GENETIC NETWORK GENERATOR

By Ioannis Katelouzos.

Rev : 1.02

Contents

1.Introduction

I.Description

Genetic Network Trainer is an attempt to investigate how effective genetic algorithms can be in training a neural network. A GNT includes both the network and the genetic algorithm into one object. The training is done at two levels. First is the internal level, where the weights and biases are selected genetically. On top of this there is also a structural level, where the number of layers, networks, connections and activation functions are defined. See next chapters for more details.

The user can train a GeneticNetwork programmatically, or use a GUI. Then once the training has given satisfactory results, the network can be saved and reloaded from another application, where it can be used into the application it is meant to. In order to keep the library as small as possible, the GNT is compiled in two projects. One containing the main library and one containing the GUI (see chapter on code). It is also worth saying that the interface to the outside world of the library is such, that permits to the user create his own GUI without any limitations.

II.Limitations

For the moment only one dimensional inputs are accepted, so imaging projects are not expected to have satisfactory results. Possibility to implement multidimensional inputs and convolutional mechanism is under consideration.

LSTMs and any kind of staging or recurrent properties are not available.

Currently, binaries for Windows are only available. MacOSX and Linux are very probable in the future.

You can find the project on GitHub: https://github.com/NeuralDip/GNT

and on the Developer's Site: www.diversemechanics.com

2.Background

I.Networks

The networks in the GNT are very simple and straightforward. As you can see from Figure 2-1, the layer model is the expected. A layer in GNT contains a matrix with the

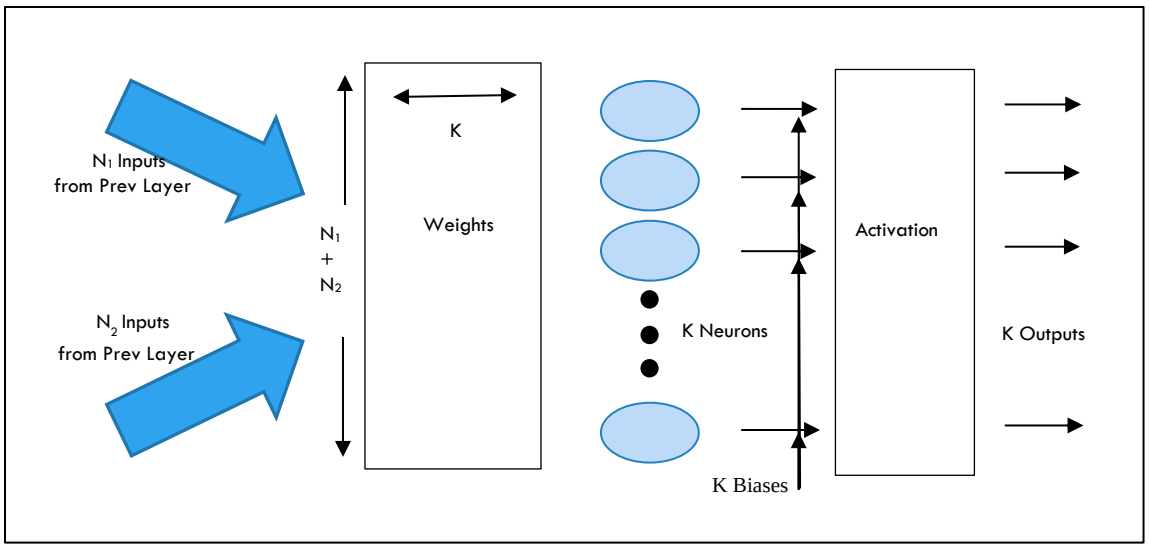

FIGURE 2-1 : LAYER MODEL IN GNT

weights, a column vector for the biases and an activation Function. The output is computed as in the general case : $Out_i = F(B_i + \sum_j^{N_1+N_2} W_{i,j} * In_i)$. Activation functions supported are: Linear, ReLU, SoftSign, Sigmoid, Tanh and SoftMax.

What is particular with the GNT design is the fact that layers are ready to connect with multiple other layers in input and in the output, contrary to what happens in a classic NN where layers are mainly packed one after the other. A typical GNT network can be seen in Figure 2-2. The fact that in a GNT there is no minimizing algorithm for the training procedure, assures that complex connections will not reduce the quality of the solution.

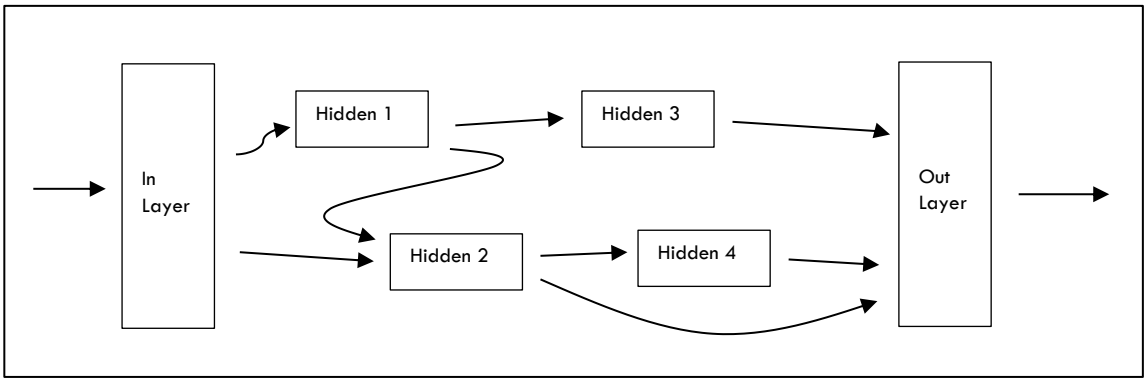

FIGURE 2-2 : A POSSIBLE GNT NETWORK

Instead, what affects the quality is the number of weights and biases which are directly related to the number of neurons and connections. This problem can be dealt with if we have enough time to use many islands and big populations.

II.Genetics

As already explained, there are two stages in the genetic selection. In order to understand how they function and how they are related, we will quickly go through the basic aspects of genetic algorithms that are used in GNT.

