CogniPat SDK
User's Manual

SOFTWARE DEVELOPMENT KIT FOR PATTERN LEARNING AND RECOGNITION WITH NEUROMEM SILICON NETWORK

Version 4.4
Revised 02/23/2018

CONTENTS

CogniPat SDK is a product of General Vision, Inc. (GV)

This manual is copyrighted and published by GV. All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of GV.

For information about ownership, copyrights, warranties and liabilities, refer to the document Standard Terms And Conditions Of Sale or contact us at www.general-vision.com.

INTRODUCTION

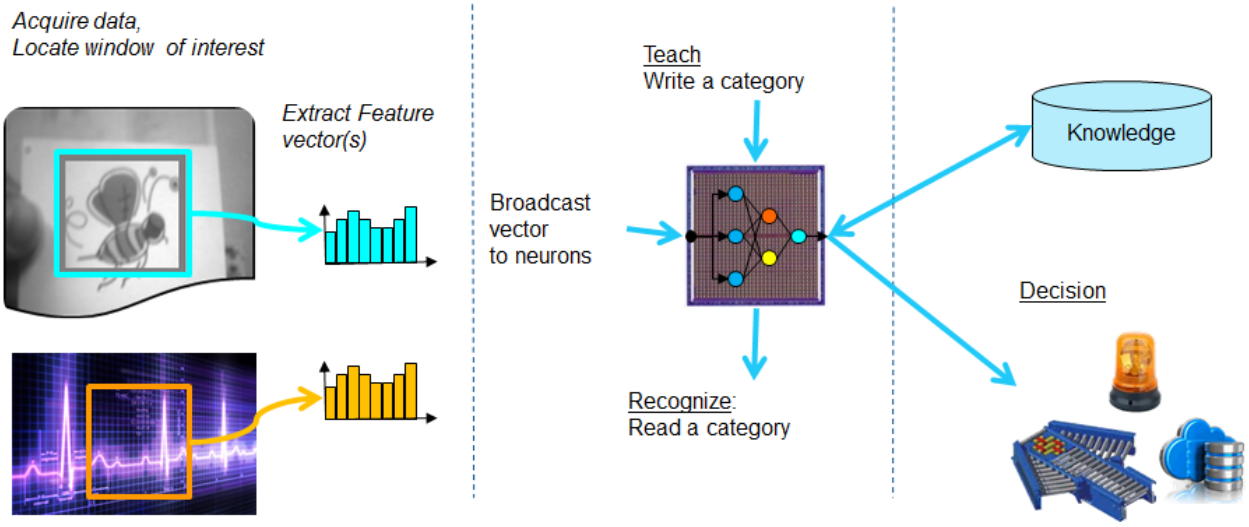
NeuroMem is a pattern recognition accelerator chip which is also trainable in real-time by learning examples. The interface to a chain of neurons of any size comes down to 4 simple main operations:

- Learn a vector
- Classify a vector
- Save the knowledge built by the neurons
- Load a knowledge into the neurons

Data coming from a variety of sources can be converted into pattern vectors which are then broadcasted to the neural network for either learning or recognition. All you must do is focus on the quality of the input signals, the selection of relevant and discriminant feature extractions.

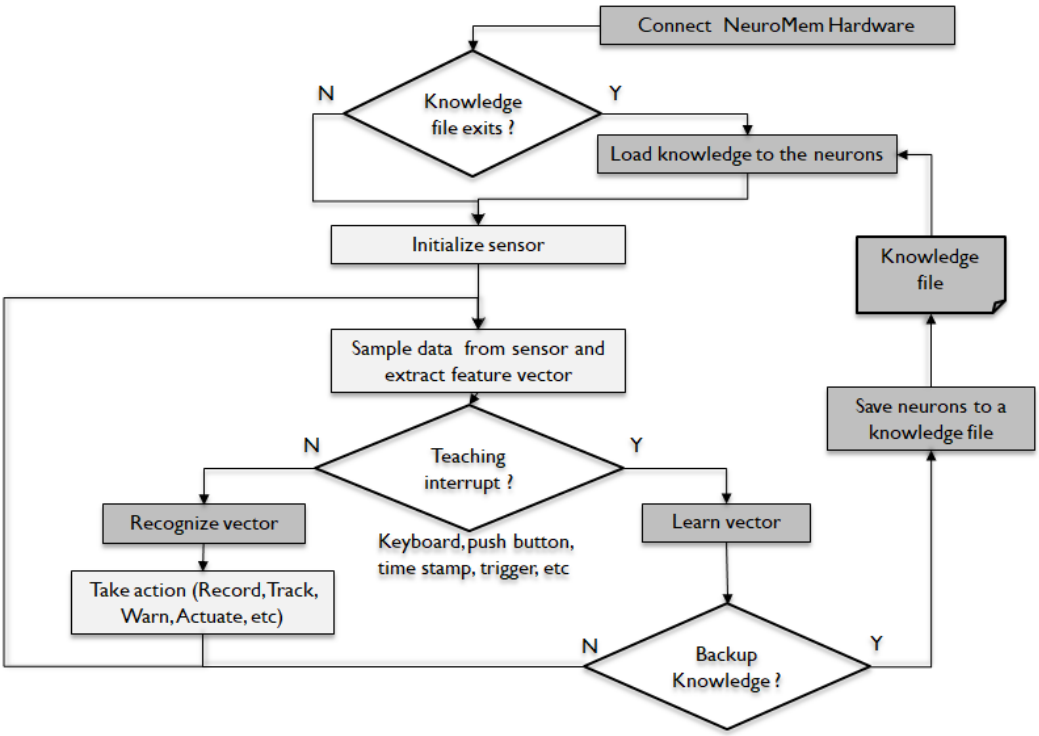
The NeuroMem neurons handle the automatic learning of your examples and associated categories, the recognition of new patterns, the detection of uncertainty cases if any, the detection of drifts or decrease of confidence, the reporting of novelties and/or anomalies.

Finally, you must format the response of the neurons to convert it into a global decision or action for your application, without forgetting to save the knowledge built by the neurons for backup purposes or distribution to other systems.



