Multilayer Neural Networks

Pattern Classification by Richard O. Duda, Peter E . Hart and David G. Stork
Chapter 6 - Part 1 (introduction)

KERESTÉLY ÁRPÁD & MAJERCSIK LUCIANA

9.04.2019

The classification problem

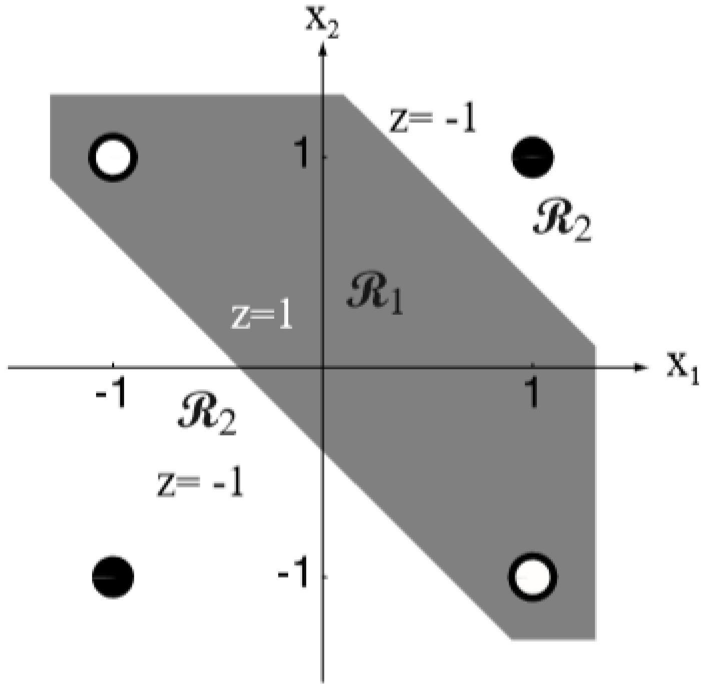
1. Let’s consider the two-bit parity or exclusive-OR problem

2. One can easily see that for this specific problem the points are not linearly separable

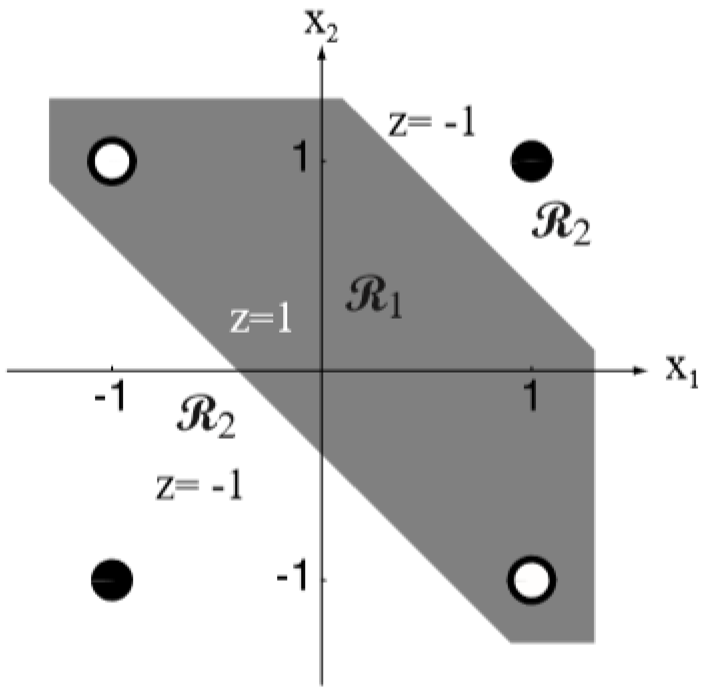
3. What can be done?