.Selection

Selection of a network over another is done by the score function. The score function will evaluate the network's ability to reproduce the expected result. Currently there are only two score functions to choose from, but there might be added more in the future. After the score is calculated, the nets are ordered and a new generation is created. New children can be generated by copying directly a parent, mutating a father, or cross-overing two parents (That is taking parts of two different parents to create a child).

.Mutating

Mutating for the internal populations is quite straightforward. A random value is added to all weights and biases as long as they are selected for mutation (see GUI for internal genetics for details).

Structural mutation is a bit more complicated and it is based on a system with a budget and mutation costs to describe the mutations that will take place (see GUI for structural genetics for detail).

.Cross-overing

Crossover for the internal populations consists in taking the weights matrix and biases vector from two parents and cutting them at random point. Then the chunks from both parents are stitched together to make a new child.

Structure crossover uses the adjacency matrix of the net and is called GraphX operator[1]. Is not much complicated, but is outside the scope of this document. (See footnote for reference).

.Islands

By using Islands (multi-populations in literature) we mean that we have many populations that evolve independently. In this way they are left to explore different traits for the networks (different local minima), and after some generations they are put together in order to merge their solutions. There are different controls for structure and internal islands (see chapter on the GUI).

[1] Stone S., Pillmore B., Cyre W. - 'Crossover and mutation in Genetic Algorithms Using Graph-Encoded Chromosomes'

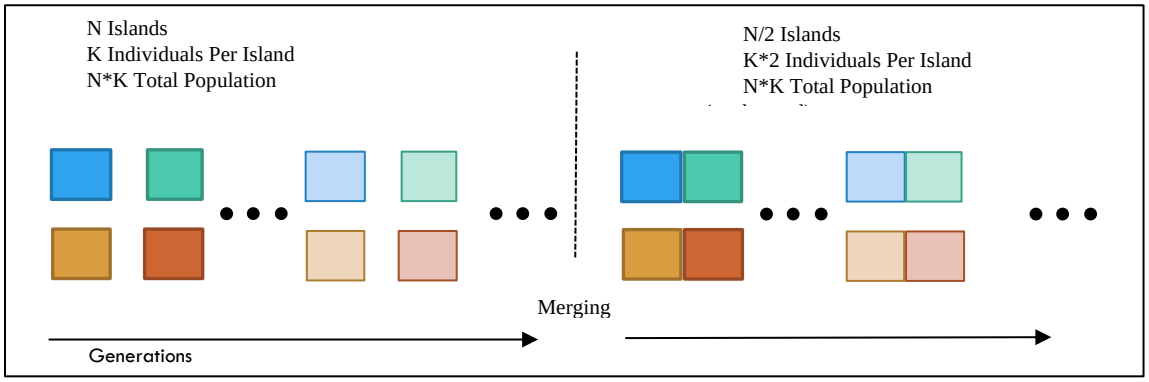

FIGURE 2-3 : MERGING OF ISLANDS

.Annealing

In GNT, Annealing is not used in its strict sense. It is used as an additional shuffling for the internal level. What it does is it adds extra noise both while mutating and cross-overing. It has to be used with care and only if we feel that the solution space is a very rugged one.

.The GNT hierarchy

In GNT, structure islands contain the net structures, that contain the internal islands that contain the actual nets. What the trainer has to do, is to train from the bottom up. Before