For detailed information about the neurons and their behavior, you can also refer to the NeuroMem Technology Reference Guide.

Typical Application block diagram



What is a feature vector?

Lets' take the example of a dataset describing Iris flowers:

Example	Feature 1	Feature 2	Feature 3	Feature 4	Category
	Sepal length	sepal width	petal length	petal width	species
1	6.4	2.8	5.6	2.2	2
2	5.0	2.3	3.3	1.0	1
3	4.9	2.5	4.5	1.7	2
4	4.9	3.1	1.5	0.1	3
5	5.7	3.8	1.7	0.3	3

- The **Features** columns contain characteristics of an example
- The **Category** column (species) is the label to learn or predict; It is naturally a string, but the digital world relies on numeric values. Therefore, it is mapped to a number.
    - 1 = versicolor
    - 2 = virginica
    - 3 = setosa
- A **Feature Vector** is a series of features which can be heterogeneous (like in this example), or on the contrary represent a time or spatial series, etc.

The neurons are agnostic to the data source which means that they ready to learn and recognize feature vectors extracted from text (parsing), discrete measurements, audio signal, video signal, digital images, etc. In the case of images, it can be a subsampling of pixels within a region of interest, a histogram, some histogram of gradients within

a patch of pixels, etc. In the case of an audio or biosensor signal, the feature vector can be a set of signal samples taken at a specific sampling rate, a histogram of peaks or zero crossing, or a power spectrum, etc.

In the NeuroMem API, a feature vector is an array of integer values with a dimension less than or equal the memory capacity of the NeuroMem neurons (256 bytes for the CM1K and NM500 chips; 128 bytes for the Intel QuarkSE/Curie module). Note that if the memory depth is 8-bit and not 16-bit, the upper byte of the vector data is discarded.

How do the neurons learn and recognize?

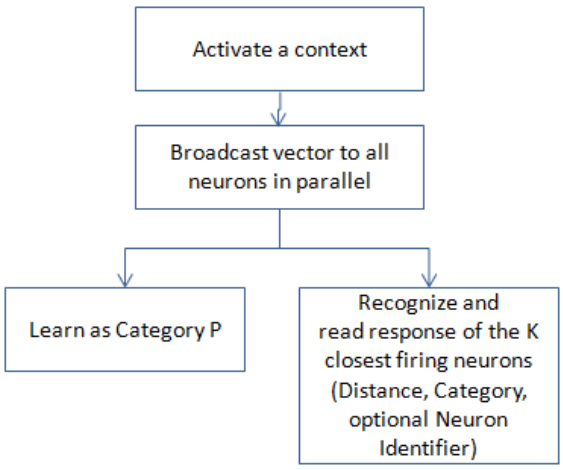
A feature vector represents a dimension (different from its length) which we associate to the notion of context. For example, a color histogram extracted from a pixel area may have the same length as a monochrome histogram, but its dimensionality is richer because it will contain information about the red, green and blue intensities of the region.

The only manipulation of a feature vector consists of broadcasting it to the neurons. From there, each neuron with the selected **Context**, will calculate its distance from the vector to the model stored in its memory.

The next operation can be to either learn the feature vector or recognize it.

Learning means associating a **Category** to the vector. This category can be supplied by a supervisor or be calculated by an unsupervised algorithm.

Recognition consists of querying the neurons to read their responses. Note that prior to broadcasting the vector, the network can be set as a Radial Basis Function (RBF) classifier which is the default, or as a simple K-Nearest Neighbor (KNN). Since the NeuroMem neurons auto-sort themselves per order of decreasing confidence, the 1st query returns the Category and Distance of the neuron with closest model to the input vector. The 2nd query returns the response of the 2nd most confident neuron, etc. How many K neurons to query depends on your application.



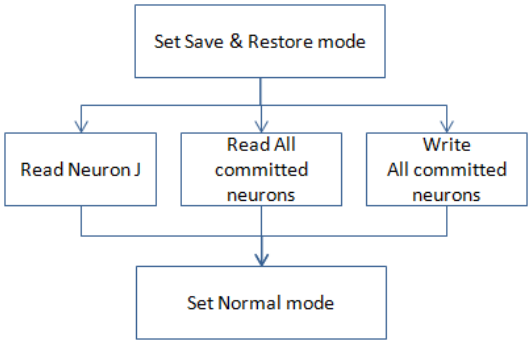
Our NeuroMem RBF tutorial is a simple application demonstrating how the neurons build a decision space autonomously by learning examples and how they recognize new vectors with ability to report cases of unknown and uncertainties too. For the sake of a simplicity, the decision space is limited to a 2D space but the mechanism is the same to recognize a (X1, X2) vector as well as an (X1, X2, … X127) vector which is the maximum length supported by the neurons of the CM1K.

How to use the knowledge built by the neurons?

The NeuroMem network can be set to Save and Restore (SR) mode to save the knowledge built by the neurons for backup purposes, or to distribute it to other NeuroMem platforms.

Under the SR mode, the neurons become dummy memories. The automatic model generator and search-and-sort logic are disabled.

The following diagram shows the possible functions which can be executed in Save & Restore mode:



For more information of the NeuroMem technology, please refer to the NeuroMem Technology Reference Guide.

PACKAGE CONTENT

Supported hardware

The CogniPat SDK is a Dynamic Link Library which interfaces to a chain of NeuroMem neurons for learning, classification, knowledge saving and restore. Several versions of the DLL exist. They have the same entry points but include different drivers for access to the NeuroMem hardware:

CogniPat_NeuroShield	Cypress USB serial driver (and simulation if board not found)
CogniPat_NSnK	FTDI USB driver for NeuroStack (and simulation if board not found)
CogniPat_Simu	Cycle accurate simulation of the NeuroMem network of 1024 neurons

Bin Folder

- Win32/ CogniPat_Simu.dll, CogniPat_NSnK.dll, CogniPat_NeuroShield.dll
- x64/ CogniPat_Simu.dll, CogniPat_NSnK.dll, CogniPat_NeuroShield.dll
- CogniPat.h Header defining the entry points to the DLL
- CogniPat.cs Class defining the entry points to the DLL for C# interface
- CogniPatClass.m Class defining the entry points to the DLL for MatLab interface

Examples

All sets of examples include the Simple Script described in this manual which helps understand the behavior of the neurons to learn new patterns, and classify new patterns with or without uncertainty, etc.

