Make AI Universal and Accessible

Neural Network's User's Guide

Version: 202111101200.

CONTENTS

1 Introduction

The **Anaimo Neural Network** is a proprietary software library which allows to create, train, and use neural networks for whatever application developed in C/C++, C#, VB.NET and more. Its main characteristics are:

- **Optimized**:
  - For computational high speed.
  - To dynamically consume the least memory possible.
- **Cloud or on-premises** (*Edge AI*).
- **Auto adaptative**: it can self-adapt its topology for faster and better performance, like the human brain.
- **Flexible topology**: connect neurons as you need.

This document constitutes the User's Guide of the Anaimo AI neural network version 202111101200.

The libraries of the Anaimo AI neural network are available in the folder \Anaimo_AI_SDK\202111101200\Libraries, for platforms:

- **Windows**: file AnaimoAI.dll
- **Linux (and potentially MacOS)**: file libAnaimoAI_nn.a
- **Windows (Win32) COM compatible**: file AnaimoAI.dll

For more information or support, please visit or contact us at: https://Anaimo.com

1.1 License

Anaimo AI neural network is a proprietary software which can only run:

- Free license: for neural networks with less than 1000 neurons.
- Paid license: for neural networks with 1000 or more neurons.

If you want to obtain a paid license from Anaimo, you need to provide Anaimo with all the hardware id codes of the computer which will run the neural network. To obtain them, please run the library on the computer which you need to license and use the HardwareId public function to obtain and provide with all the hardware identificators.

1.2 How functions are ordered in this guide

The functions in this manual are ordered alphabetically, for faster reference, and not in the order in which the functions should be used.

1.3 Inputs, ouputs and topology

Inputs and ouputs of the neural network are regular neurons. You indicate how many inputs your network has in the NetCreate function. After that, to indicate how many outputs your network has, you use the function NetAddOutput. Therefore, there does not exist any function to add inputs.

Therefore, the topology, this is the number of neurons, inputs, outputs and their connections; is completely free and it is responsibility of the programmer.

1.4 Control of cycles

Depending on the network topology, neurons can be interconnected so that they stablish closed loops, for example when neuron A outputs to neuron B which outputs back to neuron A. This sometimes can happen after multiple levels and therefore cycles will not be easily seen.

If your topology connects neurons in loops, the neural network could enter an infinite loop and hang or even drain the resources of the computer. The functions which might experience this problem will require the parameter CyclesControl to be with value 1 to avoid this problem.

1.5 Numbering of neurons

Neurons are numbered starting with 0. In general, this is applied to all other items (inputs, outputs, weights, etc.).

1.6 The set and the auto adaptative mode

One of the disadvantages of the learning process of an artificial neural network versus a person, is that the first needs thousands of records and iterations to learn what the second can learn with just a few records and with almost no iteration.

To reduce that difference, Anaimo AI developed:

- **Set**: all the records that are used to learn can be memorized inside the neural network.
- **Self-training from the set**: the neural network can use the set for self-training.
- **Auto adaptation**: once the self-training has finished, the neural network self-adapts (auto adaptative mode) its topology to increase speed and save computational resources.

Only when using the set functionality, the Anaimo AI neural network will enter into the auto adaptative mode. The related Set functions are also explained in this guide.

1.7 Modes

There are different working modes for the neural network. The mode affects mainly to training, but it could also affect other operations. There are currently these modes available:

1. **Standard back propagation**:

    - Calculates neurons' outputs.
    - Calculates deltas.
    - Adjusts biases and weights.

2. **Standard back propagation optimized**: same as mode standard back propagation but optimized for parallel hardware and high volume of neural networks and data. Works only with full connected layers. When working with multiple threads, it is faster than standard back propagation but consumes typically a 70% less processing power.

    This mode will automatically create an internal memory structure (cache). You can also create this cache manually with the method NetLayersAnalyze. This cache will take some time and memory to create. Once created, it will be maintained during operation. This cache will be automatically deleted and, after, recreated, if the following methods are used: NeurAddInput, NeurAddInputsConsecutive, NeurAddOutput, NetCreate and NetDestroy.

In this mode, the following methods will return always the value 1: NeuInputWeightUpdatedGet and NetValueUpdatedGet. This is because these internal counters will not be updated for the efficiency of the mode.

This mode does not currently support neither Dropout nor Momentum.

3. **Dynamic propagation (beta)**: for all neurons, adjusts biases and weights via calculating neurons' outputs and deltas when needed. This mode supports any network topology. This mode is more than 50% faster than standard back propagation, although is still being validated in different use cases.

2 Public functions

The following are the published available functions.

2.1 Common functions

2.1.1 HardwareId

2.1.1.1 Purpose

Provides 16 hardware identificators, in 4 rows by 4 columns, which uniquely identify the computer running the neural network. This data must be sent to Anaimo to obtain a licensed version of the neural network library.

2.1.1.2 Declarations

Standard C:

```
extern int HardwareId(unsigned int pIntRow, unsigned int pIntCol);
```

MS Visual Studio:

```
extern int _cdecl HardwareId(unsigned int pIntRow, unsigned int pIntCol);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall HardwareId(unsigned int pIntRow, unsigned
int pIntCol);
```

2.1.1.3 Parameters

- Row [0,3]
- Column [0,3]

2.1.1.4 Returns

An integer.

2.1.1.5 Usage

The following C example will provide the hardware id of the machine running the neural network:

```
char lStrTmp[1024] = "";

char lStrTmp2[1024] = "";

for (int i = 0; i < 4; i++)
```

```
        for (int j = 0; j < 4; j++)
        {
                snprintf(lStrTmp, sizeof(lStrTmp), "%X", HardwareId(i, j));
                strcat(lStrTmp2, lStrTmp);
                strcat(lStrTmp2, ":");
        }
```

2.2 Network functions

A network is a group of neurons.

2.2.1 NetActivationFunctionSet

2.2.1.1 Purpose

Selects the activation function to be used.