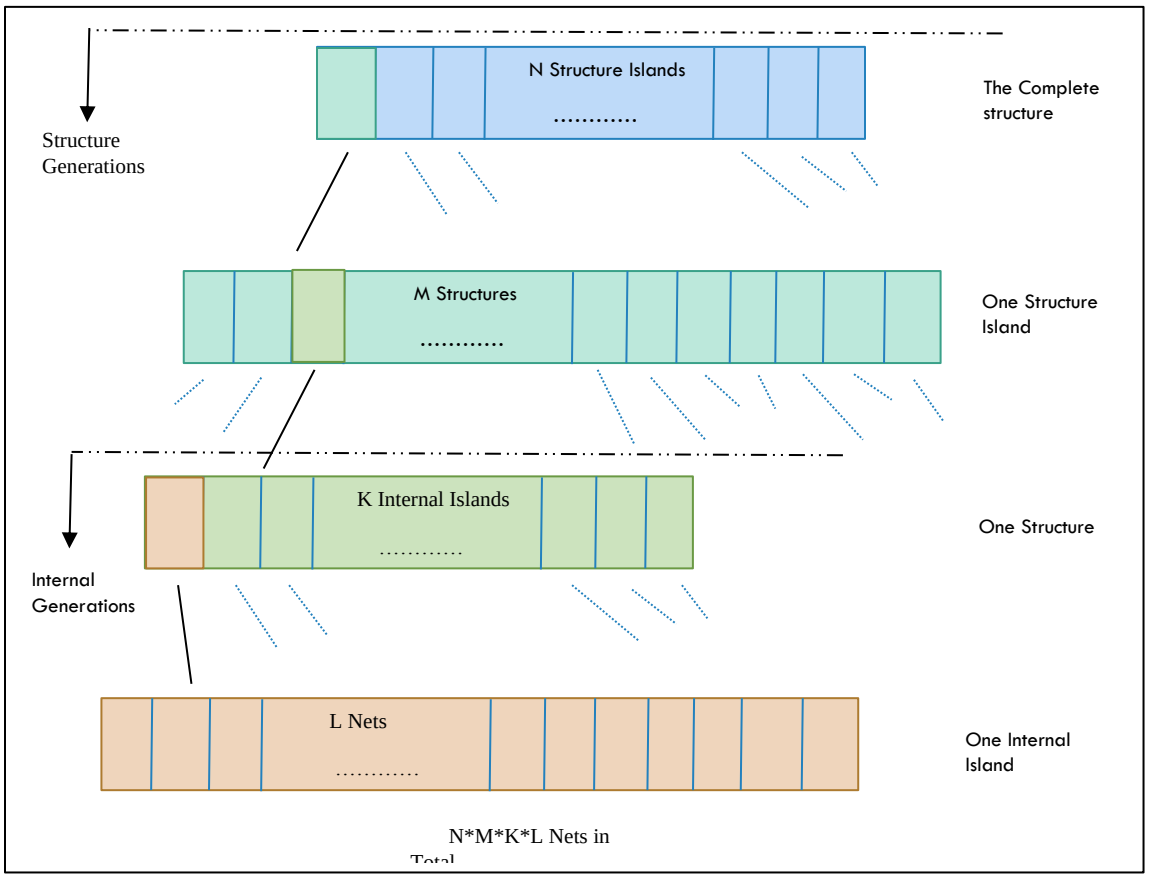

FIGURE 2-4 : THE INTERNAL STRUCTURE OF THE GNT

discarding a structure, training of its internal properties (weights, biases) has to be complete. This is why the internal part of the structure has to complete all the generations, before creating a new structure generation. This is a key detail, that helps the user

understand the performance of his simulation, but also some other aspects, like the different behaviors of structure and internal islands, but also the multithreading jog splitting.

Jobs for the multithreading case refer to one structure, so if we have N Structure islands and M structures per island, one structure generation will be split in N*M jobs. Simulation cannot be halted on any state of the process, but only when a structure generation is complete. This means that internal island merging is transparent to the user, as he will only see as many internal island as he has set for the initial state. This is also a reason why not letting the internal islands to merge to become one in an internal generation's time will not yield better results.

On the other hand, one may decide not to let the structure islands to merge into one (i.e. with StructIslands = 8, Steps = 2. See GUI chapter for details) because he might want to divide the simulation into many sessions and start merging when he thinks it is time.

3.The GUI

The GUI (GNTForm) is compiled as a separate project and is used to help the user run the simulation. In no way this GUI is required for the GNT to be trained and also the GNT contains function delegates that can access any custom interface. The GNTF is comprised by 6 tabs and we present their functionality hereafter.

I.Main Control

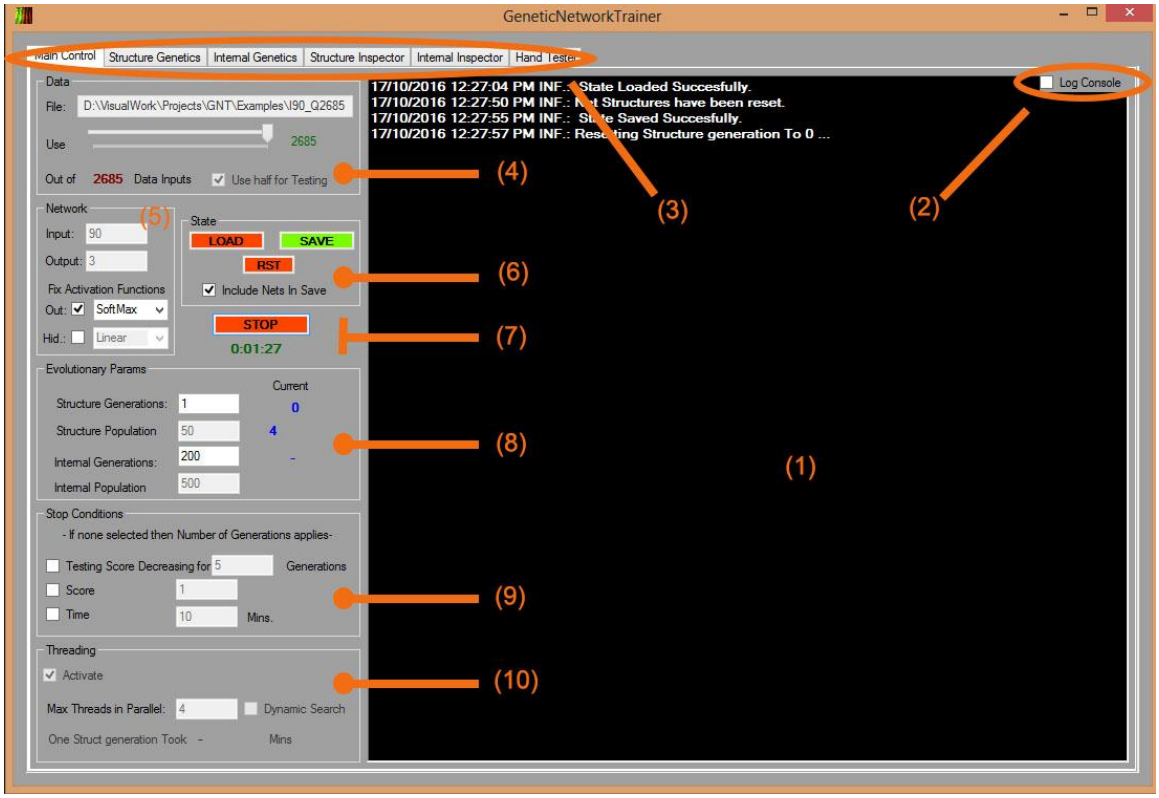

FIGURE 3-1 MAIN CONTROL TAB

1) Console. Three types of messages (infos, warnings and errors).
2) Check this box to output the console into a file named: Log.txt
3) The main Tabs
4) Input data are selected from here (see chapter on the code for details about the data structure). There is a file selector, a slider that tells the program to only use a proportion of the whole dataset and a checkbox enabling the TestScores. TestScores are calculated exactly as the normal scores, but they are not used for the selection process. They are an indicator of how well the net does on "new and unseen" inputs. It has the same use as the Train dataset for a normal NN training.
5) Here you chose some main parameters for your net, like inputs and outputs. They cannot be smaller than 1 and their sum must be equal to values contained in a line in the data file. You can also fix the activation function for the the output and the hidden layers.

