Computational Intelligence and Deep Learning

Computational Intelligence includes many topics:

- Neural networks
- Fuzzy systems
- Evolutionary computation



Neural networks can be used for many purposes:

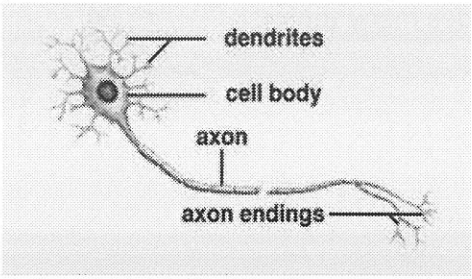
- Function fitting
  It is always possible to train a neural network able to approximate a function with a certain degree of accuracy. We just need to have training samples (pair <input, desired output>).
- Classification
- Clustering
- Forecasting
- Model approximate reasoning
  We can enable a system to draw a conclusion even in front of the presence of ambiguity/vagueness. This is typically performed by fuzzy systems.

We can always find a NN or a fuzzy system that solves a certain problem.
With a fuzzy system, there is no training process though, it writes down linguistic rules and has a higher degree of interpretability.
We can also think about a neuro-fuzzy system to have a trade-off between the advantages of the two. We provide the computer with a training set that will autonomously generate rules without expertise.
With evolutionary computation, we reproduce biological evolution. In some problems, we have no analytical and optimization method tool to solve a problem, while with this we can. We will go inside genetic algorithms that try to reproduce the typical way in which the human species evolves using three operators mimicking selection, cross-over, and mutation.
It uses fitness functions for each candidate of a population to identify a subset of them constating the best ones.
We go towards the second population of candidate solutions, on average better than the first one, and so on.
A genetic algorithm is an optimization algorithm that can be applied in any situation, and so it can be used together with NNs, FSs, or NFSs.
It can be used to optimize the parameters of these systems.

A subset of ML is Deep Learning, we train a NN in order to create an intelligent system.
In DL we do not need to perform feature engineering that will just need the training set and it will autonomously identify the necessary and sufficient features to reach the result needed.

Artificial neural networks

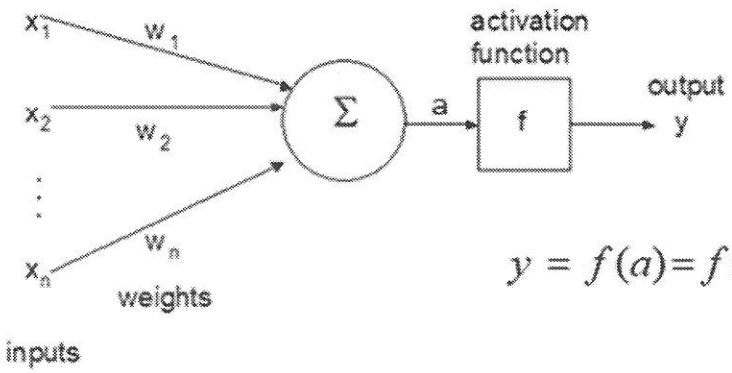
They try to reproduce the ability of neural networks to solve problems.



A biological neuron has:

- Dendrites (input channels), coming from other neurons or the external environment
- Cell body, 'soma', where the 'computation' takes place
- Axon (output)

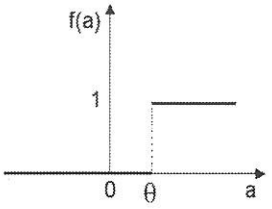
It receives electrical signals from other neurons that sum up, approximately, within the cell body and when the sum exceeds a threshold a signal will pass through the output channel.
The axon endings almost patch the dendrites, there is a small gap filled by the release of chemical substances (neurotransmitters) that bind to receptors in the receiving neurons.
Electrical signals arriving at the end of the axon, are transformed into chemical signals and then converted back into electrical signals at the end. We have an electrochemical process.
Synapses are the connection between axon endings and dendrites.
Depending on the type of synapse the signal can increase/decrease.



$$y = f(a) = f\left(\sum_{j=1}^{n} w_j x_j\right)$$

An artificial neuron has many inputs but only one output.
Each input connection has an associated adjustable weight (a real number). A positive (negative) weight has an excitatory (inhibitory) effect on the neuron. A transformation process can increase/decrease the potential of the signal, as we said.
A neuron computes some function f (called activation function) of the weighted sum of its inputs.
The output can be a real number, a real number restricted to some interval (e.g., [0,1] or [-1,1]), a discrete number (e.g., {0, 1} or {+1, -1}).
The activation function can be of different types:

- Binary threshold function

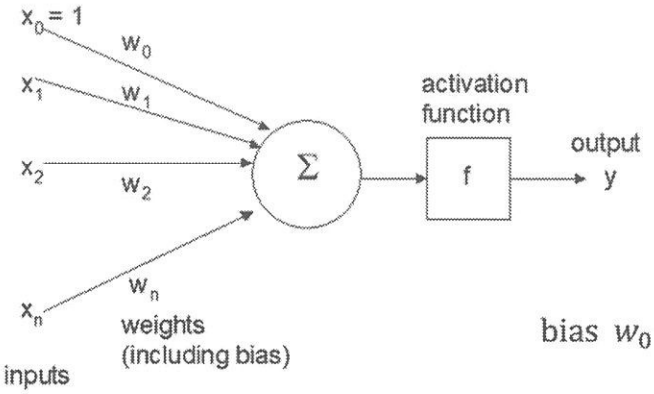


$$y = \begin{cases} 1 & \text{if } a \geq \theta \\ 0 & \text{if } a < \theta \end{cases}$$

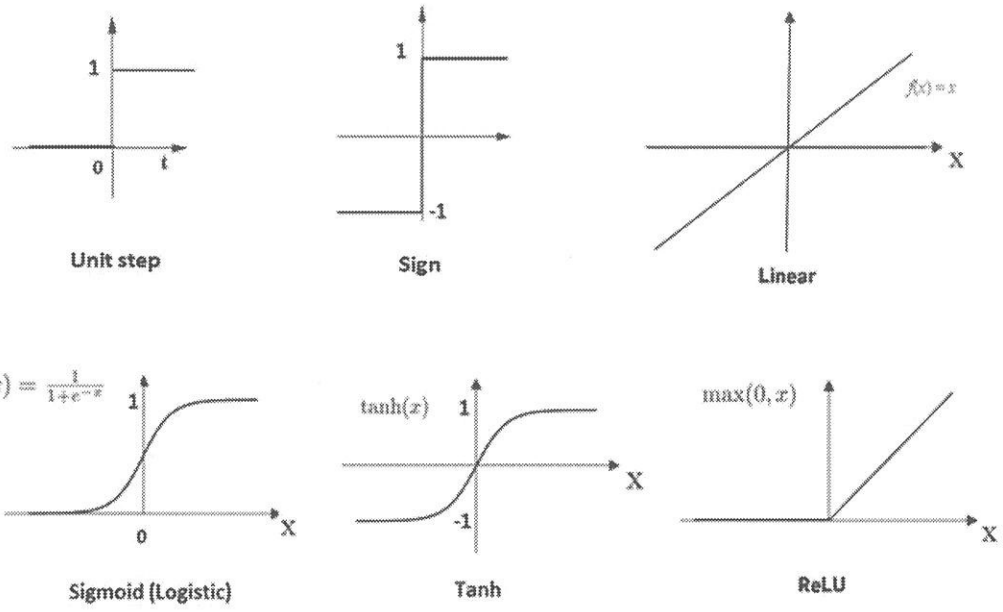
Typically, the threshold is subtracted from the weighted sum of the inputs:

$$y = f(\sum_{j=1}^{n} w_j x_j - \theta) = f(\sum_{j=0}^{n} w_j x_j)$$

and the negative of the threshold (called bias) is considered as a weight connected to a unit that always has an output of 1 (x0 = 1).



Different other activation functions can be used.



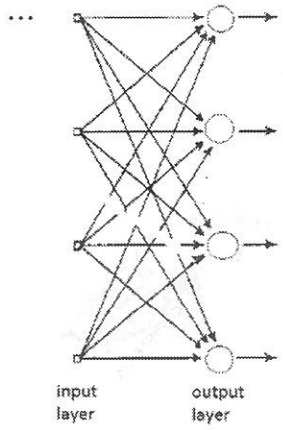
If we want to have a continuous set of values in the range [0,1] we can use other functions such as the sigmoid function or others.
In front of asymptotic behavior, with a particular slope parameter, the sigmoid function approaches the unit step function. If we need a differentiable function the logistic function can be used. It generalizes the unit step function.
We can also generalize the sign function using a tanh function.
For efficiency reasons, the most frequently used is the ReLU function.

There are many types of NNs depending on how neurons are connected to each other, an architectural graph will be used to represent it:

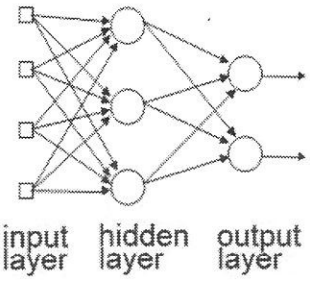
- neurons are arranged in an ordered hierarchy of layers in which connections are only allowed between neurons in immediately adjacent layers (most frequent topology),
- connections back from later to earlier neurons are allowed,

- each neuron can connect to every other neuron,
- ...