2.2.1.2 Declarations

Standard C:

```
extern void NetActivationFunctionSet(int pInt);
```

MS Visual Studio:

```
extern void _cdecl NetActivationFunctionSet(int pInt);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetActivationFunctionSet(int pInt);
```

2.2.1.3 Parameters

- Activation function, with possible values of:
    - 0: for Sigmoid.
    - 1: for ReLU.
    - 2: for fast Sigmoid.

2.2.1.4 Returns

Nothing.

2.2.1.5 Usage

The following C example will set ReLU as the activation function.

```
NetActivationFunctionSet(1);
```

2.2.2 NetConnect

2.2.2.1 Purpose

Connects two neurons.

2.2.2.2 Declarations

Standard C:

```
extern bool NetConnect(int pLngSrc, int pLngDst);
```

MS Visual Studio:

```
extern bool _cdecl NetConnect(int pLngSrc, int pLngDst);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  bool _stdcall NetConnect(int pLngSrc, int pLngDst);
```

2.2.2.3 Parameters

- Number of the source neuron.
- Number of the destination neuron.

2.2.2.4 Returns

A boolean with true if connection was successful.

2.2.2.5 Usage

The following C example connects the 10th neuron to be the input of the 20th neuron and returns in a boolean variable, true or false if there was an error.

```
bool lBolTmp = NetConnect(9, 19);
```

2.2.3 NetConnectConsecutive

2.2.3.1 Purpose

Connects a consecutive list of neurons to one another neuron. Its purpose is to speed up the connection of a high number of neurons to a destination neuron.

NetConnectConsecutive will behave as NetConnect when the first 2 parameters are the same number (pLngSrc1 equals pLngSrc2).

2.2.3.2 Declarations

Standard C:

```
extern bool NetConnectConsecutive(int pLngSrc1, int pLngSrc2, int pLngDst);
```

MS Visual Studio:

```
extern bool _cdecl NetConnectConsecutive(int pLngSrc1, int pLngSrc2, int
pLngDst);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  bool _stdcall NetConnectConsecutive(int pLngSrc1, int
pLngSrc2, int pLngDst);
```

2.2.3.3 Parameters

- Number of the initial source neuron.
- Number of the final source neuron. Note that all neurons from initial to final will be connected to the destination neuron.
- Number of the destination neuron.

2.2.3.4 Returns

A boolean with true if connection was successful.

2.2.3.5 Usage

The following C example connects the 10th neuron to be the input of the 20th neuron and returns in a boolean variable, true or false if there was an error.

```
bool lBolTmp = NetConnectConsecutive(9, 18, 19);
```

2.2.4 NetCreate

2.2.4.1 Purpose

Creates the basement of the neural network by dynamically creating part of the memory structures.

2.2.4.2 Declarations

Standard C:

```
extern bool NetCreate(int pIntMaxNeurons, int pIntMaxInputs, int
pIntMaxOutputs);
```

MS Visual Studio:

```
extern bool _cdecl NetCreate(int pIntMaxNeurons, int pIntMaxInputs, int
pIntMaxOutputs);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  bool _stdcall NetCreate(int pIntMaxNeurons, int
pIntMaxInputs, int pIntMaxOutputs);
```

2.2.4.3 Parameters

- Maximum number of neurons.
- Maximum number of inputs.
- Maximum number of outputs.

2.2.4.4 Returns

An integer, indicating:

- 0: **success**.
- 1: **success**, but license will **expire** in less than 30 days.
- 2: **not licensed**, as more or equal than 1000 neurons were requested to be created and neural network library is not licensed for the current hardware running it. In this case, you need to send to Anaimo the 16 integers obtained with the HardwareId function to qualify for a licensed version.
- 3: **out of memory**, you are trying to create more neurons or connections than the memory of your device supports.
- 4: **unknown error**.

2.2.4.5 Usage

The following C example will create in memory the neural network.

```
#define NetCreate_Success 0

#define NetCreate_LicenseExpiresInLessThan30Days 1

#define NetCreate_NotLicensed 2

#define NetCreate_OutOfMemory 3

#define NetCreate_UnknownError 4

int lIntTmp = NetCreate(pIntFinalNumberOfTotalNeurons, pIntInputsNumber,
pIntOutputsNumber);
```

2.2.5 NetDestroy

2.2.5.1 Purpose

Destroys the neural network from memory. Also destroys the set.

2.2.5.2 Declarations

Standard C:

```
extern void NetDestroy();
```

MS Visual Studio:

```
extern void _cdecl NetDestroy();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetDestroy();
```

2.2.5.3 Parameters

None.

2.2.5.4 Returns

Nothing.

2.2.5.5 Usage

The following C example destroys from memory the current neural network.

```
NetDestroy();
```

2.2.6 NetDropOutGet

2.2.6.1 Purpose

Returns a float with the current dropout rate in percentage, indicated with a number in the range [0, 1]. It is set to 0 by default.

If different than 0, will make that the dropout rate percentage of neurons will not be considered on each training cycle. These ignored neurons will be randomly selected.

2.2.6.2 Declarations

Standard C:

```
extern float NetDropOutGet();
```

MS Visual Studio:

```
extern float _cdecl NetDropOutGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetDropOutGet();
```

2.2.6.3 Parameters

None.

2.2.6.4 Returns

A float with the current dropout rate as a value in the range of [0,1]. It is set to 0 by default.

2.2.6.5 Usage

The following C example gets the dropout rate and puts it into a variable.

```
lSngDropOut = NetDropOutGet();
```

2.2.7 NetDropOutSet

2.2.7.1 Purpose

Sets the dropout rate in percentage, indicated with a number in the range [0, 1]. You can set it to 0 for no dropout. It is set to 0 by default.

If different than 0, will make that the dropout rate percentage of neurons will not be considered on each training cycle. These ignored neurons will be randomly selected.

2.2.7.2 Declarations

Standard C:

```
extern void NetDropOutSet(float pSng);
```

MS Visual Studio:

```
extern void _cdecl NetDropOutSet(float pSng);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetDropOutSet(float pSng);
```

2.2.7.3 Parameters

- The dropout rate [0,1].

2.2.7.4 Returns

Nothing.

