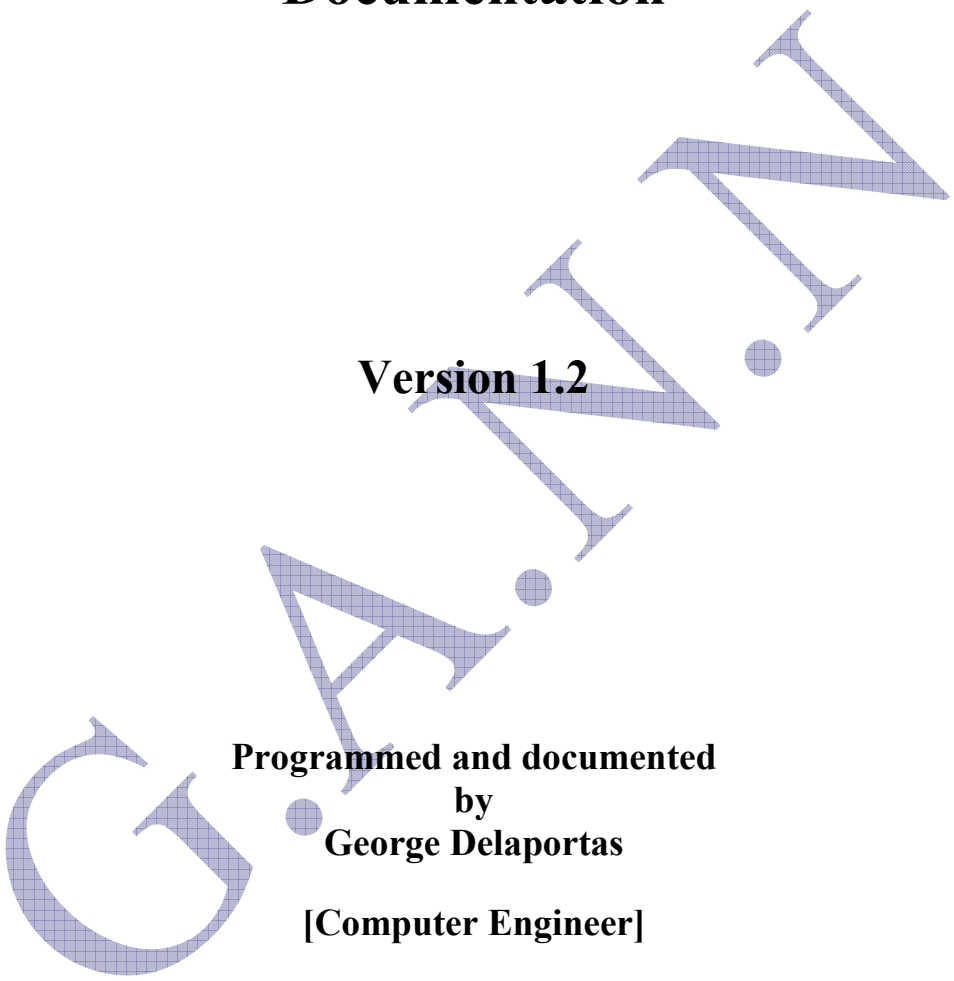
Geeks Artificial Neural Network (G.A.N.N)

Documentation



Version 1.2

Programmed and documented by
George Delaportas

[Computer Engineer]

Conformed according to the GNU/GPL

Prologue

Geeks Artificial Neural Network (G.A.N.N) is an extension to the classic model of Artificial Neural Networks Design. Almost all the procedures are fully compatible with the ones of the classic A.N.N.
G.A.N.N though, tries to fill the gaps and find solutions in subjects that the classic A.N.N can not. For that reason, many of the elements of the classic A.N.N do not fit or do not have any usage in G.A.N.N.
The new leading architecture and philosophy of G.A.N.N gives us more flexible ways to face most problems and together with its embedded control and optimization techniques makes it a powerful tool.
Furthermore, G.A.N.N, as we will see, is an integrated environment and not just another A.N.N.

Contents

Chapter 4

Chapter 1

1.1 Introduction

Artificial Neural Networks (A.N.N) are systems that help us find solutions and get results among a variety of problems. They can be successfully applied on everyday needs or even advanced industrial problems and they produce results rapidly and efficiently but in a much different way and philosophy than the classic ones do.

To understand the way that an A.N.N works we have to see the way that a real neural network works in the human brain.

The human brain uses elementary functional units. These elementary units structure the human brain and are interlinked with a complicated way, creating clusters. These functional units are named neurons and their clusters are named neural networks. Our neural network has to do with the artificial neural networks and of course with the artificial neurons. The A.N.N and the real neural networks do not have certain connectivity in space or in place. In the human brain billions of neurons are interlinked and cause the creation of enormous, not arranged in space or dimensions, neural networks. The result is a system which includes incredibly big crowd of small units that communicate and exchange information.
The power of A.N.N is in the fact that each neuron constitutes a small processor (functional unit) and finally with the connection of all these small "processors" we have an enormous in dimensions, distributed information system where the actions of every single unit is independent but also cooperative at the same time.
In other words we have a big distributed processor (computer).

Most A.N.N have important advantages that put them in the first place of preference for the solution of problems of general content. A very important and interesting problem that finds solution via the use of technology of the A.N.N is the noise. Noise is the word that describes all these parameters that are not legitimate and influence the results of a research, analysis or generally a process. The noise can have various forms as temperature, humidity, acoustic parasitises, radio parasitises and values of results that escape from certain formal expected distributions. For a real system the noise is something inevitable of course, but A.N.N are not so vulnerable to noise and this give them the ability of coping even with difficult environments and extreme changes. Another critical point that A.N.N have is that they produce results in polynomial time which is something very useful for problems that we want to be solved in short or reasonable time. A.N.N also have very important advantages that put them at the top of the choices for mathematical applications problems and digital signal analysis problems. When the mathematical analysis is difficult though, and we ask for direct results with precision A.N.N have the ability to withhold the processes that will take part and just give us the results.
So in other words we are given a frame of abstractive reality where the only important thing for us is the input and the output of the system.