Depending on your hardware, each example can be executed with the CogniPat_Simu.dll or CogniPat_NSnK.dll. This selection is made as follows depending on the programming language:

	Where do I select the CogniPat DLL?	**Location**
C++	Linked library	Project/Properties/Linker/Input (declared in C++ project)
C#	DLL name in DLL Import	CogniPat.cs (included in your C# project)
MatLab	DLL path in the CogniPat Class method	CogniPatClass.m (instantiated in your MatLab script)
Python	DLL path on the Ctypes.CDLL call	Beginning of each example

WHAT IS NEW IN THIS VERSION?

Addition of a driver for the NeuroShield board used as a USB device

The platform type passed to the Connect function is now defined as follows:
0, Simulation (dependency to the CogniPat_Simu.dll)
1, NeuroStack (dependency to the CogniPat_NSnK.dll)
2, NeuroShield (dependency to the CogniPat_NeuroShield.dll)

Vector, Model and Neuron data type have been changed from byte to integer

For practicality, the functions manipulating vector, model and neuron data now expect an array of integer as inpu. Still the neurons of the CM1K and NM500 have a memory size limited to a byte array, so the upper byte of the input arrays are expected to be equal to 0 when interfacing to a NeuroMem network of CM1K or NM500.

Simulation of larger neurons

The “Simulation” platform allows to simulate digital neurons with a memory larger than 256 bytes. This feature is still under testing so please report any error or inconsistency you may encounter.

The selection of the simulation platform is done by using the platform value 0 in the Connect function. By default, the function allocates 1024 neurons of 256 bytes memory. The number of neurons can be changed by passing a DeviceID different from 0 to the Connect function. The size of the neurons can be changed by executing the new function SizeNeuronsandReset.

HARDWARE INTERFACE FUNCTIONS

int Version();

Returns the version of the DLL.

int Connect(int Platform, int DeviceID);

Establishes communication with your NeuroMem platform and if applicable a specific DeviceID. This function returns 0 if the connection is successful.

The function also detects the number of neurons available in the selected platform and clear their content including registers and memory. Its execution takes approximately 1 second per thousand neurons, but it is necessary to size the network in the case of architecture where NeuroMem expansion modules can be stacked. The network size can then be read with the getNetworkInfo function.

Platform code	**DeviceID definition**
0= Simulation of NeuroMem neurons	Code for the simulated device: 0 = CM1K 1 = NS4K_chain of 4096 2 = CurieNeurons 3 = NM500
1=NeuroStack board (4096 neurons of 256 bytes)	N/A. Default is 0.
2= V1KU camera (1024 neurons of 256 bytes)	DeviceID is the USB device number connected to your host. Default is 0.

int Disconnect();

Closes the communication with the current device.

void GetNetworkInfo(int *neuronAvailable, int *neuronMemorySize)

- neuronsAvalaible, number of neurons available
- neuronMemorySize, memory capacity of each neuron in bytes

NETWORK INITIALIZATION AND GLOBAL SETTINGS

Initialization

```
int  ClearNeurons();
```

Clear the entire contents of all the neurons including registers and memory cells. The function returns the number of committed neurons which should be equal to zero. It also resets the Global Context, Min and Max Influence fields to their respective default values 1, 2 and 16,384.

```
Forget();
```

Clears the category register of all neurons so they become uncommitted. The neurons are ready to start learning again with the default Global Context, Min and Max Influence fields to their respective default values 1, 2 and 16,384. Note that the memory of the neurons is not cleared. It is best to use the ClearNeurons function whenever possible, but the Forget function is provided for applications being speed sensitive.

Working with multiple contexts

At initialization, all neurons have a context equal to the value 1. If an application manipulates vectors belonging to different contexts, the proper context value must be written to the Global Context register prior to broadcasting the vectors belonging to the new context. For more details, refer to the NeuroMem Technology Reference Guide.

The Context value is encrypted on the lower byte as follows:

Bit 7 = Norm (0=L1, 1=LSUP)

Bit [6-0]= context value

The neurons with a context value lesser than 128 calculate their distance using the L1 norm.
The neurons with a context value greater than 128 applies the LSup norm.

With a change of Context often comes a change of dimension of the feature, so the library allows changing the minimum and maximum influence fields at the same time. The default MAXIF is 16384, and the default MINIF is 2.

```
SetContext(int context, int minif, int maxif);
GetContext(int* context, int* minif, int* maxif);
```

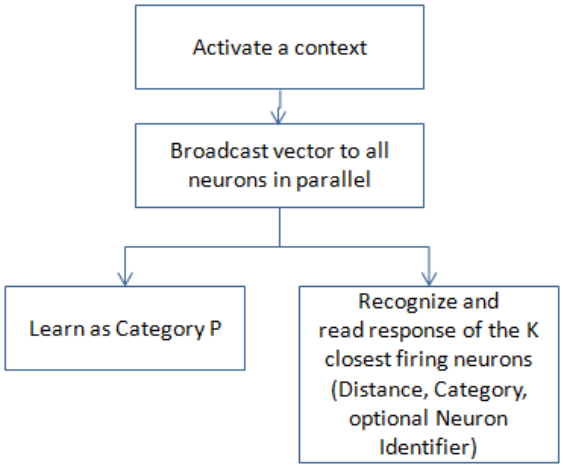
Choice of classifier

The neurons can classify patterns as a Radial Basis Function (RBF) or K-Nearest Neighbor (KNN) Classifier. Use one of the following two functions to toggle between the two modes.
RBF is the default classifier at initialization of the neurons. RBF allows to learn examples and build a highly non-linear decision space. RBF also introduces the powerful notion of “unknown” and “uncertainty”. Refer to chapter 4.3 and 4.4 of the NeuroMem Technology Reference Guide for more details.

```
SetRBF();
SetKNN();
```

LEARNING AND RECOGNITION FUNCTIONS

Under Normal mode, the neurons can learn and recognize patterns as shown in the following diagram:



void Broadcast(int vector[], int length)

Broadcast the input vector composed of "length" components to the neurons. The value length is cropped to the neuron memory size if necessary. Note that the function takes an array of int as input, but the upper byte is expected to be equal to 0 since the existing NeuroMem chips are limited to byte array memories.