2.2.7.5 Usage

The following C example sets the dropout rate at 10%.

```
NetDropOutSet(0.1);
```

2.2.8 NetErrorGet

2.2.8.1 Purpose

Returns a float which is the sum, in absolute values, of the errors of all neurons.

2.2.8.2 Declarations

Standard C:

```
extern float NetErrorGet(int pIntCyclesControl);
```

MS Visual Studio:

```
extern float _cdecl NetErrorGet(int pIntCyclesControl);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetErrorGet(int pIntCyclesControl);
```

2.2.8.3 Parameters

- Control of cycles (for more information please view the introduction):
  - 0: no cycles will be checked.
  - 1: cycles will be checked and avoided.

2.2.8.4 Returns

A float.

2.2.8.5 Usage

The following C code puts into a float variable the network total error:

```
float lSngError = NetErrorGet(0);
```

2.2.9 NetInitialize

2.2.9.1 Purpose

Initializes the neural network, by initializing in memory:

- Bias: puts 0.
- Values: puts 0.
- Neurons' weights: a random number in the range of [0,1] or the parameter if it is different than 0.

2.2.9.2 Declarations

Standard C:

```
extern void NetInitialize(float pSngDefaultVal);
```

MS Visual Studio:

```
extern void _cdecl NetInitialize(float pSngDefaultVal);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetInitialize(float pSngDefaultVal);
```

2.2.9.3 Parameters

- Default value for the weights of the neurons. Put 0 to set a random value in the range of [0,1].

2.2.9.4 Returns

Nothing.

2.2.9.5 Usage

The following C code initializes the neural network:

```
#define DEFAULT_VALUE_TO_INITIALIZE_WEIGHTS 0 //0 means random (recommended)

NetInitialize(DEFAULT_VALUE_TO_INITIALIZE_WEIGHTS);
```

2.2.10 NetInputSet

2.2.10.1 Purpose

Sets the value of an input.

2.2.10.2 Declarations

Standard C:

```
extern void NetInputSet(int pLngInput, float pSng);
```

MS Visual Studio:

```
extern void _cdecl NetInputSet(int pLngInput, float pSng);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetInputSet(int pLngInput, float pSng);
```

2.2.10.3 Parameters

- The number of the input.

2.2.10.4 Returns

Nothing.

2.2.10.5 Usage

The following C example sets the value of the 10th input to 0.1.

```
NetInputSet(9, 0.1);
```

2.2.11 NetInputsMaxNumberGet

2.2.11.1 Purpose

Returns the number of inputs that the neural network has.

2.2.11.2 Declarations

Standard C:

```
extern int NetInputsMaxNumberGet();
```

MS Visual Studio:

```
extern int _cdecl NetInputsMaxNumberGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetInputsMaxNumberGet();
```

2.2.11.3 Parameters

None.

2.2.11.4 Returns

Nothing.

2.2.11.5 Usage

The following C example get the number of inputs of the networks and puts it into a variable:

```
int lIntTmp = NetInputsMaxNumberGet();
```

2.2.12 NetLayersAnalyze

2.2.12.1 Purpose

Analyzes the current neural network and automatically builds the internal memory structure for the mode Standard Backpropagation Optimized. This function will be automatically called internally when learning in the cited mode. For more information, please read the information about the different working modes.

2.2.12.2 Declarations

Standard C:

```
extern bool NetLayersAnalyze();
```

MS Visual Studio:

```
extern bool _cdecl NetLayersAnalyze();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  bool _stdcall NetLayersAnalyze();
```

2.2.12.3 Parameters

None.

2.2.12.4 Returns

An boolean, indicating:

- true: success.
- false: failure, possibly out of memory.

2.2.12.5 Usage

The following C example analyzes the neural network and creates the internal memory structure to use the working mode Standard Backpropagation Optimized.:

```
NetLayersAnalyze(0);
```

2.2.13 NetLearn

2.2.13.1 Purpose

Makes the neural network to learn with the current inputs and outputs.

2.2.13.2 Declarations

Standard C:

```
extern int NetLearn(int pIntCyclesControl);
```

MS Visual Studio:

```
extern int _cdecl NetLearn(int pIntCyclesControl);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetLearn(int pIntCyclesControl);
```

2.2.13.3 Parameters

- Control of cycles (for more information please view the introduction):
  - 0: no cycles will be checked.
  - 1: cycles will be checked and avoided.

2.2.13.4 Returns

An integer, indicating:

- 0: success.

- 1: learning process generated NaNs (Not A Numbers), although it continued. This could mean that you should use Sigmoid as the activation function.
- 2: error managing threads.

2.2.13.5 Usage

The following C example makes the neural network to learn, without considering cycles, that the current inputs generate the current outputs.

```
#define NetLearn_Success 0

#define NetLearn_NAN 1

#define NetLearn_ThreadsError 2

NetLearn(0);
```

2.2.14 NetLearningRateGet

2.2.14.1 Purpose

Returns a float with the current learning rate.

2.2.14.2 Declarations

Standard C:

```
extern float NetLearningRateGet();
```

MS Visual Studio:

```
extern float _cdecl NetLearningRateGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetLearningRateGet();
```

2.2.14.3 Parameters

None.

2.2.14.4 Returns

A float with the current learning rate.

2.2.14.5 Usage

The following C example gets the current learning rate and introduces it into a variable:

```
lSngTmp = NetLearningRateGet();
```

2.2.15 NetLearningRateSet

2.2.15.1 Purpose

Sets the learning rate.

2.2.15.2 Declarations

Standard C:

```
extern void NetLearningRateSet(float pSng);
```

MS Visual Studio:

```
extern void _cdecl NetLearningRateSet(float pSng);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetLearningRateSet(float pSng);
```

2.2.15.3 Parameters

- The learning rate. It must be in the range of [0,1]. If not set, it defaults to 0.5.

2.2.15.4 Returns

Nothing.

2.2.15.5 Usage

The following C example sets learning rate to 0.5.

```
NetLearningRateSet(0.5);
```

2.2.16 NetModeGet

2.2.16.1 Purpose

Returns an integer with the current working mode.