6) Buttons to load, save and reset the state of the simulation. Make sure to follow the instructions in the console in case there are any. The checkbox allows you to save the whole structure with the nets. This can take some time if the structure is too big. Filename is: GenTrainingSave.state.
7) The Start-Stop button and an indicator of the time elapsed since the last time the start was pressed.
8) Evolutionary parameters refer to the structural and internal numbers of generations and populations. They describe half of the structure in Figure 2-4. Be aware that by population here we mean the total population and not the per island which is calculated automatically.
9) A simulation can be stopped if the maximum number of structure generations has been reached, if we stop it by hand, or if one of these three conditions is met. They all are self-explanatory. For the first condition to become accessible, one has to have checked the checkbox on bullet (4).
10) This is the multi-threading panel. Once you activate it you can set the maximum number of threads in parallel, which anyway cannot exceed the number of total structures. If the Dynamic search is activated, the best solution for your particular system will be sought automatically.

II.Structure Genetics

1) This is the panel where the rules by which the next generation is defined are set. Percentages of copied mutated and cross-overed offsprings will sum up to 1.
2) Here you will define the amount of mutation that each mutated child will have. Mutation Strength is like the available budget of mutation to spend. Beneath that, number of layers, number of connections number of neurons and type of activation function have a certain cost in order to be modified. Smaller is the cost and bigger the budget, higher the probability for that particular aspect to be mutated (if connections are mutated once, it means that a connection has been added or removed and not that the total number of connections has changed for some amount bigger than one. The same applies for the other variables. If you want the mutation to happen for more than once, then you have to reduce accordingly the cost).
3) These checkboxes randomize the weights and the biases of all nets every time a new structure generation is constructed. If left unchecked, the weights from a previous generation will persist in the new one (remember we are at the beginning of a structure generation).
4) Here you can control structure islands. There is a slider that defines the number

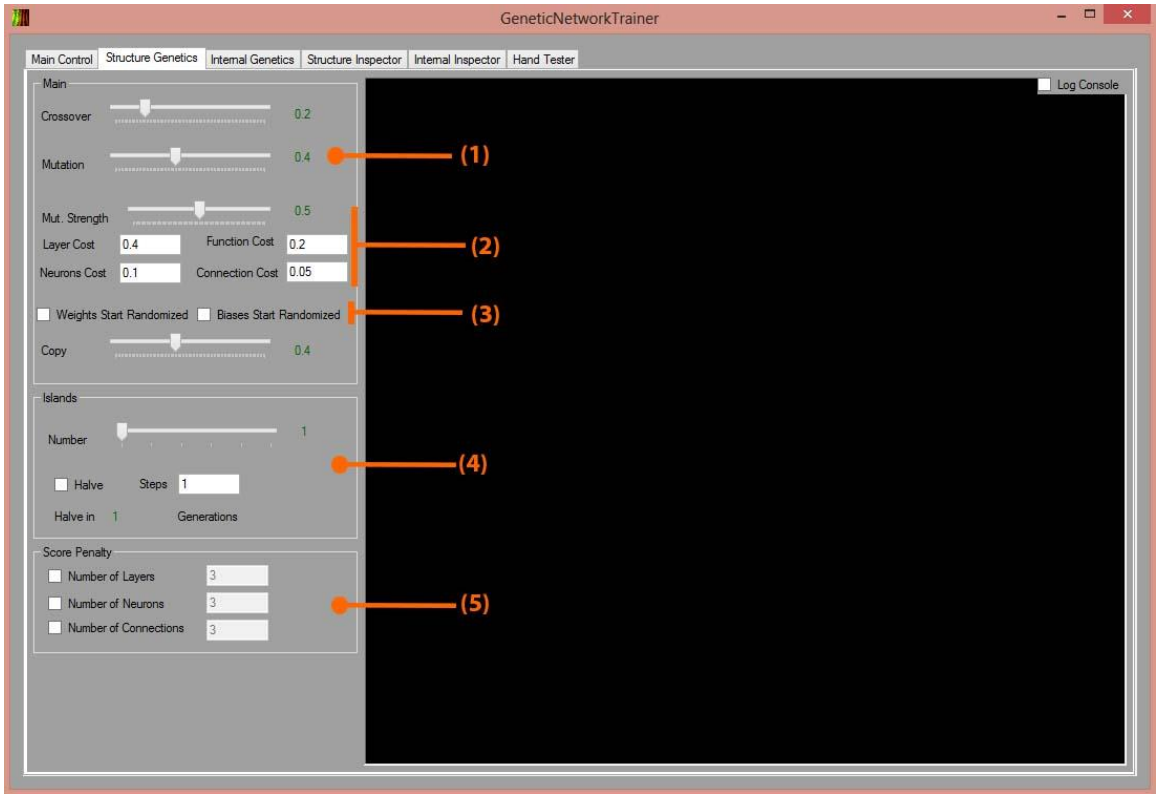

FIGURE 3-2 : STRUCTURE GENETICS TAB

of initial islands to use (power of 2 in order to merge in couples). Merging will take place only if the halve checkbox is set. During the total duration of the generations, there will be as many mergings as indicated in the Steps textbox

equidistantly placed (i.e. with islands=8, steps=2 and generation =30, at the end of the simulation we will have 2 islands while mergings will take place at generations 20 and 10). Current number of islands will be updated on the slider.

5) These checkboxes limit the dimension that a net can have. Once a net exceeds the amounts described, during mutation, adding a component will be less probable than removing it. Of course if they are not checked they have no effect.

III.Internal Genetics

In internal genetics tab the panels and controls are very similar to the structure counterpart.

