A primer on

# Artificial Neural Networks



- Please note that this material is protected by copyright
- Permission to copy this material is hereby granted to everybody, provided that:
  - The document is copied in its entirety and no parts of it are extracted and used in any form separately
  - Explicit reference to NeuriCam is made as the source of this material
  - The material is not sold, neither directly nor indirectly

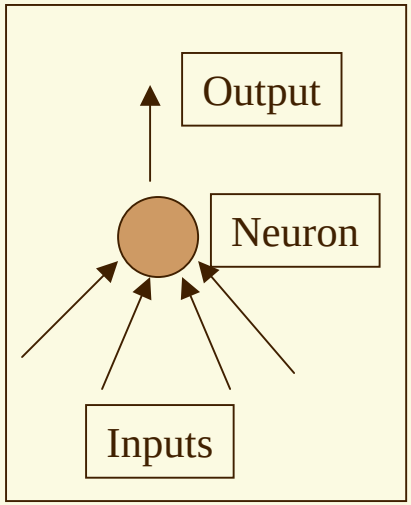
Web: www.neuricam.com
mail: info@neuricam.com
Phone: +39 (461) 260.552

A primer on

# Artificial Neural Networks



- Artificial Neural Networks (ANN) are a mathematical abstraction of their biological counterpart
  - Neurons are connected to each other by means of (unidirectional) connections (axons)
  - Neurons compute their activation as the weighed sum of their inputs
  - When the activation of a neuron exceeds a given threshold, the neuron "fires", producing an output whose value is mapping of its activation by a transfer function



# Characteristics of an artificial neuron

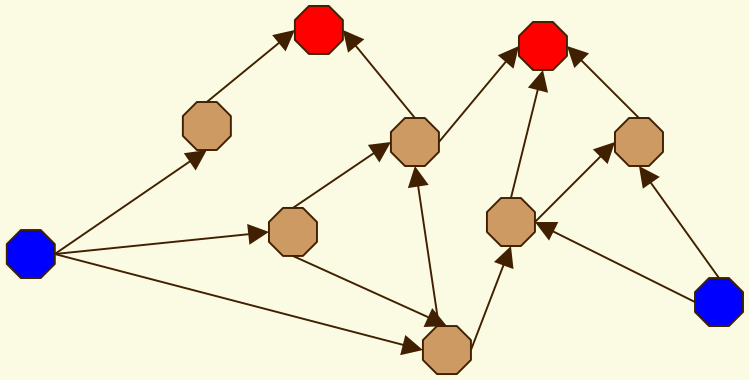
- Each connection from neuron $i$ to neuron $j$ has an associated weight $w_{ji}$
- Neuron $i$ with inputs $x_1, x_2, \ldots x_i$ produces an activation $a_i$

$$a_i = \sum_{j=1}^{i} x_j w_{ji}$$

- Neuron $i$ produces an output $y_i = f(a_i)$, where $f()$ is a transfer function
- Each neuron's threshold can be taken into account by introducing an additional fixed input (bias) = -1 and appropriate weight

# Building an ANN

- An ANN is a directed acyclic graph whose nodes are:
  - Input nodes: those without incoming edges
  - Output nodes: those without outgoing edges
  - Hidden nodes: all others