2.2.16.2 Declarations

Standard C:

```
extern float NetModeGet();
```

MS Visual Studio:

```
extern void _cdecl NetModeGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetModeGet();
```

2.2.16.3 Parameters

None.

2.2.16.4 Returns

An integer indicating:

- 0: standard back propagation mode.
- 1: dynamic propagation mode.

2.2.16.5 Usage

The following C example gets the momentum and puts it into a variable:

```
#define MODE_STANDARD_BACKPROPAGATION 0

#define MODE_STANDARD_BACKPROP_OPTIMIZED 1

#define MODE_DYNAMIC_PROPAGATION 2

lSngTmp = NetModeGet();
```

2.2.17 NetModeSet

2.2.17.1 Purpose

Sets the current working mode. Changing the working mode could initialize the network, therefore, it is if you want to keep the model you should save it before changing the mode.

2.2.17.2 Declarations

Standard C:

```
extern void NetModeSet(int pInt);
```

MS Visual Studio:

```
extern void _cdecl NetModeSet(int pInt);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetModeSet(int pInt);
```

2.2.17.3 Parameters

- The desired working mode.

2.2.17.4 Returns

Nothing.

2.2.17.5 Usage

The following C example sets mode to dynamic.

```
#define MODE_STANDARD_BACKPROPAGATION 0

#define MODE_STANDARD_BACKPROP_OPTIMIZED 1

#define MODE_DYNAMIC_PROPAGATION 2

NetModeSet(MODE_DYNAMIC_PROPAGATION);
```

2.2.18 NetMomentumGet

2.2.18.1 Purpose

Returns the current momentum rate. Momentum rate different than 0 will add, to weights and bias, the values of the previous training iteration, multiplied by this rate.

2.2.18.2 Declarations

Standard C:

```
extern float NetMomentumGet();
```

MS Visual Studio:

```
extern void _cdecl NetMomentumGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetMomentumGet();
```

2.2.18.3 Parameters

None.

2.2.18.4 Returns

A float with the current momentum rate.

2.2.18.5 Usage

The following C example gets the momentum and puts it into a variable:

```
lSngTmp = NetMomentumGet();
```

2.2.19 NetMomentumSet

2.2.19.1 Purpose

Sets the momentum rate.

2.2.19.2 Declarations

Standard C:

```
extern void NetMomentumSet(float pSng);
```

MS Visual Studio:

```
extern void _cdecl NetMomentumSet(float pSng);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetMomentumSet(float pSng);
```

2.2.19.3 Parameters

- The momentum.

2.2.19.4 Returns

Nothing.

2.2.19.5 Usage

The following C example sets momentum to 0.9.

```
NetMomentumSet(0.9);
```

2.2.20 NetNeuronAdd

2.2.20.1 Purpose

Adds a neuron to the neural network.

2.2.20.2 Declarations

Standard C:

```
extern void _cdecl NetNeuronAdd();
```

MS Visual Studio:

```
extern void NetNeuronAdd();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetNeuronAdd();
```

2.2.20.3 Parameters

None.

2.2.20.4 Returns

Nothing.

2.2.20.5 Usage

The following C adds a neuron to the network.

```
NetNeuronAdd();
```

2.2.21 NetNeuronsAddedNumberGet

2.2.21.1 Purpose

Returns the number of neurons that have been added with NetNeuronAdd to the neural network. The maximum number of neurons that can be added is determined by the function NetCreate.

2.2.21.2 Declarations

Standard C:

```
extern int NetNeuronsAddedNumberGet();
```

MS Visual Studio:

```
extern int _cdecl NetNeuronsAddedNumberGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetNeuronsAddedNumberGet();
```

2.2.21.3 Parameters

None.

2.2.21.4 Returns

An integer.

2.2.21.5 Usage

The following C example gets the maximum number of neurons that have been added and puts it into a variable:

```
lSngTmp = NetNeuronsAddedNumberGet();
```

2.2.22 NetNeuronsMaxNumberGet

2.2.22.1 Purpose

Returns the maximum number of neurons that can be added with NetNeuronAdd to the neural network. This value is set by the function NetCreate.

2.2.22.2 Declarations

Standard C:

```
extern int NetNeuronsMaxNumberGet();
```

MS Visual Studio:

```
extern int _cdecl NetNeuronsMaxNumberGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetNeuronsMaxNumberGet();
```

2.2.22.3 Parameters

None.

2.2.22.4 Returns

An integer.

2.2.22.5 Usage

The following C example gets the maximum number of neurons that can be added and puts it into a variable:

```
lSngTmp = NetNeuronsMaxNumberGet();
```

2.2.23 NetOutputAdd

2.2.23.1 Purpose

Adds an output neuron to the neural network.

2.2.23.2 Declarations

Standard C:

```
extern void NetOutputAdd(int pLngSrc);
```

MS Visual Studio:

```
extern void _cdecl NetOutputAdd(int pLngSrc);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetOutputAdd(int pLngSrc);
```

2.2.23.3 Parameters

- Neuron number to be the output.

2.2.23.4 Returns

Nothing.

2.2.23.5 Usage

The following C example make the 100th neuron to be an output.

```
NetOutputAdd(99);
```

2.2.24 NetOutputGet

2.2.24.1 Purpose

Computes the value of an output and returns it.

2.2.24.2 Declarations

Standard C:

```
extern float NetOutputGet(int pLngOutput, int pIntCyclesControl);
```

MS Visual Studio:

```
extern float _cdecl NetOutputGet(int pLngOutput, int pIntCyclesControl);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetOutputGet(int pLngOutput, int
pIntCyclesControl);
```

2.2.24.3 Parameters

- The number of the output.
- Control of cycles (for more information please view the introduction):
    - 0: no cycles will be checked.
    - 1: cycles will be checked and avoided.

2.2.24.4 Returns

A float.

2.2.24.5 Usage

The following C example computes the value of output 10th and puts it into a float variable, without controlling cycles.

```
float lSngTmp = NetOutputGet(9, 0);
```

2.2.25 NetOutputsAddedNumberGet

2.2.25.1 Purpose

Returns the number of outputs that have been added with NetNeuronAdd to the neural network. The maximum number of outputs that can be added is determined by the function NetCreate.