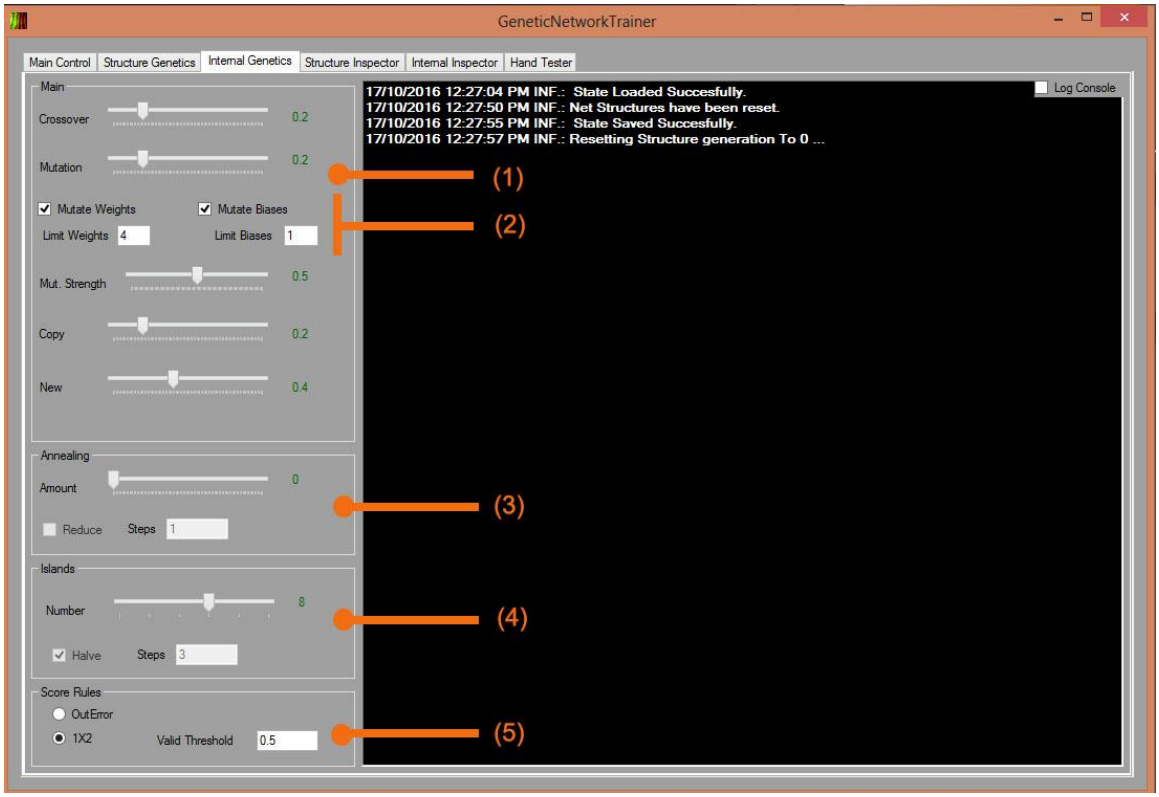

FIGURE 3-3 : INTERNAL GENETICS TAB

1) The Crossover, Mutation, Copy and New percentages of the next generation New stands for the networks that will be created with random weights and biases.
2) The two checkboxes confirm that you want the weights or the biases to take part in the mutation. Mutation strength is the maximum limit of the delta that the mutated weight or bias can have. The functioning of the limits is demonstrated in Figure 3-4. The purpose of it is to not let the values spread indefinitely and also to not let them gather on the limit.

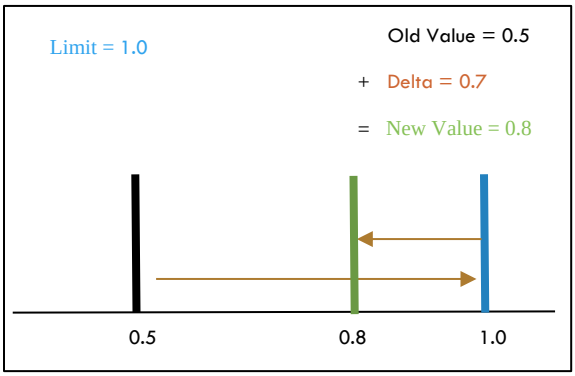

FIGURE 3-4 : WEIGHT & BIAS LIMITATION

3) Annealing is some additional noise added to the variables of the new child. Reduce and Steps have the same effect as in Islands, except from the fact that here annealing is not halved, but divided Steps times.

4) The islands here function exactly as in the structure genetics, except from the fact that the user will not see the islands reduce on the fly, as the simulation cannot be stopped in the middle of a structure generation.
5) Currently there are only two score functions implemented. The OutError function is the negative of the error between the actual and the expected output. The negative value is taken in order to give to the best performant net the bigger score. The 1X2 is a score thought for training with football match data. If the net's prediction is smaller than Valid Threshold then we did not bet for the game and the score remains unchanged. If the prediction is bigger than Valid Threshold then we played the bet and there are two possibilities. The expected outcome was 1 and we won the bet or the expected was zero and we lost it. If we lose, then the score is reduced by 1 and if we win the score is increased by the value in the input. In order for the 1X2 score to become accessible, we have to set the outputs of the net to 3 in the Main Control tab. It is worth noting at this point that if some day we define a score that is independent of the output, then it will be easily possible to use GNTs for unsupervised learning too.

IV.Structure Inspector

The two inspectors are where we can make the quality analysis of the GNT. They are very similar, and intuitively there is one dedicated for the structural level and one for the internal.