If the length of the vector is L, this function executes L-1 Write COMP followed by a single Write LCOMP. It can be used whether you intend to learn or recognize the vector.

Considerations before calling this function:

- Change the Global Context Register (GCR) if the pattern belongs to a context other than the active context
- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier
- Change the Min Influence Field and Max Influence Field

Considerations after calling this function:

- Read the Network Status Register to find out if the vector is recognized or not, with uncertainty or not
- Write the Category register to learn the vector
- Read the Distance, followed by Category to classify the vector

Int(ncount) Learn(int vector[], int length, int category);

Learn the input vector composed of "length" components as belonging to the "category". The value length is cropped to the neuron memory size if necessary. Note that the function takes an array of int as input, but the upper byte is expected to be equal to 0 since the existing NeuroMem chips are limited to byte array memories. Category can range between 1 to 32766. A category of 0 can be used to teach a counter example or a background example.

The function returns the number of committed neurons (ncount). Note that this number does not increase necessarily after each execution of the Learn function. Ncount will increase only if the vector and its associated category represents novelty to the commited neurons.

Considerations before calling this function:

- Change the Global Context Register to match the context represented by the input vector
- Change the Min Influence Field and Max Influence Field
- Make sure the network is in RBF mode (Write NSR 0).
    - The KNN mode is not appropriate for learning since it will create a new neuron each time a new category value is taught and do nothing more. The RBF mode must be used to build a decision space modeling multiple categories and also with areas of "unknown" or "uncertainties" which are essential for true artificial intelligence with voluntary redundancy for accuracy, context awareness, hypothesis generation and more.
    - The Save and Restore mode is not compatible with the learning mode.

Int(nsr) BestMatch(int vector[], int length, int* distance, int* category, int* nid);

Broadcast the vector to the neurons and read the response of the first "firing" neuron, if any, including its distance, category and identifier. The value length is cropped to the neuron memory size if necessary. Note that the function takes an array of int as input, but the upper byte is expected to be equal to 0 since the existing NeuroMem chips are limited to byte array memories.

The function returns the Network Status Register and its lower byte can be decoded as follows:

- NSR=0, the vector is not recognized by any neuron (UNKnown)
- NSR=8, the vector is recognized and all firing neurons are in agreement with its category (IDentified)
- NSR=4, the vector is recognized but the firing neurons are in disagreement with its category (UNCertain)
- NSR=32, the network is in KNN mode

Distance represents the distance value between vector and the firing neuron with the closest model stored in its memory (i.e. top firing neuron). Distance is calculated by the neurons according to the Norm assigned to the neuron at the time it was committed (bit7 of the Context register). If no neuron fires, Distance=0xFFFF.

Category of the top firing neuron or 0xFFFF is non existent. Bit 15 is reserved for the Degenerated flag, so the effective category value is coded on bit[14:0] and can be obtained with an AND mask with 0x7FFF.

NID is the identifier of the top firing neuron. NID can range between 1 and the number of neurons available in the network. If no neuron fires, Identifier=0.

Considerations before calling this function:

- Change the Global Context Register to match the context represented by the input vector
- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier, or to turn ON/OFF the Save-and-Restore mode.

Considerations after calling this function:

- Change the Network Status Register (NSR) to turn OFF the KNN classifier, if selected, prior to the next learning operation.
- If the NSR indicates a case of uncertainty (value 4 or 36), you can immediately execute a series of Read(DIST) + Read(CAT) to obtain the response of the next closest firing neurons, and this until you read a DIST=0xFFFF.

Int(responseNbr) Recognize(int vector[], int length, int K, int distance[], int category[], int Nid[]);

Broadcast the vector to the neurons and read the response of the first K "firing" neurons, if applicable, including their distance, category and identifier. The value length is cropped to the neuron memory size if necessary. Note that the function takes an array of int as input, but the upper byte is expected to be equal to 0 since the existing NeuroMem chips are limited to byte array memories.

The function returns the number of firing neurons or K whichever is the smallest. For example, if K=3, but only 2 neurons fire, the function returns, the value 2.

Distance represents the distance value between vector and the firing neuron with the closest model stored in its memory (i.e. top firing neuron). Distance is calculated by the neurons according to the Norm assigned to the neuron at the time it was committed (bit7 of the Context register). If no neuron fires, Distance=0xFFFF.

Categories of up the K closest firing neurons or 0xFFFF if non-existent. Bit 15 is reserved for the Degenerated flag, so the effective category value is coded on bit[14:0] and can be obtained with an AND mask with 0x7FFF.

NID is the identifier of the top firing neuron. NID can range between 1 and the number of neurons available in the network. If no neuron fires, Identifier=0.

Considerations before calling this function:

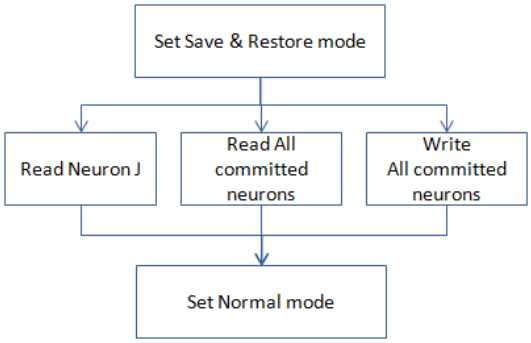
- Change the Global Context Register to match the context represented by the input vector
- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier, or to turn ON/OFF the Save-and-Restore mode.

Considerations after calling this function:

- Change the Network Status Register (NSR) to turn OFF the KNN classifier, if selected, prior to the next learning operation.