2.2.25.2 Declarations

Standard C:

```
extern int NetOutputsAddedNumberGet();
```

MS Visual Studio:

```
extern int _cdecl NetOutputsAddedNumberGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetOutputsAddedNumberGet();
```

2.2.25.3 Parameters

None.

2.2.25.4 Returns

An integer.

2.2.25.5 Usage

The following C example gets the maximum number of outputs that have been added and puts it into a variable:

```
lSngTmp = NetOutputsAddedNumberGet();
```

2.2.26 NetOutputSet

2.2.26.1 Purpose

Sets the value of an output.

2.2.26.2 Declarations

Standard C:

```
extern void NetOutputSet(int pLngOutput, float pSng);
```

MS Visual Studio:

```
extern void _cdecl NetOutputSet(int pLngOutput, float pSng);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetOutputSet(int pLngOutput, float pSng);
```

2.2.26.3 Parameters

- The number of the output.
- The value.

2.2.26.4 Returns

Nothing.

2.2.26.5 Usage

The following C example sets the value of the 10th output to 0.5.

```
NetOutputSet(9, 0.5);
```

2.2.27 NetOutputsMaxNumberGet

2.2.27.1 Purpose

Returns the maximum number of outputs that can been added with NetOutputAdd to the neural network. The maximum number of outputs that can be added is determined by the function NetCreate.

2.2.27.2 Declarations

Standard C:

```
extern int NetOutputsMaxNumberGet();
```

MS Visual Studio:

```
extern int _cdecl NetOutputsMaxNumberGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetOutputsMaxNumberGet();
```

2.2.27.3 Parameters

None.

2.2.27.4 Returns

An integer.

2.2.27.5 Usage

The following C example gets the maximum number of outputs that can been added and puts it into a variable:

```
lSngTmp = NetOutputsMaxNumberGet();
```

2.2.28 NetSnapshotGet

2.2.28.1 Purpose

Sets the current state of the neural network to exactly how it was when NetSnapshotTake was called for the last time.

2.2.28.2 Declarations

Standard C:

```
extern void NetSnapshotGet();
```

MS Visual Studio:

```
extern void _cdecl NetSnapshotGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetSnapshotGet();
```

2.2.28.3 Parameters

None.

2.2.28.4 Returns

A boolean with true if everything went well.

2.2.28.5 Usage

The following C example sets the neural network to how it was when NetSnapshotTake was called for the last time:

```
NetSnapshotGet();
```

2.2.29 NetSnapshotTake

2.2.29.1 Purpose

Saves the current state of the neural network to a snapshot in memory, to be recalled later by NetSnapshotGet.

2.2.29.2 Declarations

Standard C:

```
extern int NetSnapshotTake();
```

MS Visual Studio:

```
extern int _cdecl NetSnapshotTake ();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetSnapshotTake();
```

2.2.29.3 Parameters

None.

2.2.29.4 Returns

An integer, indicating:

- 0: success.
- -1: out of memory.
- -2: there were no neurons to snapshot.

2.2.29.5 Usage

The following C example saves the current state of the neural network to a snapshot in memory, to be recalled later by NetSnapshotGet:

```
NetSnapshotTake();
```

2.2.30 NetThreadsMaxNumberGet

2.2.30.1 Purpose

Returns the current maximum number of internal threads that will be used to train the neural network. It is set to 1 by default.

Note: Currently this option is only available in Windows.

2.2.30.2 Declarations

Standard C:

```
extern int NetThreadsMaxNumberGet();
```

MS Visual Studio:

```
extern int _cdecl NetThreadsMaxNumberGet();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetThreadsMaxNumberGet();
```

2.2.30.3 Parameters

None.

2.2.30.4 Returns

An integer indicating the current maximum number of threads.

2.2.30.5 Usage

The following C example puts into a variable the current maximum number of threads:

```
int lIntTmp = NetThreadsMaxNumberGet();
```

2.2.31 NetThreadsMaxNumberSet

2.2.31.1 Purpose

Sets the current maximum number of internal threads that will be used to train the neural network. It is set to 1 by default.

Note: Currently this option is only available in Windows.

2.2.31.2 Declarations

Standard C:

```
extern void NetThreadsMaxNumberSet(int pInt);
```

MS Visual Studio:

```
extern void _cdecl NetThreadsMaxNumberSet(int pInt);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API void _stdcall NetThreadsMaxNumberSet(int pInt);
```

2.2.31.3 Parameters

An integer indicating the maximum number of threads.

2.2.31.4 Returns

Nothing.

2.2.31.5 Usage

The following C example sets the maximum number of threads that will be used to train the neural network:

```
NetThreadsMaxNumberSet(8);
```

2.3 Neuron functions

A neuron is an entity, kept in memory, which internally has a value, a bias and a delta. Neurons can be also inputs and outputs. A neuron can be connected to any other neuron in the network.

2.3.1 NeuBiasGet

2.3.1.1 Purpose

Returns the current bias of a neuron.

2.3.1.2 Declarations

Standard C:

```
extern float NeuBiasGet(int pLngNeuron);
```

MS Visual Studio:

```
extern float _cdecl NeuBiasGet(int pLngNeuron);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NeuBiasGet(int pLngNeuron);
```

2.3.1.3 Parameters

- Number of neuron.

2.3.1.4 Returns

A float.

2.3.1.5 Usage

The following C code puts into a float variable the bias value of the 10th neuron:

```
float lSngTmp = NeuBiasGet(9);
```

2.3.2 NeuDeltaGet

2.3.2.1 Purpose

Returns the current delta of a neuron.

2.3.2.2 Declarations

Standard C:

```
extern float NeuDeltaGet(int pLngNeuron);
```

MS Visual Studio:

```
extern float _cdecl NeuDeltaGet(int pLngNeuron);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NeuDeltaGet(int pLngNeuron);
```

2.3.2.3 Parameters

- Number of neuron.

2.3.2.4 Returns

A float.

2.3.2.5 Usage

The following C code puts into a float variable the delta value of the 10th neuron:

```
float lSngTmp = NeuDeltaGet(9);
```

2.3.3 NeuValueGet

2.3.3.1 Purpose

Returns the current value of a neuron.