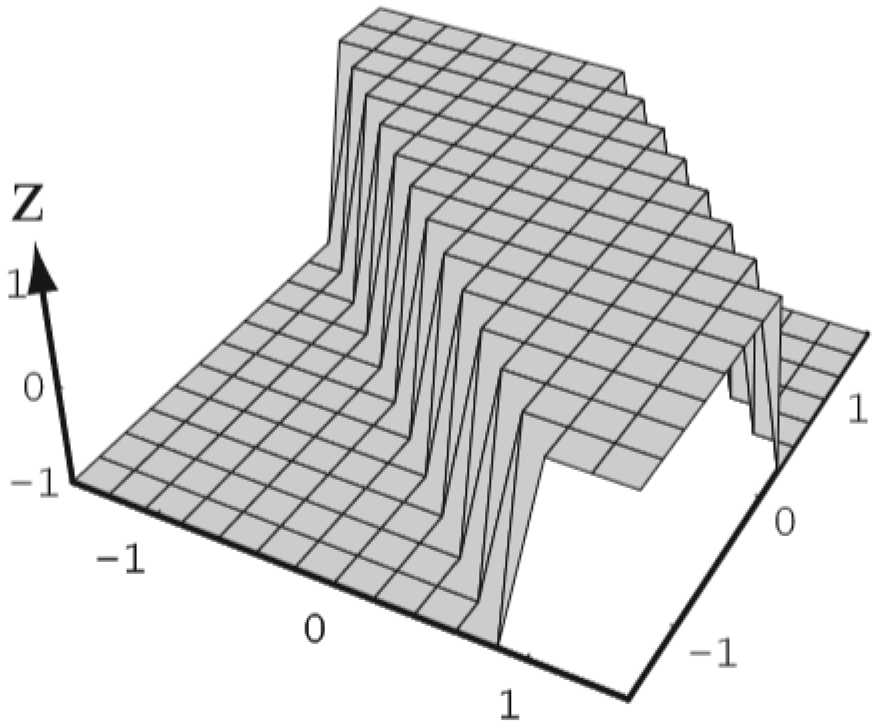
4. Go on with the SVM idea : Make a transformation of the space with the sample points (2D here) into a higher dimensional space such that the points are linearly separable there

5. The simplest transformation here would be into a 3D space (intuitively, it is easier to understand)



The transformation:

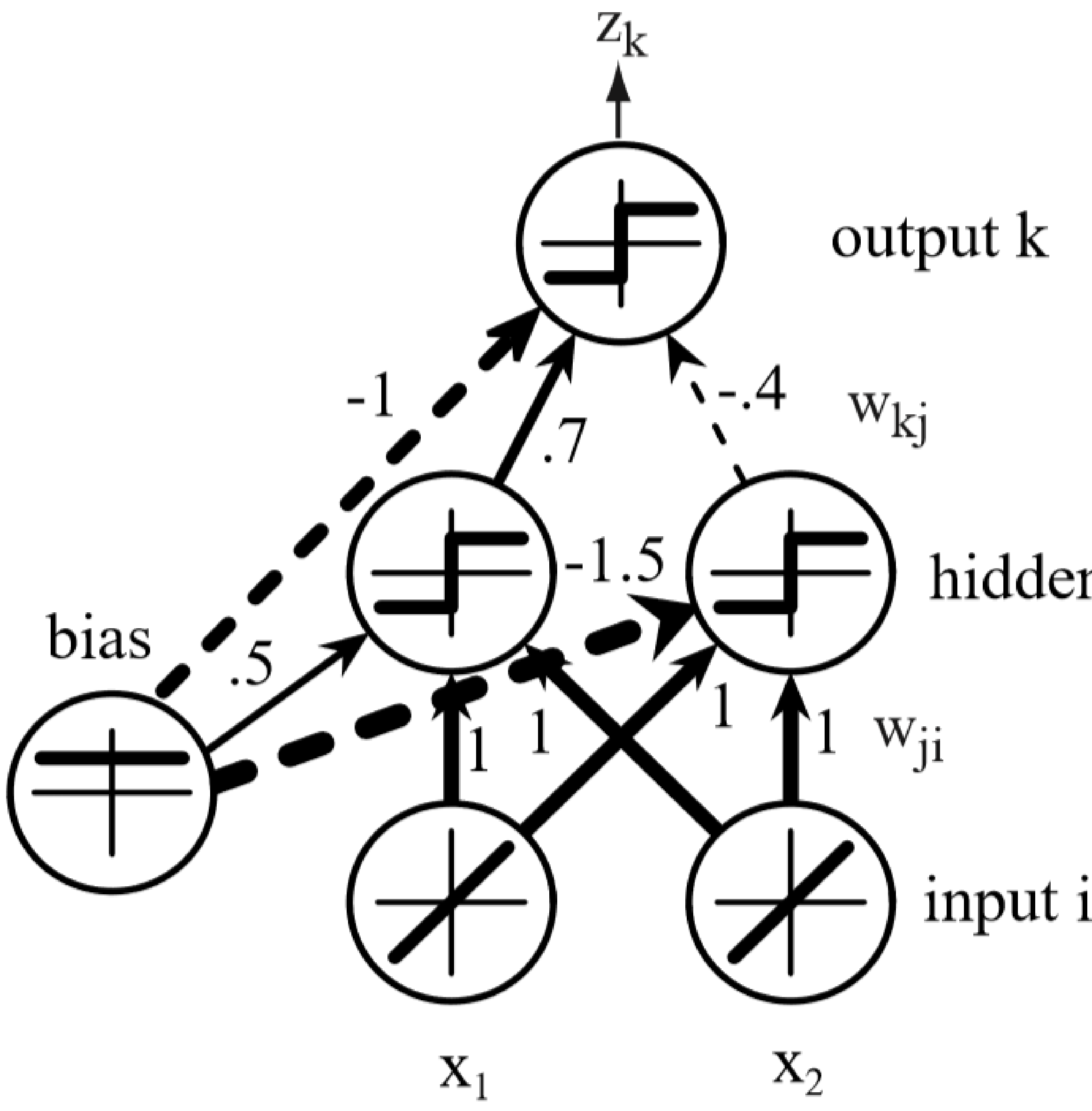




How can we find the transformation function behind this?

Solving the problem with a Neural Network

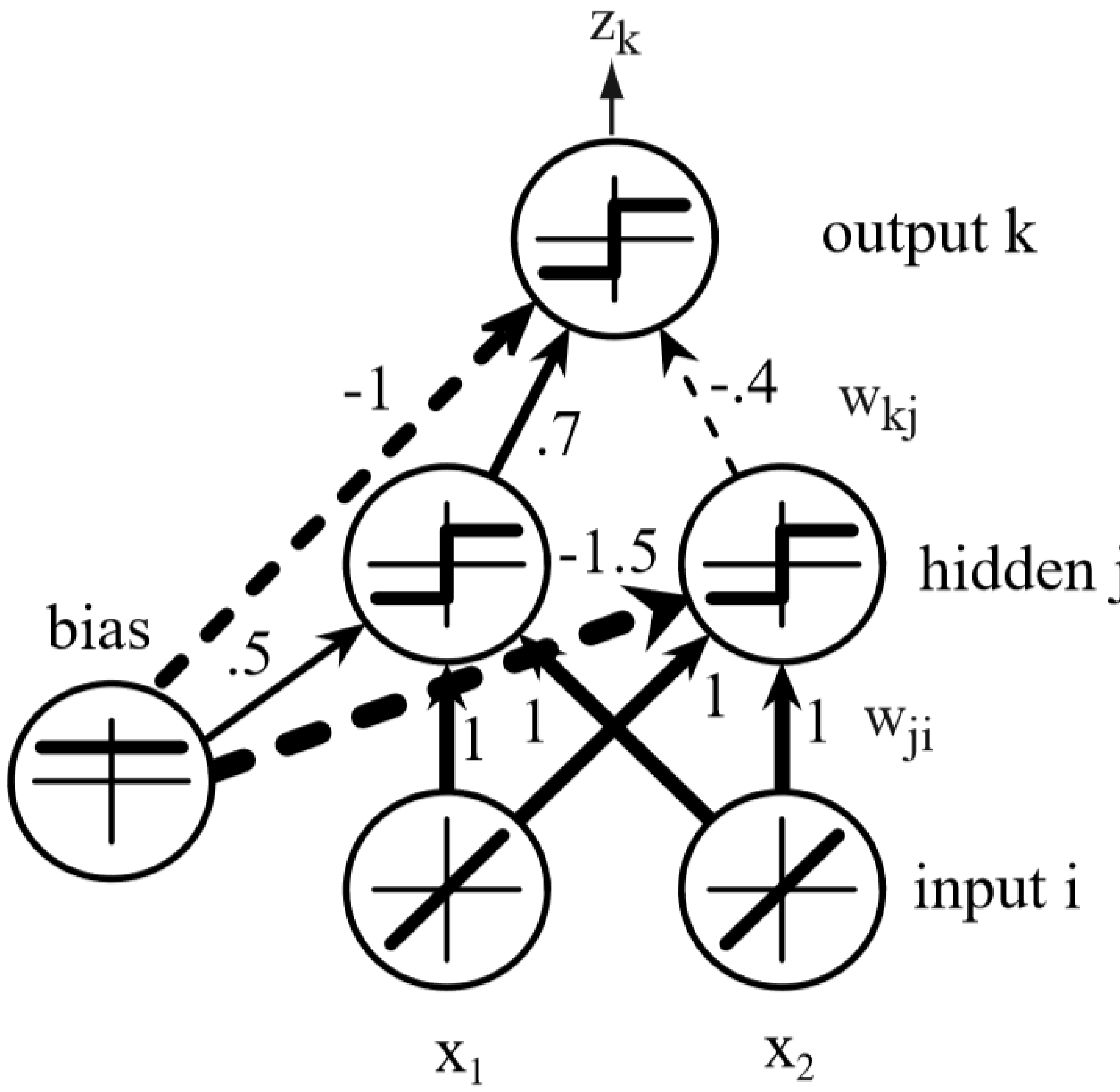
- The figure on the right shows a simple three-layer neural network for this problem. The network consists of:
  - an input layer (having two input units),
  - a hidden layer (with two hidden units),
  - an output layer (a single unit).
- The input units represent the components of a feature vector (to be learned or to be classified) and signals emitted by output units will be discriminant functions used for classification.