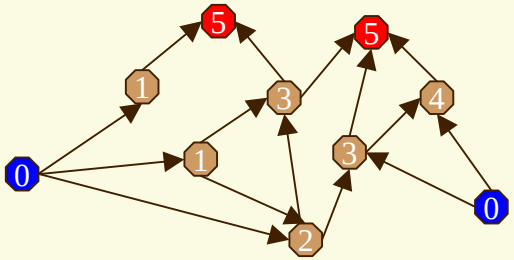


# Common simplifications

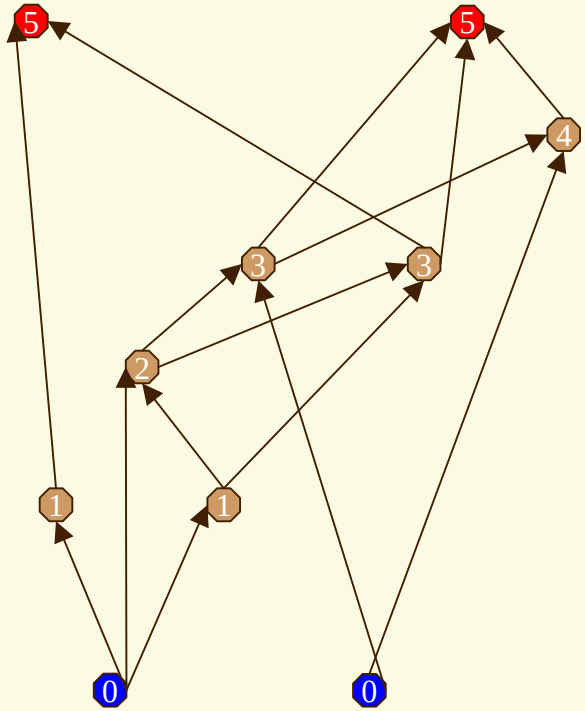
- It is convenient to draw the ANN as a layered structure, with:
  - the input nodes are placed all on the bottom layer (layer 0)
  - the neurons whose predecessors are at least on layer *l-1* are placed on layer *l*
  - the output nodes are placed all on the top layer (= max l)

# Example - redrawn

The previous example, with the nodes labelled as their respective layer




And the graph redrawn in layers



# Computing with ANNs

- By appropriate selection of the network topology and the values of the weights, an ANN can compute any computable function
- ANNs are a very powerful tool
- In addition: ANNs can learn from experience (more on that later on)

# A simple example:
## computing $y = XOR(x_1, x_2, x_3)$

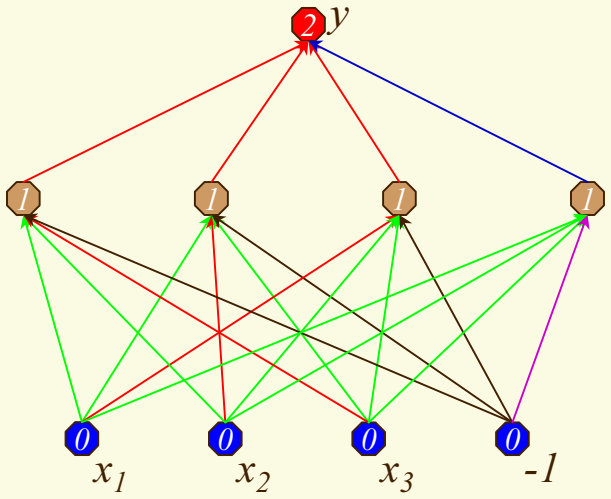
| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Truth table

Edge colors are used to denote weights

+1 0 -1

-2 -3



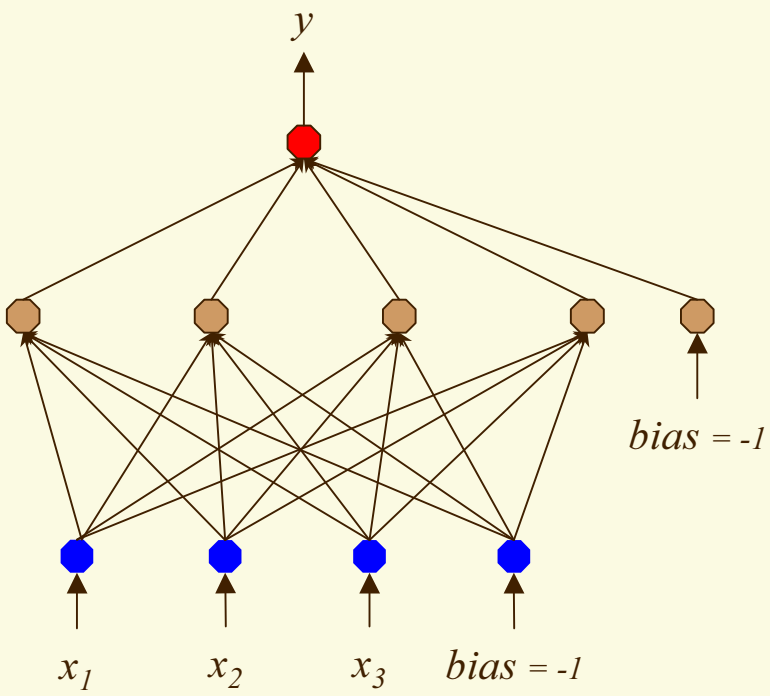

ANN

Where the transfer function $f(a) = \begin{cases} 0 \text{ if } a<0 \\ 1 \text{ if } a \geq 0 \end{cases}$