2.3.3.2 Declarations

Standard C:

```
extern float NeuValueGet(int pLngNeuron);
```

MS Visual Studio:

```
extern float _cdecl NeuValueGet(int pLngNeuron);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NeuValueGet(int pLngNeuron);
```

2.3.3.3 Parameters

- Number of neuron to obtain its current value.

2.3.3.4 Returns

A float.

2.3.3.5 Usage

The following C code puts into a float variable the bias value of the 10th neuron:

```
float lSngTmp = NeuValueGet(9);
```

2.3.4 NeuValueUpdatedGet

2.3.4.1 Purpose

Returns the number of times that the value was calculated for a neuron, since these functions were called for the last time:

- NetCreate
- NetInitialize

2.3.4.2 Declarations

Standard C:

```
extern int NeuValueUpdatedGet(int pLngNeuron);
```

MS Visual Studio:

```
extern int _cdecl NeuValueUpdatedGet(int pLngNeuron);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NeuValueUpdatedGet(int pLngNeuron);
```

2.3.4.3 Parameters

- Number of neuron.

2.3.4.4 Returns

An integer.

2.3.4.5 Usage

The following C code puts into an integer variable the number of times that the 10th neuron’s value was updated:

```
int lIntTmp = NeuValueUpdatedGet(9);
```

2.3.5 NeuInputWeightGet

2.3.5.1 Purpose

Returns the current weight of an input of a neuron.

2.3.5.2 Declarations

Standard C:

```
extern float NeuInputWeightGet(int pLngNeuron, int pLngInput);
```

MS Visual Studio:

```
extern float _cdecl NeuInputWeightGet(int pLngNeuron, int pLngInput);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NeuInputWeightGet(int pLngNeuron, int
pLngInput);
```

2.3.5.3 Parameters

- Number of neuron.
- Number of input.

2.3.5.4 Returns

A float.

2.3.5.5 Usage

The following C code puts into a float variable the current weight of the 10th neuron's 1st input:

```
float lSngTmp = NeuInputWeightGet(9, 0);
```

2.3.6 NeuInputWeightUpdatedGet

2.3.6.1 Purpose

Returns the number of times that the weight was calculated for a neuron since the last call to the function NetInitialize.

2.3.6.2 Declarations

Standard C:

```
extern int NeuInputWeightUpdatedGet(int pLngNeuron, int pLngInput);
```

MS Visual Studio:

```
extern int _cdecl NeuInputWeightUpdatedGet(int pLngNeuron, int pLngInput);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NeuInputWeightUpdatedGet(int pLngNeuron, 
int pLngInput);
```

2.3.6.3 Parameters

- Number of neuron.
- Number of input.

2.3.6.4 Returns

An integer.

2.3.6.5 Usage

The following C code puts into an integer variable the number of times that weight was updated for 10th neuron's 1st input:

```
int lIntTmp = NeuInputWeightUpdatedGet(9, 0);
```

2.3.7 NeuInputWeightSet

2.3.7.1 Purpose

Sets the current weight of an input of a neuron.

2.3.7.2 Declarations

Standard C:

```
extern void NeuInputWeightSet(int pLngNeuron, int pLngInput, float pSng);
```

MS Visual Studio:

```
extern void _cdecl NeuInputWeightSet(int pLngNeuron, int pLngInput, float pSng);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NeuInputWeightSet(int pLngNeuron, int
pLngInput, float pSng);
```

2.3.7.3 Parameters

- Number of neuron.
- Number of input.
- Weight value.

2.3.7.4 Returns

Nothing.

2.3.7.5 Usage

The following C code sets the current weight value of the 10th neuron's 1st input:

```
NeuInputWeightSet(9, 0, 0.25);
```

2.3.8 NeuInputsNumberGet

2.3.8.1 Purpose

Returns the number of inputs of a neuron.

2.3.8.2 Declarations

Standard C:

```
extern int NeuInputsNumberGet(int pLngNeuron);
```

MS Visual Studio:

```
extern int _cdecl NeuInputsNumberGet(int pLngNeuron);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NeuInputsNumberGet(int pLngNeuron);
```

2.3.8.3 Parameters

- Number of neuron.

2.3.8.4 Returns

An integer.

2.3.8.5 Usage

The following C code puts into an integer variable the number of inputs of the 10th neuron:

```
int lIntTmp = NeuInputsNumberGet(9);
```

2.3.9 NeuBiasSet

2.3.9.1 Purpose

Sets the current bias of a neuron.

2.3.9.2 Declarations

Standard C:

```
extern void _cdecl NeuBiasSet(int pLngNeuron, float pSng);
```

MS Visual Studio:

```
extern void _cdecl NeuBiasSet(int pLngNeuron, float pSng);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NeuBiasSet(int pLngNeuron, float pSng);
```

2.3.9.3 Parameters

- Number of neuron.
- Bias value.

2.3.9.4 Returns

Nothing.

2.3.9.5 Usage

The following C code sets the current bias value of the 10th neuron:

```
NeuBiasSet(9, 0.25);
```

2.4 Set functions

If you want, as this is optional, you can create a set with your records. Then, you can train the network directly in memory from the set. This will result in faster learning and the need of a smaller number of records.

The procedure is, before training the network, you can create a set of records with the function NetSetStart. Then, every time you have set up the inputs and the outputs, call NetSetRecord to create a new record in the set. When all the records that you want to use in the training have been memorized, then you can use NetSetLearnStart, NetSetLearnContinue and NetSetLearnEnd to finish.

Remember to use NetSnapshotTake to store the network configuration while the set is being used for training and, after NetSetLearnEnd, to use NetSnapshotGet to recall the best network configuration obtained during training.

Please check a source code example to better understand the set and its great benefits.

2.4.1 NetSetDestroy

2.4.1.1 Purpose

Destroys the current set of records from memory.

2.4.1.2 Declarations

Standard C:

```
extern void NetSetDestroy();
```

MS Visual Studio:

```
extern void _cdecl NetSetDestroy();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  void _stdcall NetSetDestroy();
```

2.4.1.3 Parameters

None.