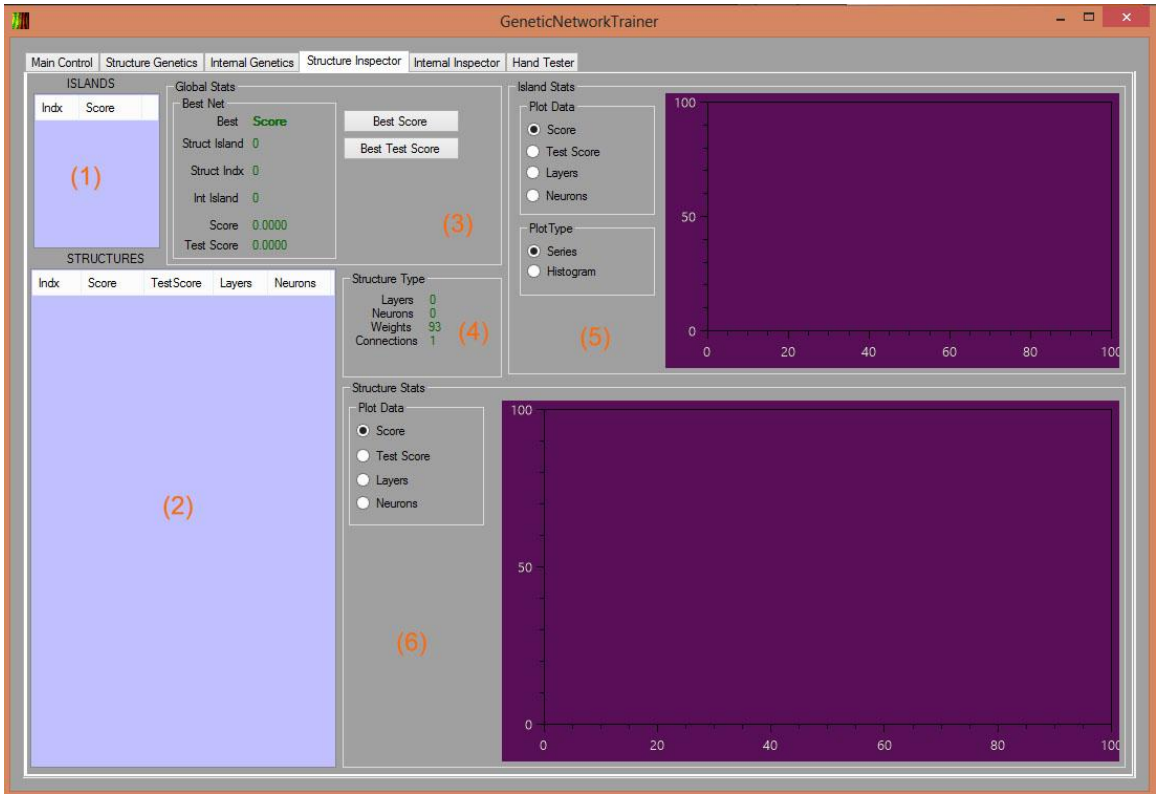

FIGURE 3-5 : STRUCTURE INSPECTOR TAB

1) This is the list of all the structure islands. Everything we see on the two inspector tabs depends on what is the structure islands selected. We can see the island index and the best score in the current island.
2) This is the list with all the structure contained in the island that has been selected in the previous list. Each structure has an index, a best score, a best test score (zero if not active), the number of layers and neurons every structure has.
3) Here you can find the scores and indications on how to find the best net currently in the whole GNT. You have also two buttons available to switch between score and Test Score seeking.
4) Stats about the selected structure.
5) Once an island is selected you will be able to see plots of the progress of the score, the test score, the max layers and the max neurons for the current island.

If the histogram is selected, then the info concerns the values from all the structures contained in the island for the current generation.

6) These plots are about the structure selected and they don't have a histogram available.

V.Internal Inspector

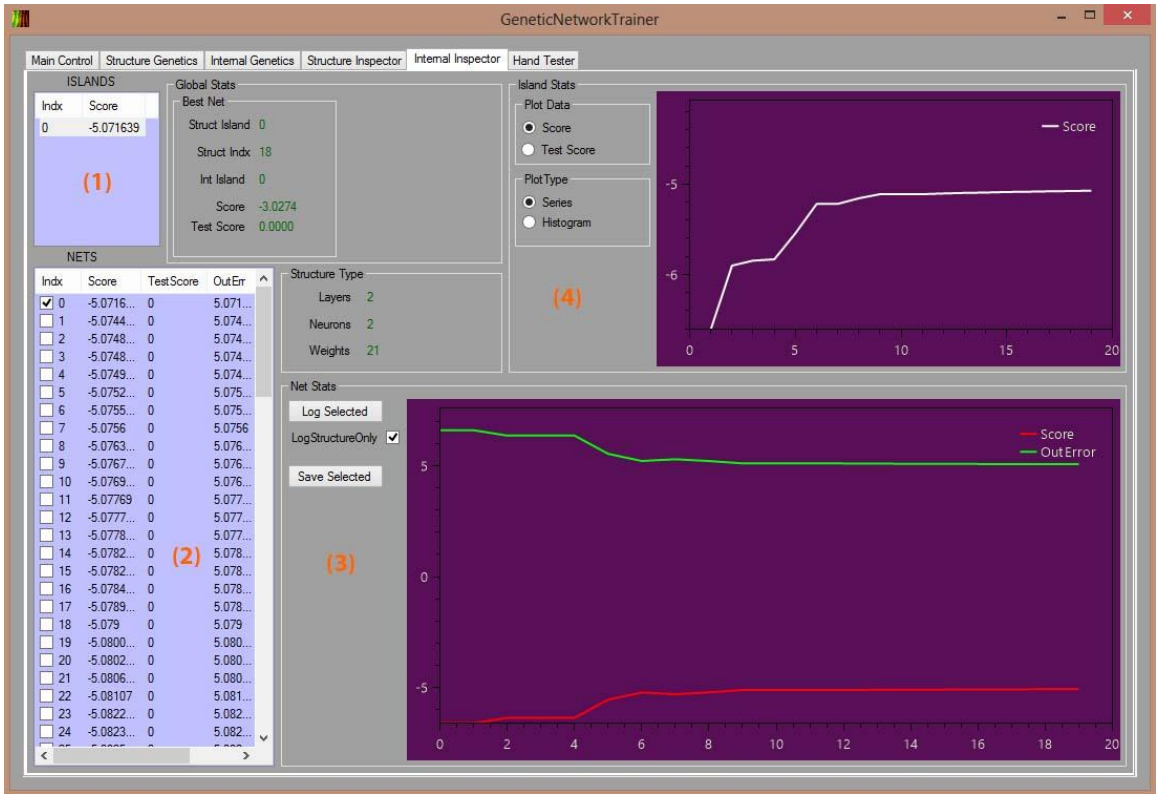

FIGURE 3-6 : INTERNAL INSPECTOR TAB

1) As before this is the list for the internal islands.
2) This is the list of the actual nets. You can multi-select them in order to compare them in the plot right next.
3) Except from plotting the scores of the nets you can also log their characteristics into the console (adjacency matrix, layers, activation function etc.). If LogStructureOnly is unchecked, then all weights and biases will be logged too. This is also the place to export the net which will have a long name describing itself and the .gnet extension.
4) This is the plot for the internal islands. Remember that everything we see in this tab has fixed structure, so structure stats are not available.

