Statistical Machine Learning
Activation Functions

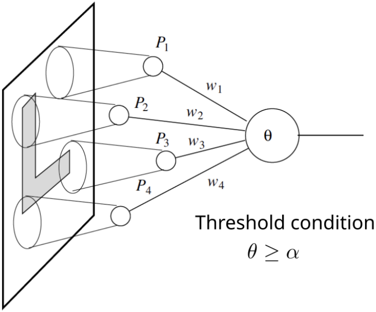
Horacio Gómez-Acevedo
Department of Biomedical Informatics
University of Arkansas for Medical Sciences

March 4, 2024

UAMS

What are activation functions?

Recall that in the threshold logic, once we receive the input, a decision must be made to *fire* or *suppress* the output.

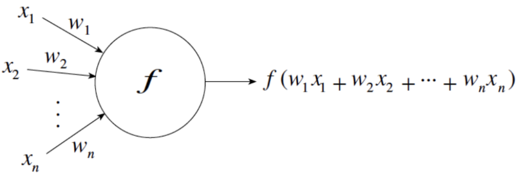


Threshold condition

$$\theta \geq \alpha$$

Activation functions

Activation functions generalize the concept of activation given a specific input.

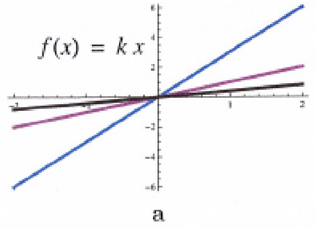


Linear Functions

These functions are used to model the firing rate of a neuron.

$$f(x) = kx \quad k \text{ is a positive constant}$$

Pros: It is continuous and differentiable.

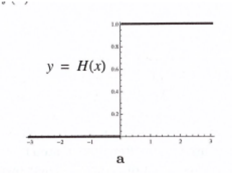


Step Functions

These are biologically inspired type of activation.

1. Heaviside function (threshold step function). This function fires only for positive values.

$$H(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$



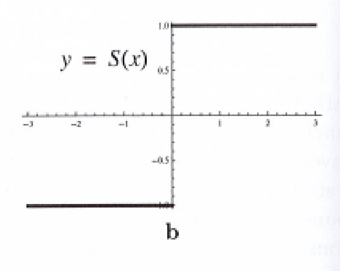
This function is differentiable except in $x = 0$. Informally, people refers to the Dirac delta function as its *derivative*.

$$H'(x) = \delta(x) = \begin{cases} 0 & \text{if } x \neq 0, \\ \infty & \text{if } x = 0 \end{cases} \quad \text{it also satisfies} \int_{-\infty}^{\infty} \delta(x)dx = 1$$

Step Functions (cont)

2. Signum Function. This is a variation of the Heaviside function

$$S(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



Hockey stick functions

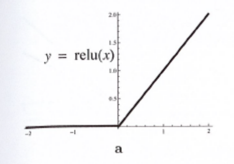
1. Rectified Linear Unit (ReLu). The activation is linear for $x \geq 0$.

$$ReLU(x) = xH(x) = \max\{x, 0\} = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } x \geq 0. \end{cases}$$

Pros:

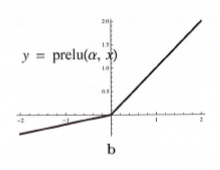
- This function does not saturate (see below with sigmoid functions)
- Neural networks with ReLU activation functions tend to learn several times faster than similar networks with saturating activation function.

Hockey stick functions



2. Parametric Rectified Linear Unit (PReLUI). In this case the activation is piecewise linear, having different firing rates for $x < 0$ and $x > 0$

$$PReLU(\alpha, x) = \begin{cases} \alpha x & \text{if } x < 0, \\ x & \text{if } x \geq 0. \end{cases} \quad \alpha > 0.$$



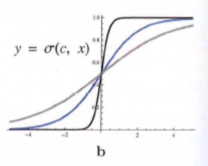
Sigmoid functions

These types of activation functions have that advantage that they are smooth and can approximate a step function to any degree of accuracy.

1. Logistic function with parameter $c > 0$.

$$\sigma_c(x) = \frac{1}{1 + \exp(-cx)}$$

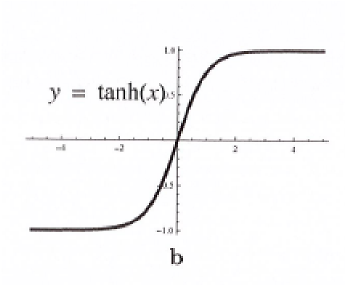
The parameter $c > 0$ controls the firing rate of the neuron. Large values of c correspond to a fast change of values from 0 to 1.



Sigmoid Functions

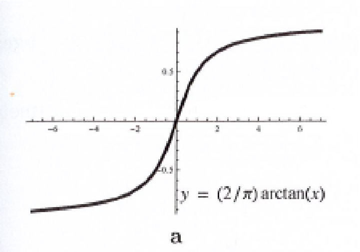
1. Hyperbolic tangent.

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



1. Arctangent function.

$$h(x) = \frac{2}{\pi}\arctan(x)$$



Summary of Activation Functions

We have seen various types of activation function. All these functions have some characteristic in common, more specifically

- they are well defined for every point in the real line,
- they are non-decreasing,
- they are smooth almost everywhere.

Cost Functions

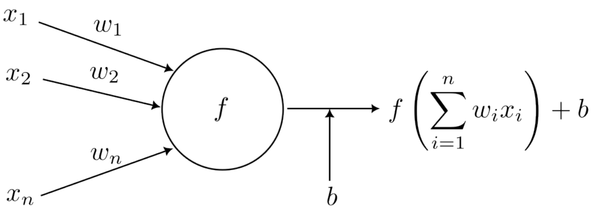
In the learning process, the parameters of a neural network are subject to minimize a certain objective function which represents a measure of proximity between the prediction of the network and the associated target. Note that these functions are also called *error functions* or *loss functions*. For instance, in multi linear regression we have RSS as the cost function.