2.4.1.4 Returns

Nothing.

2.4.1.5 Usage

The following C example destroys the current set from memory:

```
//destroys current set of records

NetSetDestroy();
```

2.4.2 NetSetLearnContinue

2.4.2.1 Purpose

Makes the neural network to learn a record from the set. What it does internally is to put inputs and outputs of a certain record from the set in memory, and then makes the neural network to learn.

If the parameter pBolEstimateSuccess is true, then this function also returns the estimated rate of success of the predictions. To calculate this rate of success, the function internally predicts the output based on the input and then takes note of the result. Therefore, for the success rate to be accurate, the parameter pBolEstimateSuccess should be true for all the records of the set.

2.4.2.2 Declarations

Standard C:

```
extern float NetSetLearnContinue(int pIntRecordNumber, bool pBolEstimateSuccess,
bool *pBolThereIsNan);
```

MS Visual Studio:

```
extern float _cdecl NetSetLearnContinue(int pIntRecordNumber, bool
pBolEstimateSuccess, bool *pBolThereIsNan);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetSetLearnContinue(int
pIntRecordNumber, bool pBolEstimateSuccess, bool *pBolThereIsNan);
```

2.4.2.3 Parameters

- The number of the record to be learnt.
- A boolean indicating if the function should return the percentage of correct predictions.
- A boolean returned parameter which will indicate with true if the training generated not numbers.

2.4.2.4 Returns

A float which will be 0 if the parameter pBolEstimateSuccess is false, or if is true, a value indicating, in percentage in the range [0,1] of how many correct predictions the neural network has produced since the last call to NetSetLearnStart.

2.4.2.5 Usage

The following C example continues training with record number I, returns if there were NaN ("Not a Number") in a parameter boolean variable named lBolThereIsNan and puts into a float variable the percentage in the range [0,1] of how many correct predictions the neural network has produced since the last call to NetSetLearnStart:

```
float lSngTmp = NetSetLearnContinue(i, true, &lBolThereIsNan);
```

2.4.3 NetSetLearnStart

2.4.3.1 Purpose

Prepares the set, by randomly changing the position of the records in the set, for the self-learning process.

2.4.3.2 Declarations

Standard C:

```
extern bool NetSetLearnStart(float pSngThresholdForActive, float
pSngDeviationPercentageTarget, int pIntCyclesControl);
```

MS Visual Studio:

```
extern bool _cdecl NetSetLearnStart(float pSngThresholdForActive, float
pSngDeviationPercentageTarget, int pIntCyclesControl);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  bool _stdcall NetSetLearnStart(float
pSngThresholdForActive, float pSngDeviationPercentageTarget, int
pIntCyclesControl);
```

2.4.3.3 Parameters

- A float value indicating the threshold, typically in the range of [0,1], to consider that an output is active (when its value is equal or greater than this parameter) or inactive.
- A float value indicating the target percentage of deviation, between the real value and the predicted value, to consider that a prediction is correct. This parameter is only used when the previous parameter threshold is zero.
- Control of cycles (for more information please view the introduction):
  - 0: no cycles will be checked.
  - 1: cycles will be checked and avoided.

2.4.3.4 Returns

An integer, indicating:

- 0: success.
- 1: learning process generated NaNs (Not A Numbers), although it continued. This could mean that you should use Sigmoid as the activation function.
- 2: error managing threads.
- 3: the set has no records.

2.4.3.5 Usage

The following C example starts the self-learning process, using the previously memorized set. As it sets the threshold to zero, and indicates a Deviation Target, it will most probably will used to learn values prediction, for example to forecast time series. It does not use control of cycles:

```
#define NetLearn_Success 0

#define NetLearn_NAN 1

#define NetLearn_ThreadsError 2
```

```
#define NetLearn_SetHasNoRecords 3
```

```
NetSetLearnStart(0, mSngDeviationTarget, 0);
```

2.4.4 NetSetLearnEnd

2.4.4.1 Purpose

Marks as finished the training based on the current memorized set of records. Returns the percentage of successful predictions during training and sets the neural network into the auto adaptative mode.

2.4.4.2 Declarations

Standard C:

```
extern float NetSetLearnEnd();
```

MS Visual Studio:

```
extern float _cdecl NetSetLearnEnd();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  float _stdcall NetSetLearnEnd(int pIntRecordNumber);
```

2.4.4.3 Parameters

None.

2.4.4.4 Returns

A float value indicating, in percentage in the range [0,1] how many correct predictions the neural network has produced since the last call to NetSetLearnStart.

2.4.4.5 Usage

The following C example finishes training with the memorized, activates the auto adaptative mode and puts into a float variable the percentage in the range [0,1] of how many correct predictions the neural network has produced since the last call to NetSetLearnStart:

```
float lSngTmp = NetSetLearnEnd();
```

2.4.5 NetSetPrepare

2.4.5.1 Purpose

Creates the memory structure for a set of inputs and outputs for the neural network. This function is not strictly necessary, as calling multiple times to NetSetRecord will also reserve memory for each record, but slower than using NetSetPrepare once and before the NetSetRecord calls. Therefore, NetSetPrepare is used before than the multiple calls to NetSetRecord, as it will be faster by reserving all the needed memory for the whole set at once.

2.4.5.2 Declarations

Standard C:

```
extern int NetSetPrepare(int pIntTotalNumberOfRecords);
```

MS Visual Studio:

```
extern int _cdecl NetSetPrepare(int pIntTotalNumberOfRecords);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetSetPrepare(int
pIntTotalNumberOfRecords);
```

2.4.5.3 Parameters

None.

2.4.5.4 Returns

- True: memory was reserved successfully.
- False: out of memory.

2.4.5.5 Usage

The following C example reserves into memory all the set:

```
NetSetPrepare();
```

2.4.6 NetSetRecord

2.4.6.1 Purpose

Inserts into the set, as a new record, the current inputs and outputs of the neural network. For faster operation of NetSetRecord, it is recommended to previously have called NetSetPrepare for the total number of records that you intend to create via NetSetRecord. By calling NetSetPrepare previously to NetSetRecord, all the necessary memory will be reserved at once, instead of per record with NetSetRecord.