Of course, A.N.N, as any other system, don't have only advantages but also important disadvantages. The classic A.N.N do not include processes to control the data at the input level and the output level respectively. So the data that flow through the network have not been checked and they may not be acceptable for the particular problem. Another big problem is the capacity of a classic A.N.N. Many of the well known A.N.N up to today have been developed to satisfy fixed, in size and type,

problems and are not adapted dynamically and so they do not give us optimal results. So according to the problem needs, the programmer or the user is forced to describe the explicit elements and has to be determinant for the optimal solution. In other words the user of each A.N.N is necessary to find with trials the explicit type of the A.N.N for each problem. This considerably complicates the process of construction!

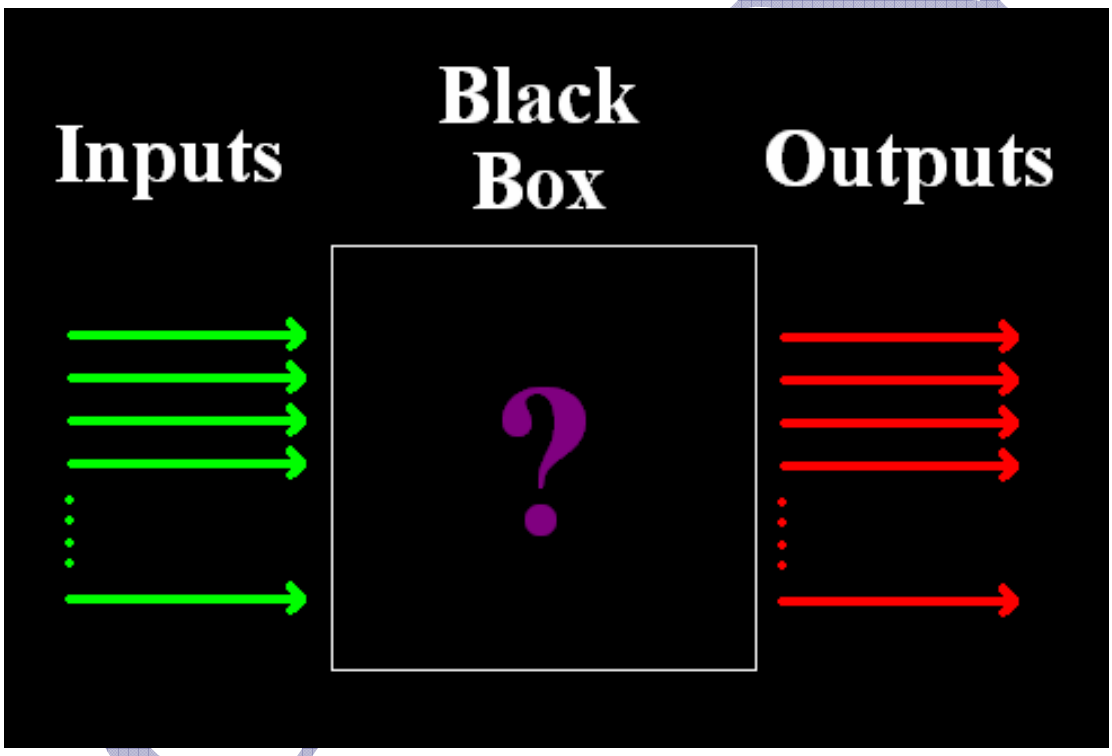
G.A.N.N gives solutions in a lot of problems and advances in a new dimension where its high adaptability might easily give us the ability of coping with multidimensional problems.

The ways and the techniques will be analyzed in the chapters that follow. We will also do extensive analysis and comparison with the already known techniques.

Chapter 2

2.1 G.A.N.N architecture philosophy

The philosophy underneath the architecture of G.A.N.N is based on a classic model that we use to describe complex systems, the «Black Box». The Black Box [2.1.1] is an abstract model of the reality which helps us describe these systems.
The most common usage of the Black Box is at applied mathematics and digital signal analysis. The result of its usage is that we only look the inputs and the outputs without the need to know the inner space! In other words we do not care for its inner architecture and furthermore we do not need to know how it works!
The big bet that G.A.N.N tries to win is to achieve to hide the way that different procedures take place in the Black Box and so to create a layer over which anything can be developed. In other words G.A.N.N tries to be a centralized and unified environment, a core, which can handle all the procedures of higher levels!
Even simpler, G.A.N.N can change its inner architecture dynamically so as to achieve the best results for different type of problems!



«Black Box» model (2.1.1)

To achieve all these things, G.A.N.N should be clever enough and well structured. This is something that I had in mind from the very first minute and so I tried to keep things simple and fast.

The source code of this project is well organized and is conformed to GNU/GPL and every piece of code is written according to ISO C++ for maximum stability, compatibility and speed.
The code is structured based on the modular programming pattern and so all the entities are modules and nearly every function is written in a way that can easily be called by the system.

2.2 Functionality

It is very important at this point, after the theoretical introductive frame, to see concisely the way that G.A.N.N operates and how it is modelled. G.A.N.N, as we saw, is well structured and that is the reason why it is modularized in parts. The model of G.A.N.N is constituted by three basic parts (sub-models):
a) the inputs socket
b) the A.N.N
c) the outputs socket

The I/O sockets are two virtual sockets that may remind you of the SCART plugs [2.7.1] and their aim is to unify the imports or exports of different variable types given respectively. Each one of the sockets is also modelling the Pins, aiming to manage the data that strictly has been determined!

The A.N.N [2.7.3] is the core or if you like the brain of the "System" and it is charged with the responsibility of the processes that will take part for the production of the results. The A.N.N encloses in its inner the basic structural elements, the artificial neurons.
The artificial neurons are also structured of three parts:
a) the input slots
b) the main body
c) the output (outputs slot)

The input slots are similar to the sockets but with the difference that their values can be of any type. The input slots are the equivalent of the synapses of a real neuron. To the values of the slots, numbers, named weights, are applied and influence the final results of the artificial neuron.
The main body or body [2.8.1] is the basic part of the artificial neuron and is the part in which all the processes of activation are held. The body of the artificial neuron is separated into two parts:
a) the adder
b) the function that produces the results

The adder multiplies the weights with the input slots values, adds the multiplied numbers and finally produces a factor which is called sum. The result that a neuron produces is only one each time but the functions that can be called are three:
a) the step function
b) the sign function
c) the sigmoid function

I have to note here that whatever the neuron will produce depends on a number that is called base. The base is calculated with some way, which I will explain later, and functions as a threshold which if it is not exceeded it does not return a result or gives reversed or opposite results opposed to the normal ones. The output is the same as the axon of a real neuron and routes the produced information from the body of the neuron to the other neurons in the network, according to the topology and the connectivity of the A.N.N. With a few words, G.A.N.N takes data, processes them with certain ways, which are not obvious to the user, and produces the desirable results. It's a very simple, convenient and reliable model too!