LOADING AND RETRIEVAL OF NEURONS' CONTENT

Under the SR mode, the automatic model generator and search-and-sort logic are disabled. The neurons become dummy memories but can be read or written in the least amount of time. This SR mode is essential to transfer knowledge bases between hardware platforms, or make backup prior to learning additional examples. The following diagram shows the possible functions which can be executed in Save & Restore mode:



Int(ncount) CommittedNeurons();

Report the number of committed neurons.

void ReadNeuron(int neuronID, int context, int model[], int aif, int minif, int category);

neuronID is not an index starting from 0 but the neuron Identifier starting at 1.

Model[] is the pattern stored in the neuron memory. Note that the function returns an array of int, but the upper byte is expected to be equal to 0 since the existing NeuroMem chips are limited to byte array memories.

Context is the neuron context register and bit 7 represents the Norm used by the neuron for distance calculation.
AIF is the Active Influence Field of the neuron (current value at the time of the Read. May decrease in the future if new vectors with the same context as the neuron are taught)

Minif is the neuron's Minimum Influence Field assigned at the time the neuron got committed.

Cat is the neuron's Category assigned at the time the neuron got committed.

NeuronID can range from 1 to the number of committed neurons. If neuronID greater, the function returns the contents of the RTL (Ready-To-Learn) neuron.

Considerations when calling this function:
If an application is dealing with vectors of a length L lesser than the size of the neuron's memory, only the first L values of the model[] are significant. Indeed, if the ClearNeurons function was executed prior to learning the first vector, the remaining components from L+1 to 256th will be equal to 0, otherwise they will report values loaded at an earlier stage into the neurons.

Int(ncount) ReadNeurons(int *neurons);

Read all the committed neurons and return their content in a array with the format described below.

The function returns the number of committed neurons (ncount)

Neurons is an array of ncount * neuronData integers
NeuronData is an array of (neuronSize + 4) integers as follows:

- NeuronData[0]= NCR, Neuron Context Register
- NeuronData[1,NeuronSize] = neuron's memory or NeuroSize components
- NeuronData[NeuronSize+1]= AIF, Active Influence Field
- NeuronData[NeuronSize+2]= MINIF, Minimum Influence Field
- NeuronData[NeuronSize+3]= CAT, Category

Considerations when calling this function:

- Note that the function returns the neurons' memory as an array of int, but the upper byte is expected to be equal to 0 since the existing NeuroMem chips are limited to byte array memories.
- If an application is dealing with vectors of a length L lesser than the size of the neuron's memory, only the first L components are significant. Indeed, if the ClearNeurons function was executed prior to learning the first vector, the remaining components from L+1 to 256th will be equal to 0, otherwise they will report values loaded at an earlier stage into the neurons.

Int(ncount) WriteNeurons(int *neurons, int ncount);

Load the input array "neurons" to the digital neurons dispatching the values according to the format described below.

The function returns the actual number of committed neurons (ncount) after the upload to the NeuroMem chip(s).

Neurons is an array of ncount * neuronData integers
NeuronData is an array of (neuronSize + 4) integers as follows:

- NeuronData[0]= NCR, Neuron Context Register
- NeuronData[1,NeuronSize] = neuron's memory or NeuroSize components
- NeuronData[NeuronSize+1]= AIF, Active Influence Field
- NeuronData[NeuronSize+2]= MINIF, Minimum Influence Field
- NeuronData[NeuronSize+3]= CAT, Category

Considerations when calling this function:
The content of the neurons must be considered as a whole (aka a knowledge). Saving the neurons to an array, editing or removing a single neuron and restoring this array can be very detrimental and produce inaccurate classification! Remember that during the learning, the neurons are all interconnected and have auto-adjusted their influence fields based on their common knowledge.

Int(ncount) SaveNeurons(char *filename);

Save the contents of all the neurons to a file with the format below. The function returns the number of committed neurons saved to the file.

File header:

Header[0]= format version number;
Header[1] = maxveclength; // default=256
Header[2] = ncount;
Header[3] = 0; //reserved

NeuronData is an array of (neuronSize + 4) integers as follows:

- NeuronData[0]= NCR, Neuron Context Register
- NeuronData[1,NeuronSize] = neuron's memory or NeuroSize components
- NeuronData[NeuronSize+1]= AIF, Active Influence Field
- NeuronData[NeuronSize+2]= MINIF, Minimum Influence Field
- NeuronData[NeuronSize+3]= CAT, Category

The function returns how many neurons were saved, or 0xFFFF if a problem with the file.

Int(ncount) LoadNeurons(char *filename);

Load the contents of all the neurons from a file. The function returns the number of committed neurons loaded from the file.

The function returns the number of neurons loaded to the NeuroMem chip(s), or an error code as follows:
0xFFFF, problem with the file
0xFFF1, unknown file format
0xFFF2, incompatible with neuron length
0xFFF3, incompatible with neuron capacity

Int(ncount) GetNeuronsInfo(int* MaxContext, int* MaxCategory)

This function analyzes the content of the neurons and retrieves the maximum values of the neuron context and category registers. The function returns the number of committed neurons.

void SurveyNeurons(int Context, int MaxCatValue, int CatHisto[], int CatDegHisto[])

This function analyzes the content of the neurons of a given Context and with a category less than or equal to MaxCatValue.

It generates the histogram of the neurons per category and the histogram of the degenerated neurons per category. The 2 histograms must be sized as arrays of dimension MaxCatValue.

The function returns the number of neurons matching the Context and MaxCatValue criteria.

VECTOR BATCH PROCESSING

Supervised Learning

```
int LearnVectors(int *vectors, int vectNbr, int vectLen, int *categories,
bool ResetBeforeLearning, bool iterative);
```

Learn an array of vectors with their asscoiated categories. The “vectNbr” vectors are all composed of the same number of components “vectLen”. It returns the total number of committed neurons.

- vectors= int Array with a size vectNbr * vectLen
- categories= int Array with a size of vectNbr

The flag **ResetBeforeLearning** allows to reset the network prior to learning. Note that this reset is a simple Write FORGET which takes a simple clock cycle, and not a ClearNeurons which erases the content of the neurons’ memory but takes 258clock cycles. If ResetBeforeLearning is set to 1, the function preserves the value of the minimum and maximum Influence fields.