VI.Hand Tester

Here you will be able to quickly test the net selected in the previous tab.

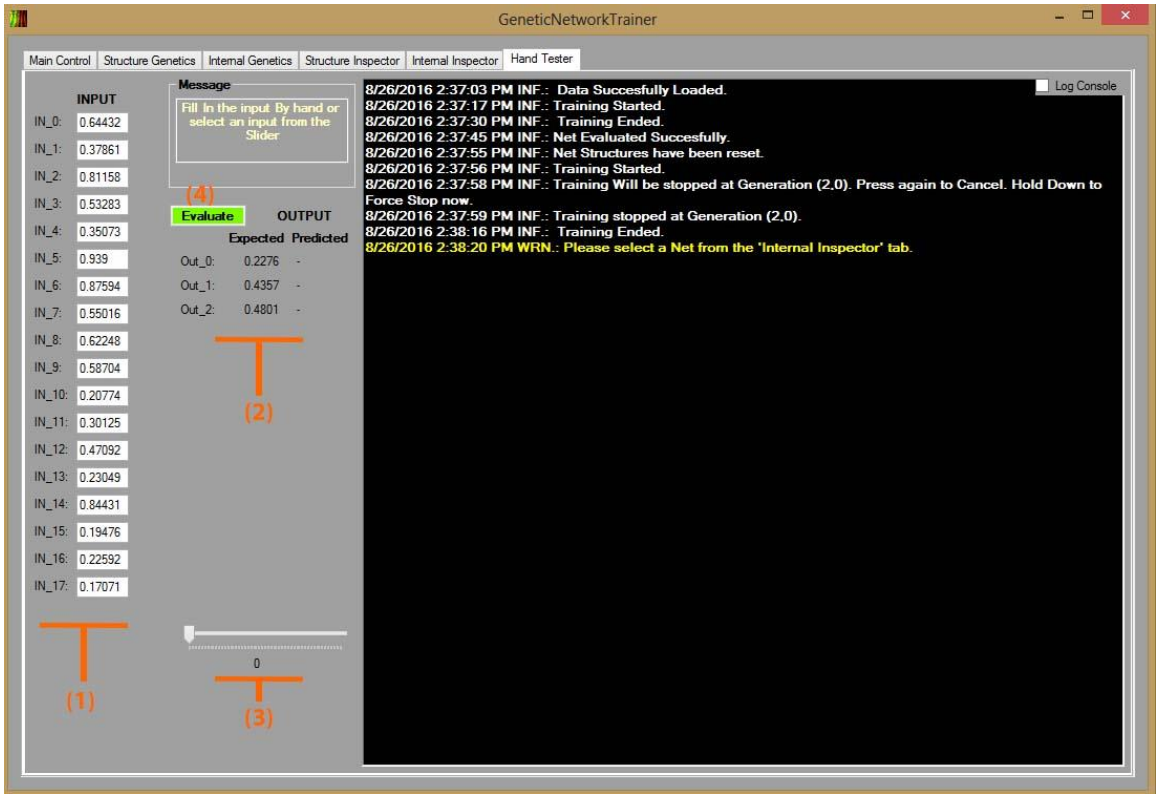

FIGURE 3-7 : HAND TESTER TAB

1) All the inputs that will be used to test the network. They are editable.
2) … and the outputs.
3) All the data from the input file you provided in the Main control tab. Select one from here to quickly fill the input.
4) A message box and the evaluation button.

4.The Code

GNT was done with C# and Visual Studio. This is why solution and project files related to VS are provided.

I.Folder structure

On top there are 5 folders.

1) Bins. Containing the binaries. Be aware that in order for the GUI .exe to execute, no other .dll is needed as everything in merged in the .exe. The projects generate their compiled assemblies directly here.
2) Documents. This Documentation.
3) Examples. Some simple examples to quickly test the GNT (See next paragraph for details).
4) GeneticNetworkTrainer. The project with the core GNT code.
5) GeneticNetworkTrainerForm. The GUI project.

II.Input Data

Currently only comma separated .csv files are accepted. One line has to contain all inputs and outputs of one example, with the output at the end. That is the inputs plus the outputs of the network will be as many are the values in one line, and the examples will be as many are the lines. In the Examples folder you will find some files that you can use.

- GNTAverage_500_10.csv. 500 examples of 10 inputs and 1 output which is the average of the inputs.
- GNTInverse_500_10.csv. 500 examples of 10 inputs and 1 output which is the inverse of the constant inputs.
- GNTInclination_500_4.csv. 500 examples of 4 inputs and 2 outputs. The inputs are points on a line and the outputs are the inclination of the line and the bias.
- GNTNeuralodds_100_30.csv. 100 examples of football games odds. 30*3 inputs which are the progress of the home, tie and away odds and 3 outputs which represent the actual result. Be aware that 100 examples are too few and even if the result may seem good, you are advised not to use it for true betting.

III.GNT Project

.GenLayer

This is a class defining the layer of our network. It is an internal class and so it is invisible to the user.

.GenNetwork

This is the class for one network. Implements the IComparable interface and provides many methods for manipulation which are described in Table 1.

Name	Description
`GenNetwork(int InDimention, int InNeurons, int OutNeurons, params int[] HiddenDimentions)`	The constructor. Give the input dimension, the number of neurons of the first layer, the output dimension, and a list of the neurons for every hidden layer if applicable.