2.3 Problems

The most important points that we have to check for an A.N.N so as to accomplish our mission are five:
a) the learning algorithm that we have to use
b) the structure of the A.N.N that we need to use
c) the topology and the way that the neurons will be connected
d) the correctness of the A.N.N
e) the output of the A.N.N

I will try to show you how and with what cost G.A.N.N gives us solutions to the above problems.

2.4 Proposals and ideas

Let's propose something different. My aim is to easily and rapidly cover, structured around the usability, the above problems.

So, we need:

a) black box architecture
b) I/O interfaces
c) automated procedure that constructs an A.N.N
d) adaptive mathematical models

According to the above models and to the references of section 2.1 G.A.N.N will try to solve the problems introduced in section 2.2.

2.5 Solutions

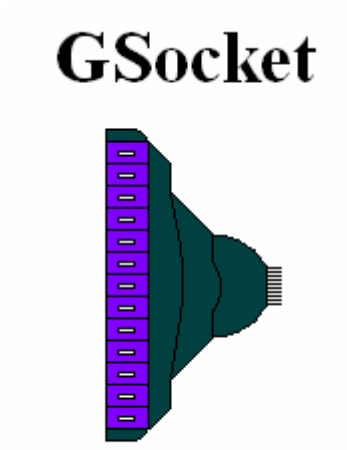
The solutions that result from the way that G.A.N.N is organized are too satisfactory. What really is an important thing though is the fact that we do not need either to construct the A.N.N, either to optimize it ourselves! These procedures are automated. Furthermore, G.A.N.N outputs will be checked and analyzed in depth to chapter 4.

2.6 Basic elements of G.A.N.N

In the rest of this chapter we will have an in depth analysis of G.A.N.N elements so as to make clear the reasons why the chosen models where selected.
We are also going to explain the procedures that take place in each one of them.

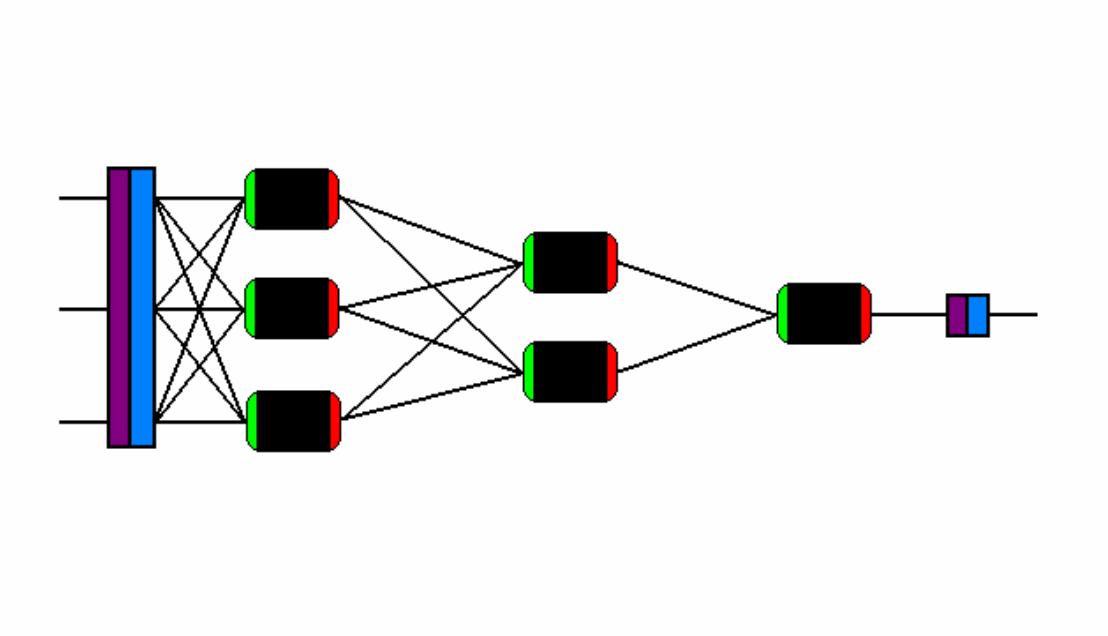
2.7 The GSockets

The I/O GSockets [2.7.1] route data from the input to the A.N.N or route data from the A.N.N to the output respectively. As we have already mentioned these sockets check the data so as to make sure that anything that passes though them is what has previously been defined and that nothing is uncertain, more or less and also integrate checks for the values that were defined for a specified range. These sockets are active «tools» that help us administrate and manage the system.
I would like to add a comment here just for the inputs sockets. The input sockets raise the complexity of the A.N.N while at the same time do not leave any neuron to be a passive element to the routing procedure. In other words, opposed to the classic A.N.N, G.A.N.N has 100% utilization of its artificial neurons. That is happening because in contrast with the classic A.N.N, G.A.N.N is transferring the I/O levels outside the actual network where the sockets exist. Furthermore, this means that the classic input and output levels do not exist in G.A.N.N at the same way as at the classic A.N.N.
The photos that follow show you a GSocket and how these GSockets are used [2.7.2] in G.A.N.N.