2.4.6.2 Declarations

Standard C:

```
extern int NetSetRecord();
```

MS Visual Studio:

```
extern int _cdecl NetSetRecord();
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  int _stdcall NetSetRecord();
```

2.4.6.3 Parameters

None.

2.4.6.4 Returns

- An integer: indicating the number of records memorized since the last call to NetSetDestroy.
- -1: if error, typically, out of memory.

2.4.6.5 Usage

The following C example memorizes in the set, as a new record, the current inputs and outputs of the neural network, and puts the total number of records memorized in an integer variable:

```
int lIntTmp = NetSetRecord();
```

2.4.7 NetSetStart

2.4.7.1 Purpose

Initiates the set, and:

- Stops the auto adaptive mode.

- If the NetInitialize parameter is set to true, it will call NetInitialize with the DefaultVal parameter.

2.4.7.2 Declarations

Standard C:

```
extern bool NetSetStart(float pSngDefaultVal, bool pBolNetInitialize);
```

MS Visual Studio:

```
extern bool _cdecl NetSetStart(float pSngDefaultVal, bool pBolNetInitialize);
```

MS Visual Studio compatible with win32 COM:

```
extern "C" AnaimoAI_API  bool _stdcall NetSetStart(float pSngDefaultVal, bool
pBolNetInitialize);
```

2.4.7.3 Parameters

- A float with the default value to initialize the network. Put 0 for random.
- A boolean to indicate the network should be initialized (internally calls to NetInitialize).

2.4.7.4 Returns

True if it had enough memory.

2.4.7.5 Usage

The following C example initializes the set and the neural network (internally calls to NetInitialize):

```
// initializes set
NetSetStart(DEFAULT_VALUE_TO_INITIALIZE_WEIGHTS, true);
```

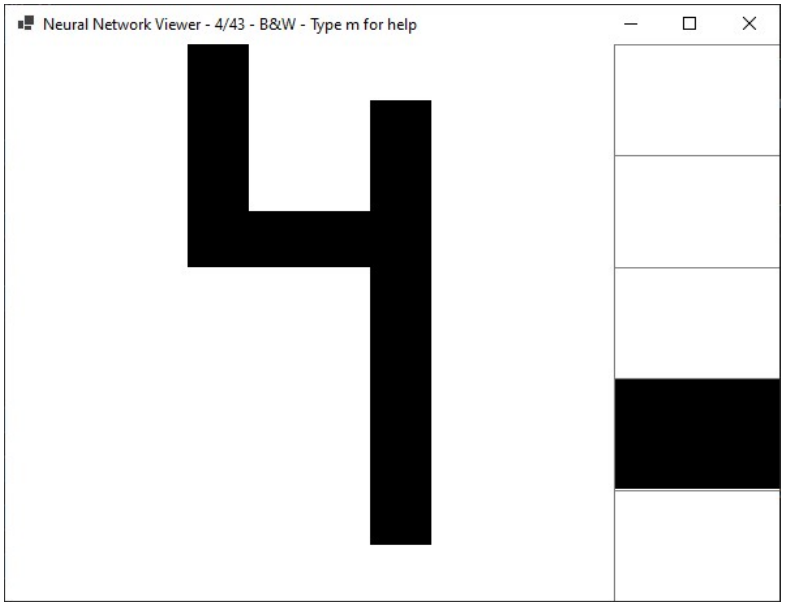
3 Example: NNViewer

It is very important to have an example to understand how the library can be used.

NNViewer is a complete, fully functional, easy to understand, source code of an example application. Allows drawing inputs and outputs to train the neural network and see how it learns the patterns.

For this purpose, the source code of the example NNViewer is provided in 2 versions:

- MS Visual Studio VB.NET (source code requires MS Visual Studio Community version 2019 or higher).
- Legacy MS VB6 (Win32 COM compatible).



NNViewer learning the 4 number. Source code is provided.

3.1 How does it work?

You can type the M key at any time to obtain manual about the available commands. Most of them should be self-explanatory.

NNViewer has a graphical user interface showing, on the left side, a grid with the inputs, and on the right side, a grid with the outputs.

Every input and output (every cell of the grids) can have a value in the range from 0 to 1 inclusive. The value of each cell will be shown with a color, from white for 0 to black for 1. A value of a cell of 0.5 will be shown in grey.

The user can change the inputs at any time by clicking with the mouse on an input cell. The user can change the outputs in the same way, only when the application is not on "thinking" mode. In the "thinking" mode (after pressing the T key), the outputs will be colored by the neural network.

You can create and delete pages, clear them, and navigate through them. Once you have all the pages of inputs and outputs that you want, you can make the neural network to learn them.

Once the neural network has learnt your pages, you can switch to "thinking" mode with the T key, and then the neural network will show you, on the outputs, which outputs it understands from the current inputs.

When the neural network is in "thinking" mode (T key), you can create a new page with the N key, draw new inputs and see in real time the outputs that the neural network predicts.

3.2 Example files

Once you have created your pages of inputs and outputs, you can save them with the S key.

The application comes with 4 examples files that can be loaded. These files are internally CSV files but renamed to extension NNV:

- **numbers.nnv**: a few handwritten numbers, in low quality, to show how the neural network can learn with very few and poor-quality inputs and outputs.

- **ranges.nnv**: marks ranges of inputs and outputs at the same height. The purpose of this file is to show how the neural network learns to mark outputs at the same height than inputs. Please note that 5% of the outputs are wrong, and that the neural network, in "thinking" mode, will mark the correct answer for the outputs. In other words, it will be detecting anomalies.
- **symbols.nnv**: a few pages of symbols to show how the neural network can learn with very few inputs and outputs. Once learned (with the A key), you can create a new page (with N key), draw a similar symbol, and see how the neural network shows the output.
- **xor.nnv**: very simple example of a XOR operation. Choose 2 layers of NN and learn with 3.000 epochs and it will learn the operation.

4 Support

In case you need support, please contact us at https://anaimo.com/support/