# More common simplifications and conventions

- It can be shown that any computable function can be computed by a three layer ANN: thus all that is needed is:
  - an input layer
  - a hidden layer and
  - an output layer
- Nodes from each layer are connected only to nodes of the next layer up
- Each layer can be considered fully connected to the next layer: edges that are not used, can be given a weight of zero

# Typical graphical representation of an ANN



# Computing with ANNs

- Consider a
  - specific network topology and a
  - specific weight assignment
- Let $X^{(t)} = (x_1^{(t)}, x_2^{(t)}, \ldots, x_m^{(t)})$ be a vector of input values and $Y^{(t)} = (y_1^{(t)}, y_2^{(t)}, \ldots, y_n^{(t)})$ be a vector of output values
- To a specific input vector $X^{(t)}$ corresponds a specific set of output values $Y^{(t)}$

# ANN as a mapping

- A particular ANN realizes a specific mapping from input $X^{(t)}$ to output $Y^{(t)}$
- This mapping can be changed by modifying the weights $w_{ji}$ associated with the edges of the network

- Given an input vector $X^{(t)}$ and the corresponding (desired) output vector $Y^{(t)}$, one can tweak the weights of a particular ANN until the (computed) outputs of that ANN are close to the desired outputs (it might be impossible to make the computed outputs the same as the desired outputs without changing the network topology as well)

# Error between desired and computed outputs

- Let $Y^{(t)}$ be the desired output vector and let $Y'^{(t)}$ be the output vector computed by a specific ANN
- Let $d(Y^{(t)}, Y'^{(t)})$ be some distance function that computes the "error" between desired and computed output vectors
  For example, a common choice for $d()$ is:

$$d(Y^{(t)}, Y'^{(t)}) = \sqrt{(Y^{(t)})^2 - (Y'^{(t)})^2}$$

- By changing the weights $w_{ji}$, one can attempt to reduce the value of $d()$ as close to zero as possible

# Extension to sets of inputs/outputs

- The process of weight adjustment can be extended to the more common situation where there is
  - a set of input vectors ($X^{(1)}, X^{(2)}, \ldots, X^{(t)}$) and the corresponding
  - set of desired output vectors ($Y^{(1)}, Y^{(2)}, \ldots, Y^{(t)}$) with
  - set of computed output vectors ($Y'^{(1)}, Y'^{(2)}, \ldots, Y'^{(t)}$)
- The error between desired and computed outputs will be given by $D(d(Y^{(1)}, Y'^{(1)}), d(Y^{(2)}, Y'^{(2)}), \ldots, d(Y^{(t)}, Y'^{(t)}))$
  (a common choice for $D()$ is the average for the various $d()$)

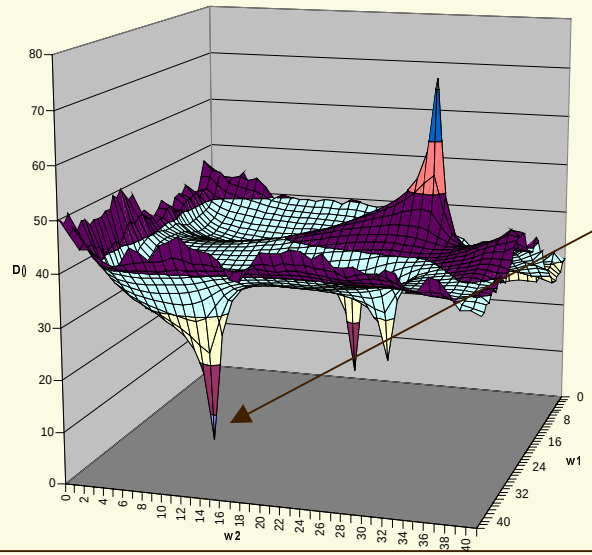
# Minimizing the error

- Given specific:
  - network topology,
  - set of inputs vectors and,
  - corresponding output vectors

  the objective is to search for a weight assignment that minimizes the error *D()*