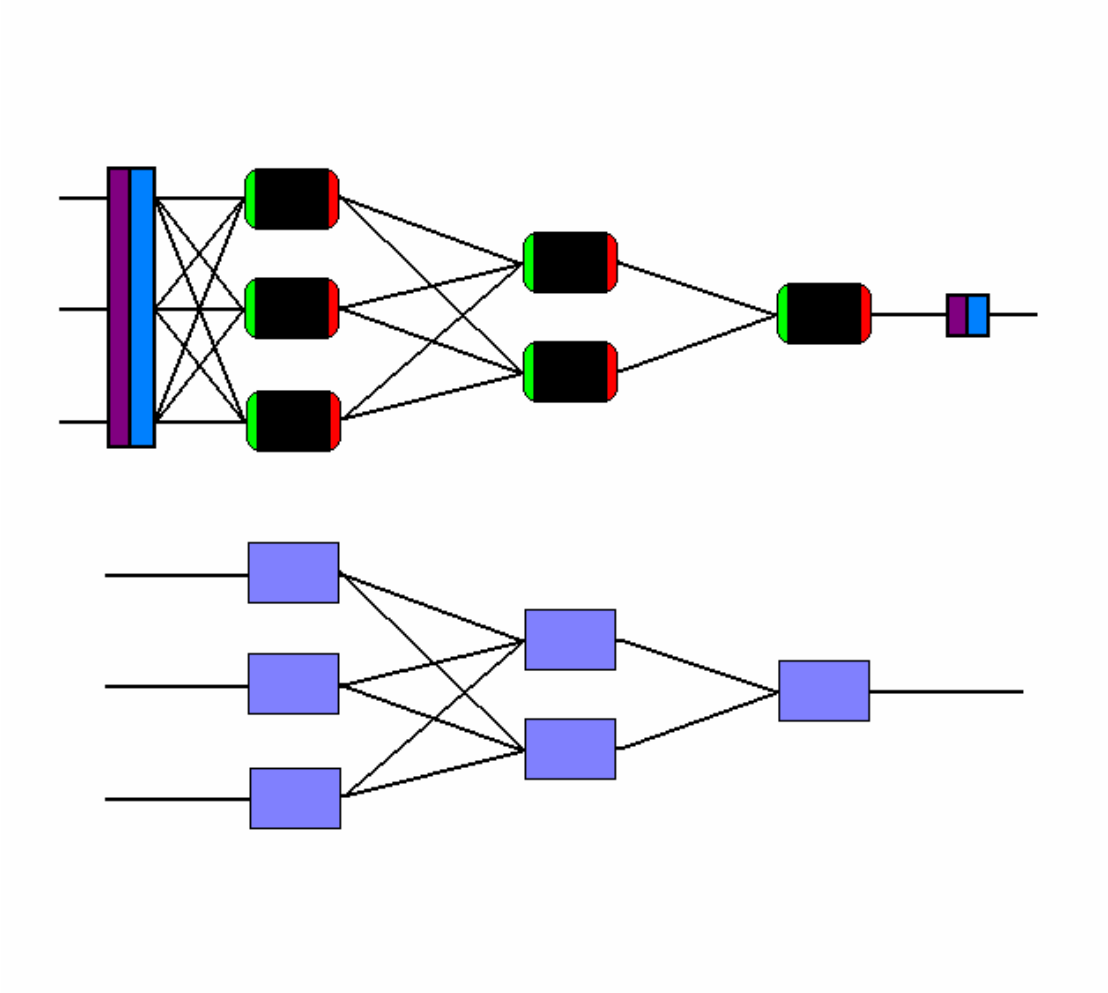
I/O GSocket model (2.7.1)

A typical socket may have theoretically limitless number of I/O and this can only be limited by the characteristics of the computer that runs G.A.N.N or from the application needs.



A typical G.A.N.N (2.7.2)

The GSockets are applied at the data entry level and just before the first neurons and before the final outputs.
You may see below the figure that compares a classic A.N.N with G.A.N.N so as to see the differences.



Comparison between G.A.N.N and a classic A.N.N (2.7.3)

As you can see from the picture above, the classic A.N.N has wasted three of its neurons as inputs. This means that these three neurons do not take part to any procedure in the network. They just route 3 inputs and they still exist there so as to consume memory and C.P.U cycles! Also the complexity has lowered and we may need to use many more neurons so as to produce the same results with G.A.N.N. This means much more C.P.U cycles and R.A.M consumption. Furthermore, the classic A.N.N do not check the data at the input or output level, just like G.A.N.N does with GSockets, and so it doesn't predisposes problems that might arise.

2.8 The GNeuron

As we have already seen the GNeuron is composed of three parts:

a) the input slots
b) the body
c) the output (outputs slot)

The input slots get data from other GNeurons or from the system data entry.
The input slot is automatically being created when data start to flow into the GNeuron and they do not exist by default. This technique gives us one important advantage. We have a great R.A.M management and we keep C.P.U utilization at very low levels! In other words, we have a dynamic, On-Demand architecture. This architecture covers more than 80% of the construction of G.A.N.N. The opposite philosophy has been used for the output o each GNeuron which we know that is always one. The output is just a passive wire that transmits information that was produced by the GNeuron.

The body is the most important part of the GNeuron and consists of two sub-parts:

a) the adder
The adder multiplies the weights, numbers that are produced by a random number generator, with the input slots values, adds the multiplied numbers and finally produces a factor which is called sum.

After the sum is produced another factor is used, the base. The base is a number that acts as a threshold for the final result of the GNeuron output. The base is computed by dividing the sum with the number of the inputs.

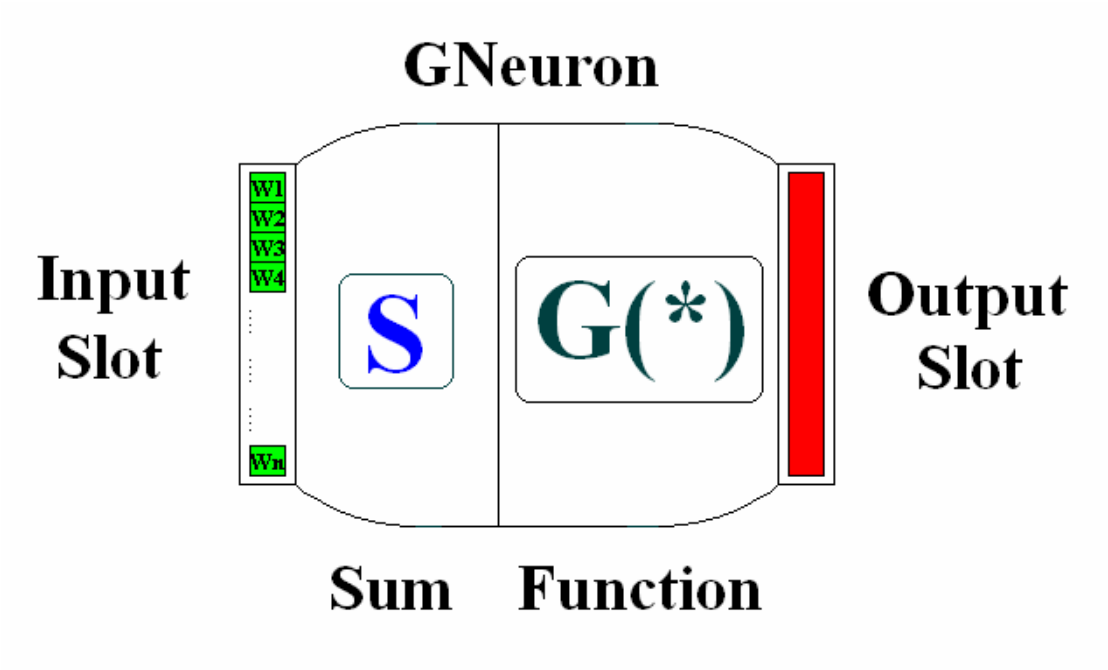
b) the activation function
The activation function produces the final value.