Input, Output and Target

The **input** of a neural networks is obtained from given data or from sensors that perceive the environment. The input is a variable that is fed into the network. It can be one-dimensional variable x, or a vector $\mathbf{x} \in \mathbb{R}^n$, a matrix, a tensor, or a random variable X.

The network acts as a function and provides an **output** which can be one-dimensional $y \in \mathbb{R}$, or a vector $\mathbf{y} \in \mathbb{R}^n$, a matrix, a tensor, or a random variable Y. We normally denote the **input-output** mapping by $f_{w,b}$. With the previous notations $f_{w,b}(x) = y$, $f_{w,b}(\mathbf{x}) = \mathbf{y}$, and $f_{w,b}(X) = Y$.

Input, Output, and Target (cont)



The **target function** is the desired relations which the network tries to approximate. This function is independent of the parameters w and will be denoted by $z = \phi(x)$, $\mathbf{z} = \phi(\mathbf{x})$ or $Z = \phi(X)$.

Goal

The neural network will tune the parameters (w, b) until the output variable y will be in a proximity of the target variable z.
The proximity will be determined by the cost function
$C(w, b) = \text{distance}(y, z)$
The optimal parameters of the network are given by

$$(w^*, b^*) = \arg\min_{w,b} C(w, b)$$

The process by which the parameters (w, b) are tuned into (w^*, b^*) is called **learning**.

Supremum Cost Function

Let's assume that a neural network takes inputs in $x \in [0,1]$, is supposed to learn a given continuous function $\phi\colon [0,1] \to \mathbb{R}$. If $f_{w,b}$ is the input-output mapping of the network the associated cost function is

$$C(w,b) = \sup_{x\in[0,1]} |f_{w,b}(x) - \phi(x)|$$

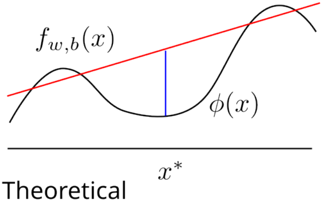
For all practical purposes, when the target function is known at n points

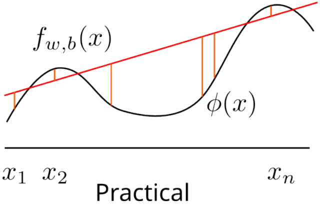
$$z_1 = \phi(x_1), z_2 = \phi(x_2), \ldots, z_n = \phi(x_n)$$

then, the supremum cost function becomes

$$C(w,b) = \max_{1\le i\le n} |f_{w,b}(x_i) - z_i|$$

Sup Cost Function (cont)





L^2 Cost Function

Assume the input of the network is $x \in [0,1]$ and that the target function $\phi\colon [0,1] \to \mathbb{R}$ is square integrable. If $f_{w,b}$ is the input-output mapping, the associated cost functions measures the distance in the L^2 norm between the output and the target

$$C(w,b) = \int_0^1 (f_{w,b}(x) - \phi(x))^2 dx.$$

If the target function is known at only n points

$$z_1 = \phi(x_1), z_2 = \phi(x_2), \ldots, z_n = \phi(x_n)$$

then this cost function becomes the square of the Euclidean distance in $\mathbb{R}^n$ between $f_{w,b}$ and z

$$C(w,b) = \sum_{i=1}^{n} (f_{w,b}(x_i) - z_i)^2 = \|f_{w,b}(\mathbf{x}) - \mathbf{z}\|^2$$

Mean Square Error Cost Function

Consider a neural network whose input is a random variable X, and its output is the random variable $Y = f_{w,b}(X)$. Assume that the network is used to approximate the target random variable Z. The cost function will measure the proximity between the output and the target random variables Y and Z. A good candidate is given by the expectation of their squared difference

$$C(w, b) = \mathbb{E}[(Y - Z)^2] = \mathbb{E}[(f_{w,b}(X) - Z)^2]$$

Let's consider the so-called **training set** consisting of n measurements of random variables (X, Z), which are given by (x_i, z_i). Then the cost function becomes the **empirical mean of the square difference of Y and Z**.

$$\hat{C}(w, b) = \frac{1}{n}\sum_{j=1}^{n}(f_{w,b}(x_j) - z_j)^2$$

Cross-entropy

Let p and q be two densities in $\mathbb{R}$ (loosely speaking, this means that the these functions can describe a random variable).
The negative likelihood function $-\ln(q(x))$ measures the information given by $q(x)$. The **cross-entropy** of p with respect to q is defined as

$$S(p,q) = -\int_{-\infty}^{\infty} p(x) \ln q(x) dx$$

This represents the information given by $q(x)$ assessed from the point of view of the distribution $p(x)$. The **Shannon entropy** is defined by

$$H(p) = -\int_{-\infty}^{\infty} p(x) \ln p(x) dx$$

Kullback-Leibler Divergence

The difference between the cross entropy and the Shannon entropy is the **Kullback-Leibler divergence**

$$D_{KL}(p, q) = S(p, q) - H(p) = \int_{-\infty}^{\infty} p(x) \ln \frac{q(x)}{p(x)} dx$$

Note that KL-divergence is not a distance, but both cross-entropy and KL-divergence can be used as cost functions for neural networks.

References

Materials and some of the pictures are from (Calin, 2019).

Calin, O. (2019). *Deep Learning Architectures*. Springer Series in the Data Sciences. Springer. ISBN: 978-3-030-36723-7.

I have used some of the graphs by hacking TiKz code from StakExchange, Inkscape for more aesthetic plots and other old tricks of $\TeX$