The flag **Iterative** executes the learning of the vectors repeatedly until the number of committed neurons remains steady between two passes. This option produces a knowledge which is invariant to the order of the vectors, but might take multiple iterations and consequently some additional time.

Considerations before calling this function:

- Clear the neurons to pad their memory with zero in case vectLen is less than 256
- Set the NSR register to 16 or 0
    - If NSR=16, the vectors are loaded in Save and Restore mode
    - If NSR=0, the vectors are learned using the RBF classifier
- Change the Global Context Register to match the context represented by the input vectors
- Change the Maxif : The Maxif is used when learning in normal mode (NSR=0). The higher the Maxif, the faster the teaching, but the higher the probability of false positive
- Change the Minif : The Minif is used when learning in normal mode (NSR=0). The higher the Minif, the more degenerated neurons and zones of uncertainty.

Unsupervised Learning

```
int BuildCodebook(int *vectors, int vectNbr, int vectLen, int CatAllocMode,
int InitialCategory, bool WithoutUncertainty);
```

Learn an array of vectors with unsupervised allocation of categories according to the selected CatAllocMode. The vectNbr vectors must be composed of the same vectLen number of components. This function does not clear the existing knowledge. It returns the total number of committed neurons.

The BuildCodebook function broadcast each vector to the neurons and a new neuron is automatically committed if this vector is not recognized (unknown). The category can be a simple increment from the category of the last committed neuron, or a function of the vector content. Upon termination of the function, the content of the newly committed neurons represents a codebook or a dictonnary of discriminant codes. The value of the Maximum Influence Field when the function is called moderates the frequency of unknown recognitions. If too high, the

codebook might be limited to too few neurons with large influence fields. If too low, the size of the codebook might be close to the initil vectNbr.

- vectors= int Array with a size vectNbr * vectLen
- CatAllocMode: Defines which category to assign to a vector which is not recognized by the currently committed neurons:
    - 0: constant value
    - 1: auto-increment by 1
    - 2: maximum delta between the vectLen components
    - 3: average value of the vectLen components
    - 4: index of the input vector. This information can help retrieve the temporal position of an audio vector, or spatial position of an image vector, etc.
- InitialCategory
    - Initial value for the CatAllocMode 0 and 1
- waiveUncertainty
    - If set to 1, this flag forces the learning of a vector which is recognized by the currently committed neurons but with an uncertainty or disagreement between at least two firing neurons. This will force the reduction of the influence fields of some committed neurons

Considerations before calling this function:

- Clear the neurons to pad their memory with zero in case vectLen is less than 256
- Set the NSR register to 16 or 0
    - If NSR=16, the vectors are loaded in Save and Restore mode
    - If NSR=0, the vectors are learned using the RBF classifier
- Change the Global Context Register to match the context represented by the input vectors
- Change the Maxif : The Maxif is used when learning in normal mode (NSR=0). The higher the Maxif, the faster the teaching, but the higher the probability of false positive
- Change the Minif : The Minif is used when learning in normal mode (NSR=0). The higher the Minif, the more degenerated neurons and zones of uncertainty.

```
int Clusterize(int  *vectors, int vectNbr, int vectLen);
```

Learn the input vectors assigning their categories autonomously by increment of 1. The vectNbr vectors must be composed of the same vectLen number of components. This function clears the existing knowledge prior to learning. It returns the total number of committed neurons.

The Clusterize function broadcast each vector to the neurons and a new neuron is automatically committed if this vector is

a) not recognized
b) recognized but at a distance greater than MINIF from an existing model.

The category is automatically assigned as a function of the recognition staus

a) an increment from the highest category value
b) the category of the closest firing neuron

Upon termination of the function, the content of the newly committed neurons features discriminant models grouped into N categories, with N being the highest category value. The values of the Maximum and Minimum Influence Fields when the function is called moderate the frequency of recognitions of type a) and b) and consequently the value N or number of clusters.

- vectors= int Array with a size vectNbr * vectLen

Considerations before calling this function:

- Save the current knowledge which will be erased otherwise.
- Clear the neurons to pad their memory with zero in case vectLen is less than 256
- Set the NSR register to 16 or 0
    - If NSR=16, the vectors are loaded in Save and Restore mode
    - If NSR=0, the vectors are learned using the RBF classifier
- Change the Global Context Register
- Change the Maxif : The Maxif is used when learning in normal mode (NSR=0). The higher the Maxif, the faster the teaching, but the higher the probability of false positive
- Change the Minif : The Minif is used when learning in normal mode (NSR=0). The higher the Minif, the more degenerated neurons and zones of uncertainty.

Classification

```
void RecognizeVectors(int *vectors, int vectNbr, int vectLen, int K, int *distances, int *categories, int *nids);
```

Recognize an array of vectors and return the response of the first K firing neurons in the 3 arrays distances, categories and nids. The “vectNbr” vectors are all composed of the same number of components “vectLen”.

- vectors= int Array with a size vectNbr * vectLen
- distances= int Array of size vectNbr*K reporting for each vector the distances of the K top firing neurons or 0xFFFF if less than K neurons fire.
- categories= int Array of size vectNbr*K reporting for each vector the categories of the K top firing neurons or 0xFFFF if less than K neurons fire.
- nids= int Array of size vectNbr*K reporting for each vector the identifiers of the K top firing neurons or 0xFFFF if less than K neurons fire.

Considerations before calling this function:

- Change the Global Context Register to match the context represented by the input vectors

ADVANCED FUNCTIONS

Register Transfer Level functions

The functions supplied in the CogniPat SDK are all based on a Register Transfer Level (RTL) protocol. Depending on your hardware platform, the compatible CogniPat DLL is built on the use of the GV protocol in compliance with your platform's communication bus such as USB, SPI, I2C, RS485, etc. Note that for certain platforms, the RTL protocol can also be used to address components other than the neurons such as memory, on-board sensors, and more.

```
int Write(unsigned char module, byte reg, int data);
```

Write the value *data* to the register *reg* of the module *module*.
Example: Write(1, 6, 3), set the minimum influence field of the network to the value 3.

```
int Read(unsigned char module, byte reg);
```

Return the value *of* the register *reg* of the module *module*.
Example: Read(1, 15) returns the number of committed neurons

```
int Write_Addr(int addr, int length_inByte, byte data[]);
```

Write the values of the byte array *data* to the address *addr* of the hardware platform.