We usually use the three functions that follow:

a) the step function
The step function returns 1 if sum is bigger than base or else 0.

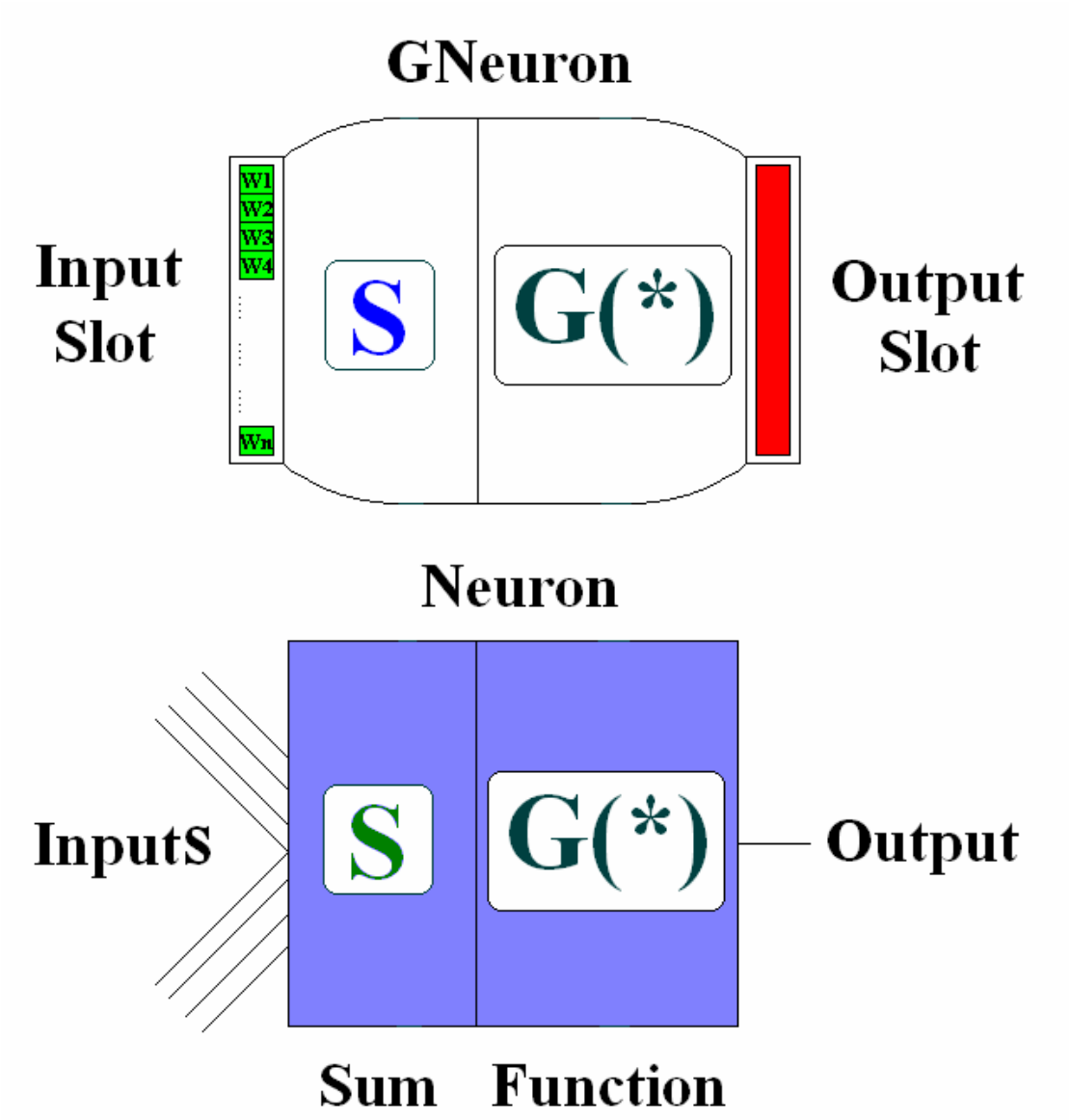
b) the sign function
The sign function returns 1 if sum is bigger than base or else –1.

c) the sigmoid function
The sigmoid function returns values in range of 0.0 to 1.0 if sum is bigger than base or else 0.0 to –1.0.



GNeuron model (2.8.1)

The picture above shows the internal architecture of the GNeuron.