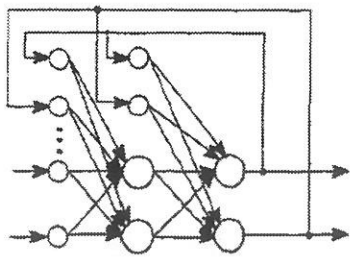
master COPY COPISTERIA



one-layer feedforward network



two-layer feedforward network



Recurrent network

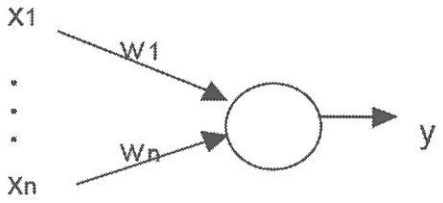
A different symbol for input neurons is used because they make no computation, they just let the input get inside the network without performing anything, using no activation function.
A network is feed-forward if the signal travels from the input layer to the following up to the output layer.
It is a one-layer FF network if just one layer performs a computation.
A hidden layer is a layer between the input and output layers.
Each neuron can also be connected to any other neuron in the network (we will have bidirectional arrows). It will be a fully connected not-layered network.
They are typically used to create an associative memory, in which we provide a part of the content of a cell to the network
We may also have a fully connected layered network if each neuron in a layer is connected to all neurons in the "following" layer, but not all other neurons. A network is said dense if this happens.
A layered network can be partially connected if each neuron is connected to a subset of neurons of the following layer.
A network will not be layered if a neuron is connected to a neuron that is not in the following layer.
Frequently, the weighted sum of inputs is referred to as the net input to the activation function, identified with net_i.
We represent with these architectural graphs the signal flow from neuron to neuron and we omit the signal inside a single flow, it has a different purpose than the graph with a single neuron.

Training
A popular learning paradigm is supervised learning, which changes the weights by applying a set of labeled training samples. Each sample consists of an input and the corresponding target (or desired) output. Each input sample is a pair.
The network is presented with a sample chosen at random from the training set and the weights are modified to minimize the error, i.e., the difference between the desired response and the actual response.
We randomly select a training sample, we show it to the network, let the network produce the output and use the error produced by the network, the difference between the desired response and actual response as a means to adjust weights.

The weights are the adjustable parameters of the network, the task of the training process is to properly adjust the values of the weights in order to produce an accurate answer.
The entire training set is presented to the network. The training is repeated until the network reaches a state with no further significant weight changes.
Therefore, the network learns from the training examples by building an input-output mapping for the problem under consideration.
We use an objective function that we want to minimize, and we evaluate a stopping condition.
We can use the error to correct the weights, we talk about error connection rules.

Delta rule

An example of an error correction rule is the delta rule (also called the Widrow-Hoff rule).



master COPY COPISTERIA

The error is $\delta = t - y$, where y and t are the actual output and the target output, respectively.
The delta rule states that the adjustment to be made is

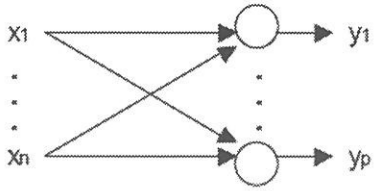
$$\Delta w_i = \eta \delta x_i$$

The term we add to the weight wi is proportional to the product of two terms delta and xi, where xi is the receiving signal of the neuron under analysis along the link associated with weight wi, where the real number eta is the learning rate, typically a real number between [0,1].
Therefore, the generic weight wi at time n+1 will be:

$$w_i(n+1) = w_i(n) + \Delta w_i(n)$$

This function suggests a default value automatically proposed, which depends on the fact that in most cases that value is an appropriate choice.
Parameters of the model will be the elements that can be changed by the training process, while hyperparameters are ones such as the learning rate, the number of hidden layers, the number of hidden neurons for each hidden layer, etc.
The size of the input and output layers are also to be chosen, we need to choose the input and output. Then, we need to decide how many hidden layers and for each how many neurons.
We also have to decide the activation function for every neuron, the learning rate, and others.

Let's prove the correctness of the delta rule.
Consider the following network:



- t_j = target (or desired) output
- y_j = actual output

It is an FF-layered network with only one input layer and one output layer.
The error made by the network for the k–th training sample can be expressed as:

$$E_k = \frac{1}{2}\sum_{j=1}^{p}(t_j - y_j)^2$$