Remark: The internal communication protocol of the various NeuroMem platforms reads and writes integer values to the registers of the modules. This means that the data array is actually read two bytes at a time and each two bytes are assembled as the upper and lower bytes of a register value.

- data= byte array[length_inByte]
- length_inByte, must be an even number of bytes (see remark above)
- addr= 4 bytes value formatted as follows:*[module*, 0, 0, *reg*]

Warning: In the case of the NeuroMem module 1, the usage of this function only makes sense for the CM_COMP register when a vector is broadcasted to the neurons or the content of the neurons is restored. However, please note the following:

- If the vector to broadcast to the neurons has N components, use Write_Addr to submit only the first N-1 component. Indeed the last component has to be written to the CM_LCOMP register.
- The N-1 components have to be formatted to a 2*(N-1) data array such that every odd byte is equal to 0 and even byte is equal to a vector component.

Example:
Vector=1,2,3,4 ➔ data=0,1,0,2,0,3,0,4
Write(0x01000001, 4, data) writes the components 1,2,3 and 4 to the CM_COMP register.

```
int Read_Addr(int addr, int length_inByte, byte data[]);
```

Read the values at the address *addr* of the hardware platform and store them to the byte array *data*.

Remark: The internal communication protocol of the various NeuroMem platforms reads and writes integer values to the registers of the modules. This means that the data array is actually filled two bytes at a time and each two bytes are assembled as the upper and lower bytes of a register value.

- data= byte array[length_inByte]
- length_inByte, must be an even number of bytes (see remark above)
- addr= 4 bytes value formatted as follows:*[module*, 0, 0, *reg*]

Warning: In the case of the NeuroMem module 1, the usage of this function only makes sense for the CM_COMP register in order to read the memory of a neuron. However, please note the following:
- Reading the N components of a neuron requires that you size the data array to 2*(N-1) bytes. Every odd byte is equal to 0 and even byte is equal to a neuron component.

Example:
Vector=1,2,3,4 ➔ data=0,1,0,2,0,3,0,4
Write(0x01000001, 4, data) writes the components 1,2,3 and 4 to the CM_COMP register.

GV Protocol definition

The GV command protocol is a generic register transfer level protocol allowing access and control the components of a "NeuroMem Smart" board whether these components are hardware or firmware blocks. Observance of this protocol will ensure the compatibility and portability of your APIs and FPGA IP cores across "NeuroMem Smart" platforms.

The GV protocol is a 10-bytes control command, possibly followed by additional bytes of data as specified in the command.

- Byte0 DeviceID
- Byte 1-2-3-4 R/W bit + 31-bit address
- Byte 5-6-7 24-bit Data length
- Byte 8-9 Minimum first 2 bytes of data
- Byte + Remaining data (up to Data Length)

Descr	**DeviceID[7:0]**	**Address[31:0]**			**Data length[23:0]**	**Data**
Size	1 byte	1 byte		3 bytes	3 bytes	2 bytes * Data length
Write cmd	Device handle	Bit 31=1	Module[6:0]	Register[24:0]	Size of the input array expressed in words	Input array
Read cmd	Device handle	Bit 31=0	Module[6:0]	Register[24:0]	Size of the output array expressed in words	Output array

Examples:

Write the 16-bit value 0x33AA to the MAXIF (register 7) of the NeuroMem network (module 1)
0x00 81 00 00 07 00 00 01 33 AA

Write a vector of 4 consecutive byte values 9,8,7,6 to the COMP (register 1 of the NeuroMem network (module 1)
0x00 81 00 00 01 00 00 02 09 08 07 06

Read the 16-bit value of the MINIF (register 6) of the NeuroMem network (module 1)
0x00 01 00 00 06 00 00 01
Data is returned into 2 bytes or a word. Unless the MINIF register has been written, its default value is 2.

Read 8 consecutive byte values of the RESULTS (register 2) of the RECO_LOGIC (module 4)
0x00 04 00 00 02 00 00 04
Data is returned into 8 bytes.

Tentative Address Mapping Common to all platforms

Address[31] = Read/Write
Address[30-24]=Module[7:0]= identifier of the component
Address[23:16]= presently unused (default 0x0000)
Address[15-8]= presently unused (default 0x0000)
Address[7:0]=Register[7:0] = identifier of an 8-bit register

Address Range[30-0]	**Module= Address[30-24]**	**Description**
0x00000000 0x0000001F	0= Board info	Access to the settings of the board connected to the host including but not limited to a version number, a clock counter, enabling/disabling communication buses and I/O lines.
0x01000000 0x0100001F	1= NeuroMem network	Access to the neurons to learn and recognize vectors, save and restore knowledge.
0x02000000 0x02FFFFFF	2= Memory	Access to the SRAM address bank managing data transfer to/from host, from a sensor, to/from another module
0x03000030 0x03FFFFF	3= Flash memory	Access to the Flash memory to read and write pages of data, but also to read and decode sequences of instructions for the other modules.
0x04000000 0x04FFFFF	4= CogniPat engine	A pattern recognition engine with functions to learn and recognize vectors in single or batch mode. This module need access to the memory module and may be others to get its vector data inputs and stored or transmit its results.
0x05000000 0x05FFFFFF	5= CogniSight engine	An image recognition engine with functions to learn and recognize visual objects or events. This module need access to the memory module and may be others like a video sensor module to get its pixel inputs and stored or transmit its results.
0x06000000 0x060000FFF	6= Sensor module	Access to the registers of a sensor
Etc.	Etc.	Etc.

Cycle Accurate timings

The API features a cycle accurate counter to evaluate the latency of the operations executed by the NeuroMem network. Multiplying this counter by the duration of a clock cycle gives time spent by the entire network to execute a given function. Note that this does not include the data transfer time between the network and the host.