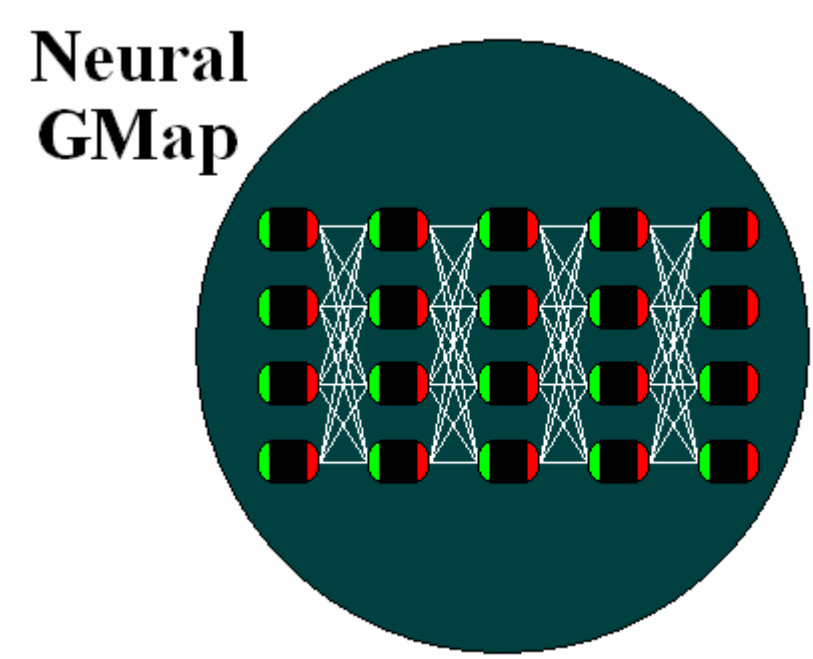


Comparison between GNeuron model and classic neuron model (2.8.2)

To the picture above we can see the differences in the architecture of the GNeuron model and the classic one.

2.9 The neural network

The A.N.N, as we have already mentioned, is the brain of the system and is responsible for the procedures that take place for the production of the results.
The A.N.N routes data among the neurons and is managed by the simulator.
The simulator instructs the A.N.N how to manage and route the data.
The A.N.N does not check the data that flow through itself.



The neural GMap (2.9.1)

The picture above depicts a typical Full Connected G.A.N.N.

Chapter 3

3.1 Explanation and analysis of other elements

In this chapter we are going to see the other elements of G.A.N.N, the passive ones, which are not implemented in code and just help us with the procedures.
These files are:

01. GKDB:	Geeks Knowledge Data Base (Auto Generated)
02. GMap:	Geeks Map (Auto Generated) [Temporary]
03. GIOD:	Geeks I/O Distribution (Auto Generated) [Temporary]
04. GHT:	Geeks Hash Table (Auto Generated)
05. GRegs:	Geeks Registrations (Auto Generated)
06. GSims:	Geeks Simulations (Auto Generated)
07. GANN:	G.A.N.N Binary File (Auto Generated)

3.2 Geeks Knowledge Data Base

The GKDB stores all the registrations and important information of G.A.N.N.
The GKDB is a database that developed only for G.A.N.N and its architecture is optimized for fast read/write.
You can see the inner structure of GKDB below:

Registration serial number

---------- Header ----------
Simulations number
Error correction type
G.A.N.N Simulation type
GNeurons connections type
Learning threshold
Total GNeurons number
Levels
Inputs
Outputs

---------- Levels ----------
GNeurons in this level
……

---------- GNeurons ----------
Pins of input slots
Inputs weights
GNeuron function type

Division modulo
……
……
……
……

---------- Outputs ----------
Minimum values
Maximum value
……..…..
……..…..

3.3 GMap

The GMap or neural map is a temporary file that helps us in some procedures before the learning state. When GMap is loaded in memory the file that contains it will be deleted and the learning procedure will begin.
You can see the inner structure of GMap below:

---------- Header ----------
Simulation number
Error correction type
G.A.N.N simulation type
GNeurons connections type
Inputs
Outputs
Threshold
Levels

---------- Levels ----------
GNeurons in this level (O,D,A)
……..……

GNeurons

Finally the simulator chooses among of three design types each time:

O: Orthogonal
D: Delta
A: Anadelta

3.4 GIOD

The GIOD stores information for the inputs and the outputs of G.A.N.N temporarily. When GIOD is loaded in memory the file that contains it will be deleted and the learning procedure will begin, just like the GMap.
You can see the inner structure of GIOD below:

---------- Header ----------
Inputs
Outputs

---------- Inputs ----------
Socket type
Pin type
Socket pin state
Minimum pin value
Maximum pin value
……
……
……
……
……

---------- Outputs ----------
Socket type
Pin type
Socket pin state
Minimum pin value
Maximum pin value
……
……
……
……
……

3.5 Geeks Hash Table

GHT file stores the offsets of the registrations of the GKDB so that we know with only one pass the exact position of each registration any time!
You can see the inner structure of GHT below:

Registration serial number
Registration offset
……
……

3.6 Geeks Registrations file

The registrations file stores the serial numbers of the registrations of GKDB.
You can see the inner structure of GRegs below:

Registration serial number
……

3.7 Geeks Simulations file

The simulations file stores the simulation number of each registration of GKDB.
You can see the inner structure of GSims below:

Registration serial number
Simulations number
……
……

3.8 The executable

The executable of G.A.N.N runs on, almost, any UNIX Like and Windows system and is compiled with the best compiler on the world, gcc (Front-End:g++) with full optimization switch enabled (-O3).

Chapter 4

4.1 Mathematical models

G.A.N.N is described by mathematical models which are divided into 4 categories. The first category describes the GNeuron functions, the second one the mapping and optimization process (for the given neural map), the third describes the routing process and the fourth the learning - error correction mathematical models.
I have to say at this point that almost all the models of G.A.N.N are compatible with the ones that Zhang and Lee have introduced.
Before we start explaining these models we have to see the general variables.
So, where:

a) x, is the given GNeuron
b) m, is the level of G.A.N.N (M : maximum number of levels)
c) $n-1$, shows the GNeurons per level
d) $k_x - 1$, is the number of inputs for the GNeuron x
e) j, is the GNeuron inputs counter

4.2 GNeuron functions mathematical models

First of all we will see the GNeuron functions and their results.

$S[x]_0^{n-1} = \sum_{j=0}^{k_x - 1} \left(P[j] \bullet W[j] \right)$ (4.2-1) is the function that produces a factor called sum.

The value of sum is produced by summing up a set of inputs ($P[j]$) that were multiplied with the weights ($W[j]$) respectively.

$B[x]_0^{n-1} = \frac{S[x]_0^{n-1}}{k_x - 1}$ (4.2–2) is the function that produces a factor called base.

The value of base is produced by the division of the sum with the inputs ($k_x - 1$) of the specified GNeuron.