`GenNetwork CloneMe(bool Reset, bool RandomizeWeights, bool RandomizeBiases, Random Rnd)`	Deep Copy of itself. Resets the weights with zeros or randomized values. Provide a fresh Random object. Returns a GenNetwork.
`float GetScore()`	Returns the score as a float.
`float GetTestScore()`	Returns the test score as a float.
`float GetOutError()`	Returns the out error as a float.
`float GetTestOutError()`	Returns the test out error as a float.
`int GetLayersNumber()`	Returns the number of Layers as an int.
`int GetWeightsNumber()`	Returns the number of weights as an int.
`int GetNeuronsNumber()`	Returns the number of neurons as an int.
`int GetConnectionsNumber()`	Returns the number of connections as an int.
`float[] GetNetOutput()`	Returns an array of floats with the current output. Make sure you evaluate the net first.
`string LogMeDesciption()`	Returns a string containing the adjacency matrix and the stats of the net.
`String LogMeParams()`	Returns a string containing weights an biases for every layer in the net.
`float[] EvaluateNet(float[] GlobalInputs)`	Evaluates the net given the input and returns the output. Make sure the dimension of the inputs equals that of the net.
`void ResetScores()`	Resets all scores to zero.
`bool CalculateScores(List<float[]> InData, List<float[]> Labels, int DataToUse, bool Test, GenTrainer.ScoreRules ScoreRule, float WinThresh)`	First evaluates the net with the input given and then calculates scores. Return false if input is inconsistent with the net. Parameters are : Input data, expected data, number of examples to use, whether test score is activated, score rule and threshold for the 1X2 score.
`GenNetwork MutateInternal(bool MutateWeights, bool MutateBiases, float MutationStength, float Annealing, Random Rnd)`	Returns a clone of this, internally mutated. Provide a fresh Random object.
`GenNetwork CrossoverInternal(GenNetwork OtherParent, bool MutateWeights, bool MutateBiases, float Annealing, Random Rnd)`	Returns a new GenNetwork starting from this and OtherParent internally cross-overed. Provide a fresh Random object.
`GenNetwork MutateStruct(float MutationStength, float[] Costs, bool[] PenaltyBools, int[] PenaltyValues, Random Rnd)`	Returns a new GenNetwork starting from this, structurally mutated. Provide the mutation strength, an array of 4 floats for the costs(layer, function, neuron, connection), an array of 3 bools for active penalties(layers, neurons, connections), an array of 3 ints with the respective values and a fresh random object.
`GenNetwork CrossoverStruct(GenNetwork OtherParent, Random Rnd)`	Returns a new genNetwork starting from this and the OtherParent Structurally cross-overed. Provide a fresh Random object.
`int CompareTo(GenNetwork Other)`	Used for the implementation of IComparable
`void ExportNet(string NetFileName)`	Save this net to disk.
`static GenNetwork ImportNet(string NetFileName)`	Return a new GenNetwork by reading the given file.

TABLE 1 : GENNETWORK METHODS

.GenTrainer

GenTrainer is the global GNT class that holds the complete structure and provides all the methods that are needed to operate the GNT. They are described In Table 2.

GenTrainer has one main file GenTrainer.cs and one dedicated for the Multi-Threaded code: GenTrainer.Multithreaded.cs.

Name	Description
`GenTrainer(Action<string, int> iLoggingFunction, Action<string, string, object> iParentFormControlSet, Func<string, string, object> iParentFormControlGet)`	Constructor. Provide three delegates belonging to the form for logging, setting and getting controls. Leave the to null if you don't want to use them.
`void ResetStructures(bool Complete, bool StatsOnly)`	Resets net and statistics structures.
`void TrainNetNotThreaded(object Dummy)`	Start training Single-Threaded. Parameter is dummy for the thread convention. That is launch this as a separate thread. Don't use the GUI thread.
`void LaunchNextStructGeneration(object JustPressedButton)`	Start Training Multi-Threaded. Provide a bool set to true.
`void LoadState()`	Loads the file named GenTrainingSave.state if it exists and fills the internal state of the GNT.
`void ForceSaveState()`	Save the state to a file named GenTrainingSave.state.
`void SaveState()`	Save the state once current structure generation ends.
`void ForceStop()`	Stop simulation right now.
`void Stop(bool Undo)`	Stop simulation once current structure generation ends.

TABLE 2 : GENTRAINER METHODS

There are also some variables that are publicly accessible shown in Table 3.

Name	Description
`Action<string, int> ParentFormLogging`	Delegate for logging into the GUI. Can be set with the constructor, or manually. Provides a string for the message and a severity integer. 0=info, 1=warning, 2=error.
`Action<string, string, object> ParentFormControlSet`	Delegate to set parameters to controls. Can be set with the constructor, or manually. Provides a string for the control's name, a string for the parameter's name to set and an object for the parameter's value.
`Func<string, string, object> ParentFormControlGet`	Delegate for getting values from controls. Can be set with the constructor, or manually. Provides a string for the control's name, a string for the parameter's name to get. GUI must return the value requested as an object.
`event SomethingHappenedDelegate CallTheForm`	Register to this event the method in the form that will be called when GNT wants to indicate an event. Type of events available is TrainingState
`StateClass MyState`	All the state of the GNT. StateClass is serializable and is the object that that will be saved in the state file.
`List<List<List<GenNetwork[]>>>SettledNetsStructure`	All nets in the structure. If you manually edit this object, it will be overwritten with the proper values once the current structure generation ends.

`StatsStructureClass SettledStatsStructure`	All statistics in the structure. If you manually edit this object, it will be overwritten with the proper values once the current structure generation ends. Make sure you study StatsStructureClass before you access it.
`int HistogramsBins`	Let the GenTrainer know how many bins you want for the histograms in case you use them.
`bool StateFileExists`	Let the trainer know if the file GenTrainingSave.state exists.

TABLE 3 : PUBLIC VARIABLES IN GENTRAINER

IV.GNTF Project

.File Splitting

Entry point for the project is Program.cs as usual. The form is called MainForm and its top files are MainForm.cs and MainForm.Designer.cs. Furthermore, every Tab of the GUI has a separate file for its event handlers and methods.

.Dependences

There are a couple of dependencies that the project will ask to download the first time you open the solution. Those are Oxyplot for all the plots and Costura that merges the assemblies together. In our case GNT,GNTForm and Oxyplot. You can check the details in the packages.config file.