Please note that enabling the cycle accurate counter imposes some overhead operations on the basic RTL functions and will consequently slow down the execution of the functions if timed by the host processor.

```
void EnableClockCounter()
```

Turn ON the usage of the cycle accurate counter and reset the counter value.

```
void DisableClockCounter()
```

Turn OFF the usage of the cycle accurate counter.

```
int ReadClockCounter()
```

Read the current value of the counter.

```
void ClearClockCounter()
```

Clear the counter.

Example code using clock counter:

EnableClockCounter	
Read CM_DIST	18 cycles
Read CM_CAT	19 cycles if ID_ is high, 3 cycles otherwise
ReadClockCounter	returns (37)

EXAMPLE SCRIPT

The following script is intended to illustrate the behavior of the neurons during learning and recognition operations. It is supplied in a variety of programming language including C/C++. C#, MatLab, Python, Arduino and more.

Step 1: Learn

Learn vector1 [11,11,11,11] (1) with category 55
Learn vector2 [15,15,15,15] with category 33(2)
Leanr vector3 [20,20,20,20] with category 100

(1) The learned vectors are purposely set to arrays of constant values so their representation and relationship are easy to understand. The distance between two "flat" vectors is indeed the difference between their constant value times their number of components. For example the distance between [11,11,11,11] and [15,15,15,15] is equal to 4 * 4. This simple arithmetic makes it easy to understand the different cases of recognition illustrated in this test script.

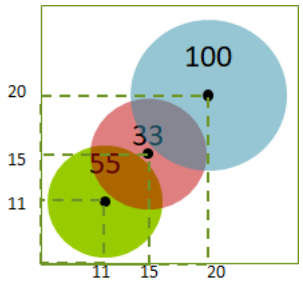
(2) The category of the second neuron is purposely set to a lesser value than the category of the first neuron to verify that if both neurons fire with a same distance, the category of the neuron on the 2nd chip is still the first the be read out

Fig1 is a representation of the decision space modeled by the 3 neurons where Neuron1 is shown in red, Neuron2 in green and Neuron3 in blue. In the follwoing 2D graph, we limit the length of the models to 2 components instead of 4, so they can be positioned in an (X,Y) plot. X=1st component and Y=Last component, and a surrounding diamond shape with the side equal to their influence field.

Committed neurons= 3

NeuronID=1	Components=11, 11, 11, 11,	AIF=16,	CAT=55
NeuronID=2	Components=15, 15, 15, 15,	AIF=16,	CAT=33
NeuronID=3	Components=20, 20, 20, 20,	AIF=20,	CAT=100

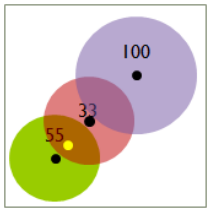
The influence fields of Neuron#0 and Neuron#1 overlap, as well as Neuron#1 and Neuron#2 overlap but differently since their distances from one another are different.



Step2: Recognition

The vectors submitted for recognition are selected purposely to illustrate cases of positive recognition with or without uncertainty, as well as cases of non recognition. The program reads the responses of all the firing neurons, which is until the distance register returns a value 0xFFFF or 65553.

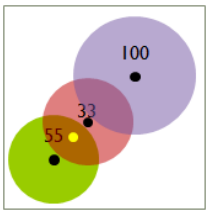
```
Case of uncertainty, closer to Neuron#1
```



Vector=12, 12, 12, 12
Neuron 1 and 2 fire. Vector is closer to Neuron1

Response#1	Distance= 4	Category= 55	NeuronID= 1
Response#2	Distance= 12	Category= 33	NeuronID= 2
Response#3	Distance= 65535	Category= 65535	NeuronID= 65535

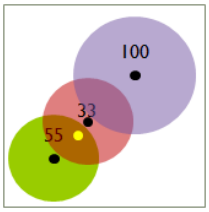
Case of uncertainty, equi-distant to Neuron#1 and Neuron#2



Vector=13, 13, 13, 13,
Neuron 1 and 2 fire. Vector is equi-distant to both neurons

Response#1	Distance= 8	Category= 33	NeuronID= 2
Response#2	Distance= 8	Category= 55	NeuronID= 1
Response#3	Distance= 65535	Category= 65535	NeuronID= 65535

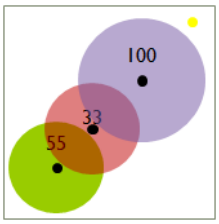
Case of uncertainty, closer to Neuron#2



Vector=14, 14, 14, 14,
Neuron 1 and 2 fire. Vector is closer to Neuron2

Response#1	Distance= 4	Category= 33	NeuronID= 2
Response#2	Distance= 12	Category= 55	NeuronID= 1
Response#3	Distance= 65535	Category= 65535	NeuronID= 65535

Case of unknown



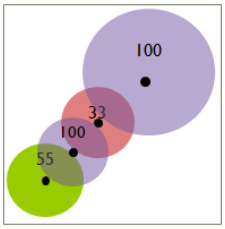
Vector=30, 30, 30, 30,
No neuron fire

Response#1	Distance= 65535	Category= 65535	NeuronID= 65535

Step3: Adaptive learning

Learning a new vector to illustrate the reduction of neurons'AIFs.

Learn vector[13,13,13,13] with category 100. This vector is equidistant to both Neuron1 and Neuron2. Learning it as a new category, will force Neuron1 and 2 to shrink from thei AIF=16 to an AIF=8 in order to make room for a new neuron which will hold the new model and its category.



Committed neurons= 4

NeuronID=1	Components=11, 11, 11, 11,	AIF=8,	CAT=55
NeuronID=2	Components=15, 15, 15, 15,	AIF=8	CAT=33
NeuronID=3	Components=20, 20, 20, 20,	AIF=20	CAT=100
NeuronID=4	Components=13, 13, 13, 13,	AIF=8,	CAT=100

Note that if the vector to learn was [12,12,12,12], Neuron1 would shrink to 4 and Neuron2 to 12.