So, $G\left(S[x]_0^{n-1}, B[x]_0^{n-1}, choice \right)$ (4.2–3) is actually the generalization of those three mathematical models, enclosed in one classic multi-variable tree function.

$$G\left(S[x]_0^{n-1}, B[x]_0^{n-1}, choice\right): \begin{cases} G_1\left(S[x]_0^{n-1}, B[x]_0^{n-1}, 0\right) = \begin{cases} 0, S[x]_0^{n-1} \le B[x]_0^{n-1} \\ 1, S[x]_0^{n-1} > B[x]_0^{n-1} \end{cases} \\ \acute{\eta} \\ G_2\left(S[x]_0^{n-1}, B[x]_0^{n-1}, 1\right) = \begin{cases} -1, S[x]_0^{n-1} \le B[x]_0^{n-1} \\ 1, S[x]_0^{n-1} > B[x]_0^{n-1} \end{cases} \\ \acute{\eta} \\ G_3\left(S[x]_0^{n-1}, B[x]_0^{n-1}, 2\right) = \begin{cases} -1.0...0.0, S[x]_0^{n-1} \le B[x]_0^{n-1} \\ 0.0...1.0, \acute{A}\nu S[x]_0^{n-1} > B[x]_0^{n-1} \end{cases} \end{cases}$$

Which function is going to be executed each time is relative to the variable called *choice* and what the result will be is relative to which are the values of sum and base.

4.3 Mapping mathematical models

This section explains the ways and methods that G.A.N.N uses to construct and optimize the neural map. It is very important to understand that the neural map is a part of the optimizer and the optimizer is always functional even if it will never be used for an optimization, but it is always checking the state of G.A.N.N!
So, let's see the functions.
$fns = nrs - (ins + outs)$ (4.3-1) tells us the number of the «free» GNeurons, GNeurons that won't be used for I/O processes. The *nrs* variable tells us the number of the GNeurons that were chosen by the user-programmer during the process of the creation of the neural mapping. The *ins* and *outs* are the selected, by the user, inputs and outputs respectively.

After that, we have to do a fast check: *If* $fns = 0 \Rightarrow fns = 1$.
This check should always be done because this way we are sure that in G.A.N.N there will always be at least one GNeuron after the process of optimization. That happens because the reason of the optimizer is to lessen the GNeurons so as to achieve less complexity and better speed. Sometimes though, the usage of more GNeurons is necessary. The optimizer knows that and when needed tries to maximize them.

$lrs = \dfrac{nrs}{(ins + outs)}$ (4.3-2) is the function that computes the number of the levels into G.A.N.N after the optimization process.
All the models above are just computations that are useful to the mapping models that we will see and explain.
The mapping models do not just construct the G.A.N.N but also optimize it. These models have to do with the crowd of the GNeurons and the shape they should construct relative to what output we want to achieve!
I have to note here that G.A.N.N is probably the first A.N.N on the world that optimizes A Priori, which means during its construction – mapping and before the learning and running state!

Let's move on with the models.

$O(z)=\left\{\begin{matrix} fns, 0...lrs-2 \\ outs, lrs-1 \end{matrix}\right\}$ (4.3-3) is the function called «Orthogonal».

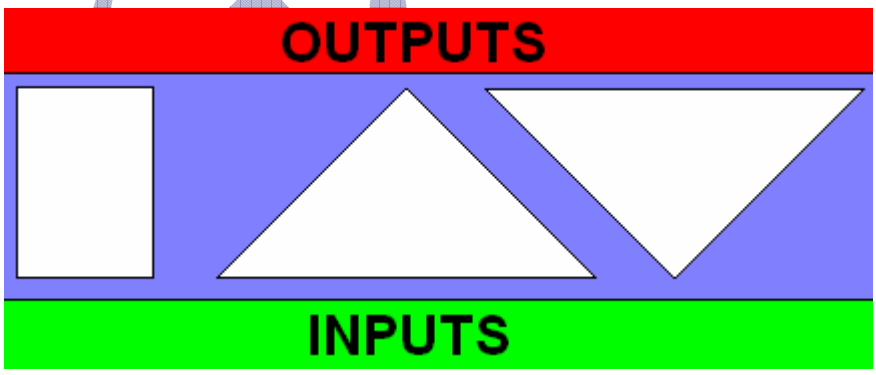
Orthogonal produces an optimized neural map that has an almost orthogonal shape. When the I/O complexity is big enough this shape is automatically selected. According to the function, until the pre-final level the GNeurons number is equal to the free ones in the network but at the last level their number is equal to the number of G.A.N.N outputs.

$D(z)=\left\{\begin{matrix} \frac{fns}{lrs}+1, 0...lrs-2 \\ outs, lrs-1 \end{matrix}\right\}, lrs+1$ (4.3-4) is the function called «Delta».

Delta produces an optimized neural map that has an almost triangular shape.
When the inputs complexity is big enough and the outputs not, this shape is automatically selected.
According to the function, until the pre-final level the GNeurons number is equal to the computation in the model but at the last level their number is equal to G.A.N.N outputs. I have to note that if we want to achieve the triangular shape the levels are increasing so that the fraction becomes smaller.

$A(z)=\left\{\begin{matrix} \frac{fns}{lrs}+1, 0...lrs-2 \\ outs, lrs-1 \end{matrix}\right\}, fns+1$ (4.3-5) is the function called «Anadelta».

Anadelta produces an optimized neural map that has an almost triangular shape which is upside down.
When the outputs complexity is big enough and the inputs not, this shape is automatically selected.
According to the function, until the pre-final level the GNeurons number is equal to the computation in the model but at the last level their number is equal to G.A.N.N outputs. I have to note that if we want to achieve the upside down triangular shape the free neurons are increasing so that the fraction becomes bigger.
You have to keep in mind the figure that follows so as to remember graphically the process of how G.A.N.N constructs and optimizes itself.



Shapes that the neural map optimizer produces (4.3.1)

$ons = \sum_{0}^{lrs-1}\left[O(z) \;\; or \;\; D(z) \;\; or \;\; A(z) \right]$ (4.3-6) is the function that keeps the final number of the optimized GNeurons in G.A.N.N.
GNeurons number is produced by the sum up of all the outputs of the called optimizing functions $O(z)$ or $D(z)$ or $A(z)$.
The upper limit of the sum is levels number (lrs).