Solving the problem with a Neural Network

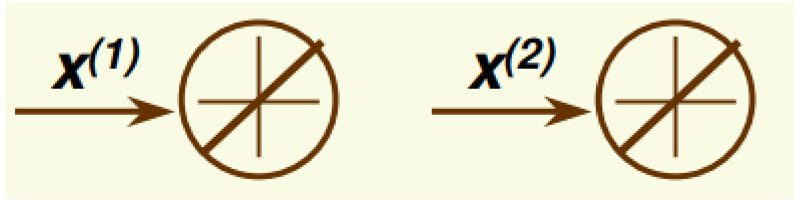
- The layers are interconnected by modifiable weights, represented by links between layers.

- There is a single bias unit that is connected to each unit other than the input units. The function of units is loosely based on properties of biological neurons, and hence they are sometimes called "neurons."



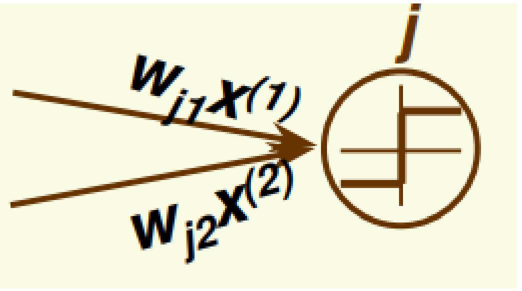
How does it work?

1. Each two-dimensional input vector is presented to the input layer, and the output of each input unit equals the corresponding component in the vector.



2. Each hidden unit performs the weighted sum of its inputs to form its (scalar) ***net activation*** or simply ***net***. That is, the net activation is the inner product of the inputs with the weights at the hidden unit.

$$net_j = \sum_{i=1}^{d} x^{(k)} w_{ji} + w_{j0}$$



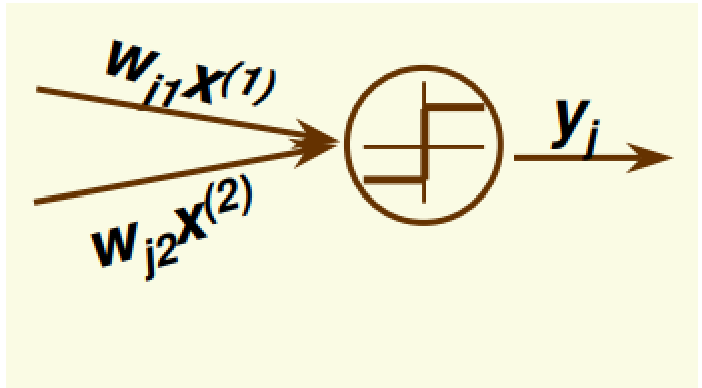
How does it work?

For simplicity, we augment both the input vector (i.e., append a feature value $x_0 = 1$) and the weight vector (i.e., append a value w_0), and can then write

$$net_j = \sum_{i=1}^{d}(x_i w_{ji} + w_{j0}) = \sum_{i=0}^{d}(x_i w_{ji}) = w_j^t x$$

3. Each hidden unit emits an output that is a **nonlinear** function of its activation, f(net), i.e.,

$$y_j = f(net_j)$$



How does it work?

- The example shows a simple threshold or sign (read “signum”) function,

$$f(net) = \begin{cases} 1, if\ net \geq 0 \\ -1, if\ net < 0 \end{cases}$$

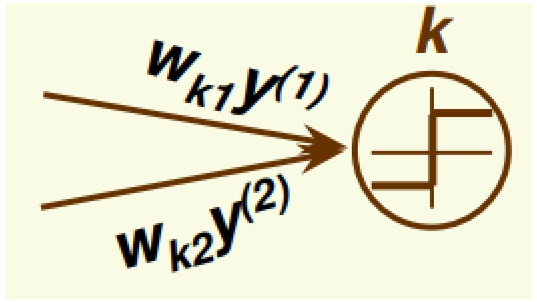
but as we shall see, other functions have more desirable properties and are hence more commonly used.

- This f() is sometimes called the ***transfer function*** or ***activation function***. So far, we have assumed the same activation function is used at the various hidden and output units, though this is not crucial.

How does it work?

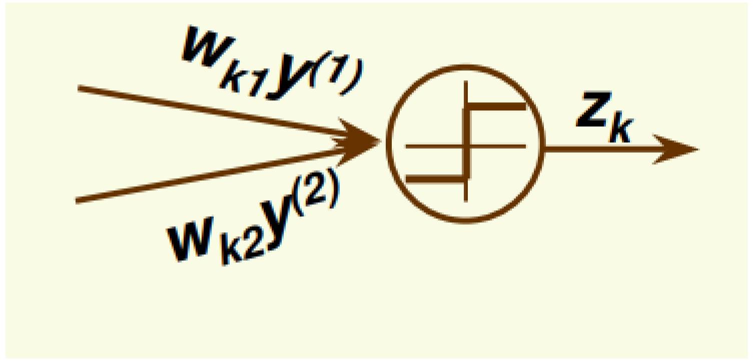
4. Each output unit k similarly computes its net activation based on the hidden unit signals as:

$$net_k = \sum_{j=1}^{n_h}\left(y_j w_{kj} + w_{k0}\right) = \sum_{j=0}^{n_h}\left(y_j w_{kj}\right) = w_k^t y$$



5. Each output unit k emits a nonlinear function of its activation

$$z_k = f(net_k)$$



Discriminant Function

- In a two-category case, such as XOR, it is traditional to use a single output unit and label a pattern by the sign of the output z.

Discriminant Function

- We can gather all the terms in previous slides in the discriminant function for class k (the output of the k th output unit)

$$g_k(x) = z_k = f(\sum_{j=1}^{n_h} w_{kj} f\left(\sum_{j=0}^{d} x_i w_{ji} + w_{j0}\right) + w_{k0})$$

Given samples x_1 ,..., x_n each of one of the **m** classes, for each sample x, we wish

$$g_k(x) = \begin{cases} 1, if\ x \in Class\ k \\ 0, otherwise \end{cases}$$

Discriminant Function

- The goal is to learn (to adjust) weights w_{kj} and w_{ji} to achieve the desired $g_k(x)$ for all k

Why neural networks can do this job?

- Because the **universal approximation theorem** says they can ☺**.**
- The universal approximation theorem in mathematical terms says that:

Let $\varphi: R \rightarrow R$ be a nonconstant, bounded and continuous function. Let I_m denote the m-dimensional unit hypercube. Then, given any $\varepsilon > 0$ and any $f \in C(I_m)$, there exists an integer N and the real constants v_i, b_i and the real vectors $w_i \in R^m$, such that, if we define the function

$$F(x) = \sum_{i=0}^{N} v_i \varphi(w_i^T x + b_i)$$

Then

$$|F(x) - f(x)| < \varepsilon, \forall x \in I_m$$

How is the formula related to NN?

- $F(x) = \sum_{i=0}^{N} v_i \varphi(w_i^T x + b_i)$

Universal approximation theorem

- In the mathematical theory of artificial neural networks, the **universal approximation theorem** states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function.

- In other words, the theorem states that **simple neural networks can *represent* a wide variety of interesting functions** when given appropriate parameters

We wanted to see that it actually works ☺

- We took the following function

$$f(x) = 0.2+0.4*x^2+0.3*\sin(15x)+0.05*\cos(50x)$$

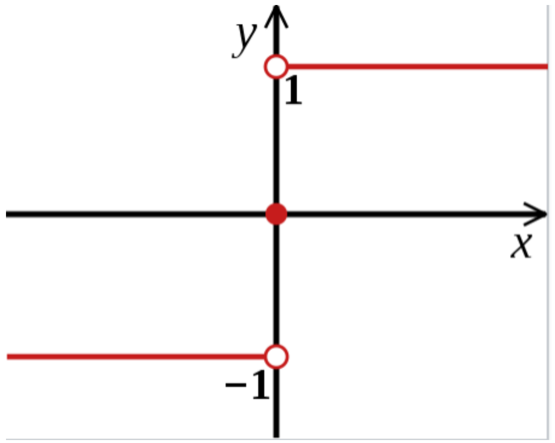
And we approximated it, using the following formula:

$$f(x) \approx f(x_0) + \sum_{i=0}^{N} [f(x_{i+1}) - f(x_i)] \left[\frac{1 + \mathrm{Sgn}(x - x_i)}{2}\right]$$

Approximating f(x)

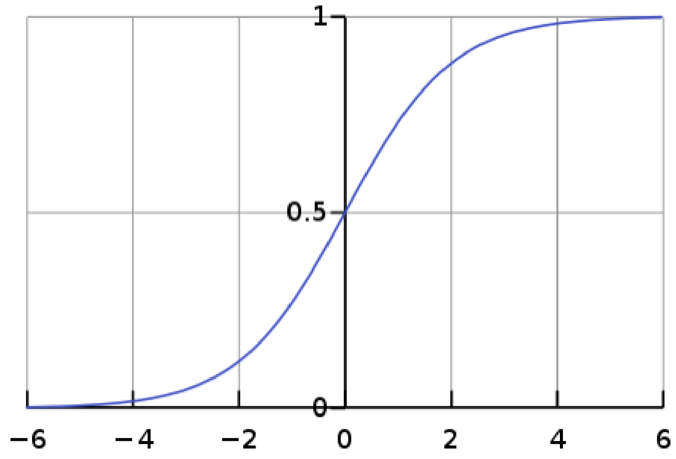
- A) using the signum function

$$\operatorname{sgn}(x) = \begin{cases} 1, \ if \ \ x > 0 \\ 0, \ if \ \ x > 0 \\ -1, \ if \ \ x < 0 \end{cases}$$



- B) using the sigmoid function (scaled, to be closer as representation to the signum)

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



To clarify …

Neural Networks for classification

What are neural networks?

- Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns.

What are they made of?

- The neural networks are composed of several layers. The layers are made of *nodes*.

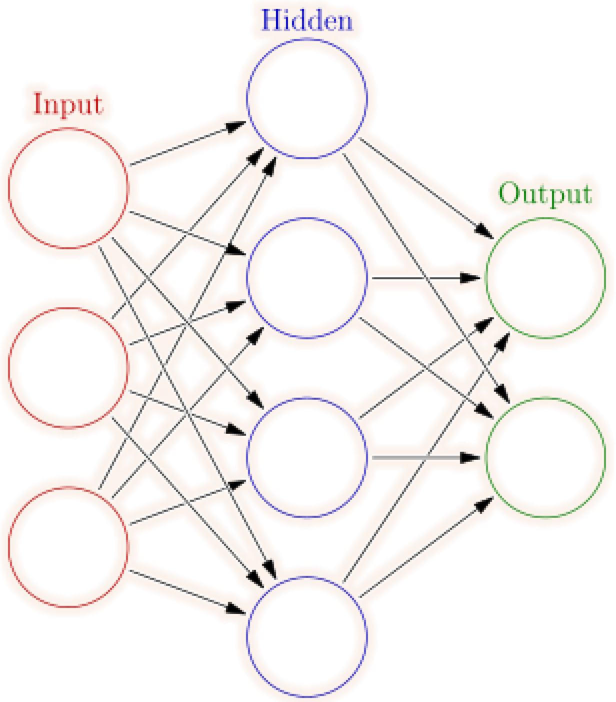
- A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs with regard to the task the algorithm is trying to learn

Neural networks architecture

- **Architecture**: **input** layer, **hidden** (nor input nor output) layer(s), **output** layer

- Each node (neuron) is a value. Each layer is a vector of values

- Each edge is a *weight* (used to compute the value of the next node)
- Each node (except nodes from the input layer) has a *bias*
- Information travels forward (from input to output), thus it's called a *feedforward* neural network
- Usually each node is connected with each node in the adjacent layers, resulting in fully-connected layers



What kind of tasks can a neural network do?

- Broadly speaking, they are designed for spotting patterns in data.
- Specific tasks could include:
  - classification (classifying data sets into predefined classes),
  - clustering (classifying data into different undefined categories),
  - and prediction (using past events to guess future ones, like the stock market).

How exactly do they “learn” stuff?

…To be seen next time

Thank you!