# A graphical example

- Assume a given set of inputs and desired outputs;
  then, for a given network topology and for a particular choice of weights $w_{ji}$
  there is a corresponding set of computed outputs and its error *D()*
- For ease of visualization, assume that there are only two weights, $w_1$ and $w_2$
- On the vertical *(z)* axis the value of *D()*




At $w_1$=32, $w_2$=12, the error function *D()* has a global minimum.
(Other local minima can be seen elsewhere)

The question is
how to find the global minimum?

# Possible ways to find the weight assignment that minimizes *D()*

- Exhaustive search: Explore all possible values of the weights
  - Computationally prohibitively expensive!!!
- Steepest gradient method: start from a random assignment of the weights (a specific point in the *D()* surface), find the steepest direction of the *D()* surface around that point and move in that direction to the next point; repeat until further improvements are not possible
  - Frequently used, usually under the name “backpropagation”
  - Requires to compute the slope of the D() surface, a computationally difficult task
  - Might get stuck in a local minimum

# Finding the best weight assignment

- Search-based methods: start with a random weight assignment, compute the error function for some neighbours of that point, go to the best neighbour and repeat this process until no further progress is possible
  - Computationally simpler than the previous case, but requires fast re-evaluation of the entire set of input vectors for each neighbour
  - Can take advantage of a hardware implementation of the basic neurons
  - Might get stuck in a local minimum
  - Can be improved by adding a mechanism to “escape” from local minima
  - Can be improved by adding a mechanism to avoid retracing points that were evaluated before

# Reactive Tabu Search (RTS)

- A Primer on RTS can be found in the Docs folder of the NeuriCam's site. Here is a brief summery
  - Global search-based method
  - Avoids getting trapped in local minima
  - Keeps track of previous "moves" to avoid looping
  - Can work with short (few bit) representation of weights and inputs and achieves at least the same performance of more costly methods based on back-propagation
  - Does not require "smooth" transfer functions (it can work even with non-continuous functions)
  - "VLSI-friendly" technique: can be implemented in application-specific digital circuits for fast ANN evaluations (see NeuriCam's Totem chip)

# ANNs that learn

- The process of error minimization described earlier can be interpreted as a training of the ANN by presenting it with "examples" of inputs and the desired corresponding outputs
- Thus, the ANN can be thought of as going through a phase of supervised learning, where we provide it with the inputs and their corresponding expected responses so that later on it can provide reasonable responses to inputs that were not seen before

  (There are also non-supervised training techniques that use an objective functions that must be optimized)

# Things that need to be decided upon

- The topology of the network (typically, a balance has to be found between too many and too few nodes)
- The transfer function of the neurons (typically some sort of “S”-shaped curve)
- The size of the input/output training set

# How many nodes should be in the ANN?

- More nodes in each layer permit more accurate associations between inputs and outputs
- More nodes allow the net to learn the training examples very well (high memorization capability), but …
- but then the trained ANN can behaves poorly when presented with new inputs (low generalization capability)
- More nodes require more computation time
- There are techniques to give guidelines for selecting the appropriate number of nodes

# What kind of transfer function?

- Generally, some “S” shaped transfer function is used, such as: $\frac{1}{1+e^{-\alpha x}}$
- Typically, one tunes the transfer function by tuning the parameter $\alpha$
- The adaptability of ANNs makes the overall computation fairly tolerant of changes in the transfer functions, so this is not a major problem

# How many input/output vectors for the training set?

- In general, the more the vectors, the better the results
- Crucial is that the training vectors are statistically representative of the entire possible input universe
- however, the training methods are usually robust in the presence of spurious or ill-defined data in the training set (outlyers)

- The WinTot32™ software package provided a graphic-based user interface that simplifies the task of network configuration and testing. (See the WinTot32 primer on this site)