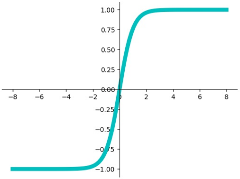
Obs: Gradients sum across a mini-batch can mitigate the problem (why?).

Centered sigmoid function: Tanh

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

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



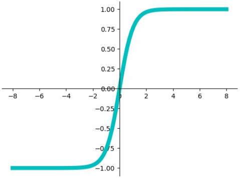


Good properties:

- Smooth and differentiable.
- Pushes output values towards extremes, good for classification.
- Zero centered.

Activation Function: Zero Centered Sigmoid

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

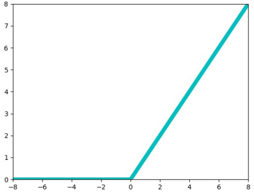


Bad Property:

- Still saturated neurons have gradient close to zero.
- Slow convergence and vanishing gradient problem.
- Actually another problem of sigmoid functions is that they are strongly sensitive to their input when is near 0.

Activation Function: ReLU

Rectified Linear Unit: $\textbf{ReLU}(x) = max(0, x)$

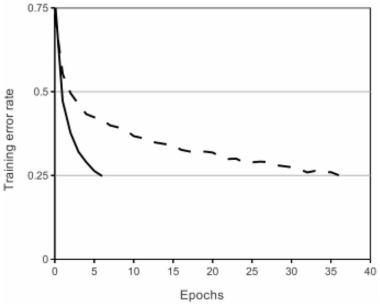


Good properties:

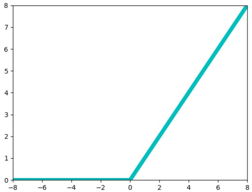
- Does not saturate in positive part of the input space.
- Faster convergence than sigmoid functions (ex. 10x) (why?).
- Computationally efficient because it inherits the nice properties of lineal functions.

ReLU achieves faster convergence than sigmoids

- Example (Krizhevsky et al., 2012): training on CIFAR-10 (a popular object recog. dataset), a 4-layer CNN using ReLUs (solid line) reaches a 25% training error rate six times faster than an equivalent network using $tanh(x)$ (dashed line).



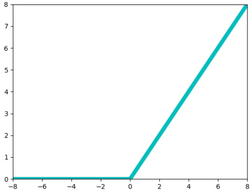
Rectified Linear Unit: $\textsf{ReLU}(x) = max(0, x)$



Bad properties:

1. Non zero-centered output.
2. Non differentiable at zero.
3. Saturate with negative inputs.
4. Not so good for classification.

Rectified Linear Unit: **ReLU**$(x) = max(0, x)$

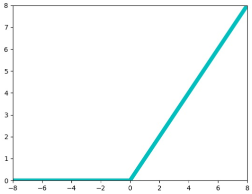


Bad properties:

1. Non zero-centered output.

- It is recommended to initialize neurons with a positive bias, ex. 0.1 (why?).
- As we will see later, some ReLU variants provide a nonzero output on the negative size: ex. Leaky ReLUs.

Rectified Linear Unit: **ReLU**$(x) = max(0, x)$

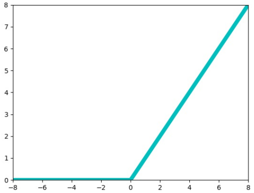


Bad properties:

2. Non differentiable at zero.

- At $x = 0$, left derivative is 0 and right derivative is 1, not good!.
- In practice, software implementations usually return one of the one-sided derivatives (do not report: undefined).
- In practice, one can safely disregard the non-differentiability of the hidden unit activation functions.

Rectified Linear Unit: $\text{ReLU}(x) = max(0, x)$

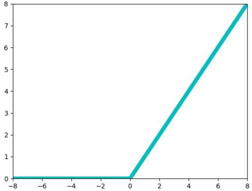


Bad properties:

3. Saturate with negative inputs.

- ReLU does have a problem with vanishing gradients.
- However, it is less critical than in the case of a sigmoid activation function.
- As we will see, Leaky ReLUs can help to reduce this problem.

Rectified Linear Unit: **ReLU**$(x) = max(0, x)$



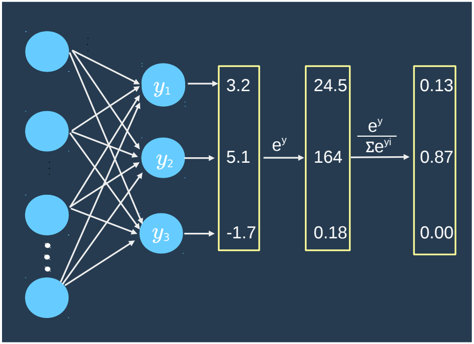
Bad properties:

4. Not so good for classification.

- For winner-take-all classification, output neurons usually use a softmax activation function.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_i e^{z_i}}$$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_i e^{z_i}}$$



- Soft-max moves NW output from unnormalized scores to normalized probabilites.

Activation Function: SoftMax

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_i e^{z_i}}$$

Sofmax is more than normalization

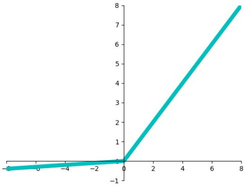
- It is differentiable.
- It works as a softmax pushing the highest value up and the rest down. So it fosters a competition among outputs (in the extreme: winner-take-all).
- It is a good output for a log-likelihood optimization such as cross-entropy.

$$\log \text{softmax}(z_i) = z_i - \log \sum_i e^{z_i}$$

  - Input z_i has a direct contribution to the cost function independent of how small is its contribution in the sum.
  - When we optimize the loss function, the first term encourages the right z_i to be pushed up, while the second term encourages all the output to be pushed down.

Leaky (Parametric) Rectified Linear Unit:

$$\mathbf{LReLU}(x) = max(\alpha x, x)$$

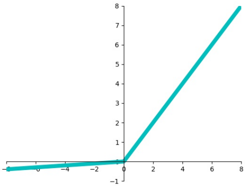


Good properties:

- Does not saturate (neurons do not die).
- Faster convergence than sigmoid functions (ex. 10x) (why?).
- Computationally efficient.
- Not so good for classification.

Leaky (Parametric) Rectified Linear Unit:

$$\textbf{LReLU}(x) = max(\alpha x, x)$$

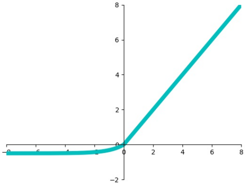


Bad properties:

- Non zero-centered output.
- Not so good for classification.

Exponential Linear Units:

$$\textbf{ELU(x)} = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \tag{1}$$

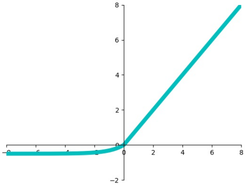


Good properties:

- All benefits of ReLUs and LReLUs.
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise.

Exponential Linear Units:

$$\textbf{ELU(x)} = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \tag{2}$$



Bad properties:

- Requiere calculation of $e(x)$.