4.4 Routing mathematical models

The mathematical models that we are going to see in this section are the typical for a Full Connected G.A.N.N.
Let's analyze them:

$$Y_m[x]_0^{n-1} = G\left(S[x]_0^{n-1}\right)$$

$Y_m[x]_0^{n-1}$ (4.4-1) is the function that computes the output values of every GNeuron of the same level (m).

$$E[x]_0^{n-1} = Y_M\left[Y_{M-1}\left[Y_{M-2}[...]_0^{n-1} \right]_0^{n-1} \right]_0^{n-1}$$

$E[x]_0^{n-1}$ (4.4-2) is not exactly a function but an array that contains the values of the final results of G.A.N.N outputs.
The reason why this model exists is to show you how the routing process works. In a few words the information is traveling from the first level and all the outputs from this level becomes input for the next level and so on. This way, we finally get the information at the output.

4.5 Learning – error correction mathematical models

Back Propagation Error is a classic widely used and well known algorithm that minimizes the output error by getting it back and changing the weights with the help of the least squares mathematical model.
This way the A.N.N is learning really fast but it has three big disadvantages:

a) a quite big computational time
b) R.A.M consumption for the reason that many arrays are used
c) minimized randomness

Especially the third one is the biggest disadvantage. We should, in no way and for no reason set borders to the choices of the weights to an A.N.N. The weights should be quite or really random if we want to have better results.
These sections models are not compatible with the classic ones. That's because these models are optimized for high precision randomness (better weights), higher C.P.U utilization (higher speed) and less R.A.M consumption.
Let's analyze these models and see how they work.

$C(S[x]_0^{n-1}, B[x]_0^{n-1}, err, check)$ (4.5-1) is a classic multi-variable tree function.

$$C(S[x]_0^{n-1}, B[x]_0^{n-1}, err, check): \begin{cases} C_1(S[x]_0^{n-1}, err, 0) = \begin{cases} Random(err), S[x]_0^{n-1} \le B[x]_0^{n-1} \quad and \quad err \ge 25.0 \\ \acute{\eta} \\ Random(100.0 - err), S[x]_0^{n-1} \le B[x]_0^{n-1} \quad and \quad err < 25.0 \\ \kappa\alpha\iota \\ Random(err), S[x]_0^{n-1} > B[x]_0^{n-1} \quad and \quad err \le 25.0 \\ \acute{\eta} \\ Random(50.0 - err + 1), S[x]_0^{n-1} > B[x]_0^{n-1} \quad and \quad err > 25.0 \end{cases} \\ \acute{\eta} \\ C_2(S[x]_0^{n-1}, err, 1) = \begin{cases} 2.0, S[x]_0^{n-1} \le B[x]_0^{n-1} \\ -2.0, S[x]_0^{n-1} > B[x]_0^{n-1} \end{cases} \end{cases}$$

$C(S[x]_0^{n-1}, B[x]_0^{n-1}, err, check)$ has two ways of facing the errors. Statistically and statically. The choice is made by the value of the control variable *check* .

In the first case when *check* has the value 0, $C_1(S[x]_0^{n-1}, err, 0$ produces $Random(err)$ or $Random(100.0 - err)$ or $Random(50.0 - err + 1)$ as outputs, if the sum ($S[x]_0^{n-1}$) is lees - equal or bigger than the base ($B[x]_0^{n-1}$) and analogous to the error of each GNeuron, the final result is produced.
The philosophy of this model is very simple!
What we want to achieve is to lessen or to maximize, under certain circumstances, the GNeurons output. So, our purpose is to check, based on the output (See Chapter 2, Section 2.8) how far or close we are from the desirable result. Finally by getting as a second criterion the error percentage and by watching whether this percentage is bigger or smaller than 50% we have the ability of showing to the GNeuron what path to follow, so as to produce the desirable output. This technique does not fixate the selected weights and the final results are much better. Because we have to check for four states we split the possibilities space into four peaces of 25%.
We have to point that *Random* is a system function and produces random (pseudo-random) numbers.
In the second case things are simpler. When the control variable *check* has the value 1, $C_2(S[x]_0^{n-1}, err, 1)$ produces 2.0 if $S[x]_0^{n-1}$ is less - equal than $B[x]_0^{n-1}$ or -2.0 if $S[x]_0^{n-1}$ is bigger than $B[x]_0^{n-1}$.

Conclusions

In conclusion, G.A.N.N seems to be much more advanced than the classic A.N.N, much smarter, adaptable, and stable and maybe is the best choice for solving any or many kind of problems.
It combines a set of techniques and methods and with the help of its leading architecture it is capable of coping with problems that the classic A.N.N do not.
The classic A.N.N try to solve the problems with fixed mathematical models. Well, the main idea is not that maths are bad but sometimes we need something that is called luck or better, randomness. When the entropy of a system is too small and we have given borders to our A.N.N it is not logical to ask for complicated results, especially for real systems. That would be really foolish!
So G.A.N.N is based on a less organized pattern, the chaos theory. It is good to organize things but in our case we'd better leave the borders and the fixed models in the books. That's the theory behind G.A.N.N.
Furthermore, the way that our brain works, though its routing models can be easily described mathematically, is random. We still haven't found the way of how our synapses change values and as far as I can see this would never be modeled!
In other words we can not model the way that the hypothetical weights are multiplied with the values that our synapses transfer or even worse how the hell these weights are produced every second!

Bibliography

[1]. Artificial Neural Networks (Bose – Liang, Prentice Hall 1987)

[2]. Neural Fuzzy Systems (A Neuro-Fuzzy Synergism to Intelligent Systems – Lin CT, Lee CS: Prentice Hall, Upper Saddle River, NJ 1996)

[3]. Neuro-Fuzzy and Soft Computing (Jang J, Sun C, Mizutani E. NJ: Prentice Hall 1997)

[4]. Artificial Neural Networks (A Comprehensive Foundation. 2nd ed. Englewood Cliffs, NJ: Haykin, Prentice Hall 1999)

[5]. Wikipedia (Web based encyclopaedia) [http://www.wikipedia.org/]

GNU/GPL

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your Freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free Software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in

whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for non-commercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues),

conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS