NeuroMem API
Open Source Library

PATTERN LEARNING AND RECOGNITION WITH A NEUROMEM NETWORK

Version 6.3
Revised 05/03/2019

CONTENTS

INTRODUCTION

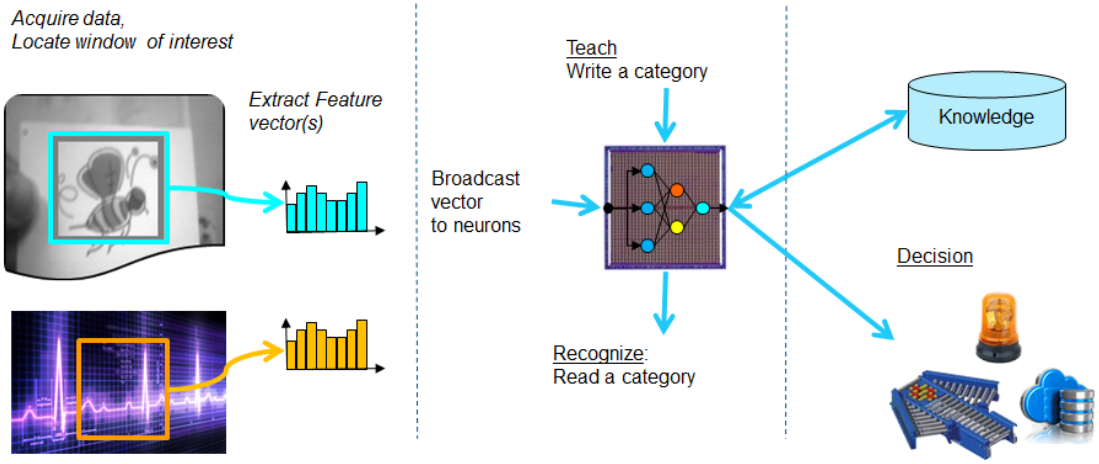
NeuroMem is a pattern recognition accelerator chip which is also trainable in real-time by learning examples. The interface to a chain of neurons of any size comes down to 4 simple main operations:

- 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 a 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.

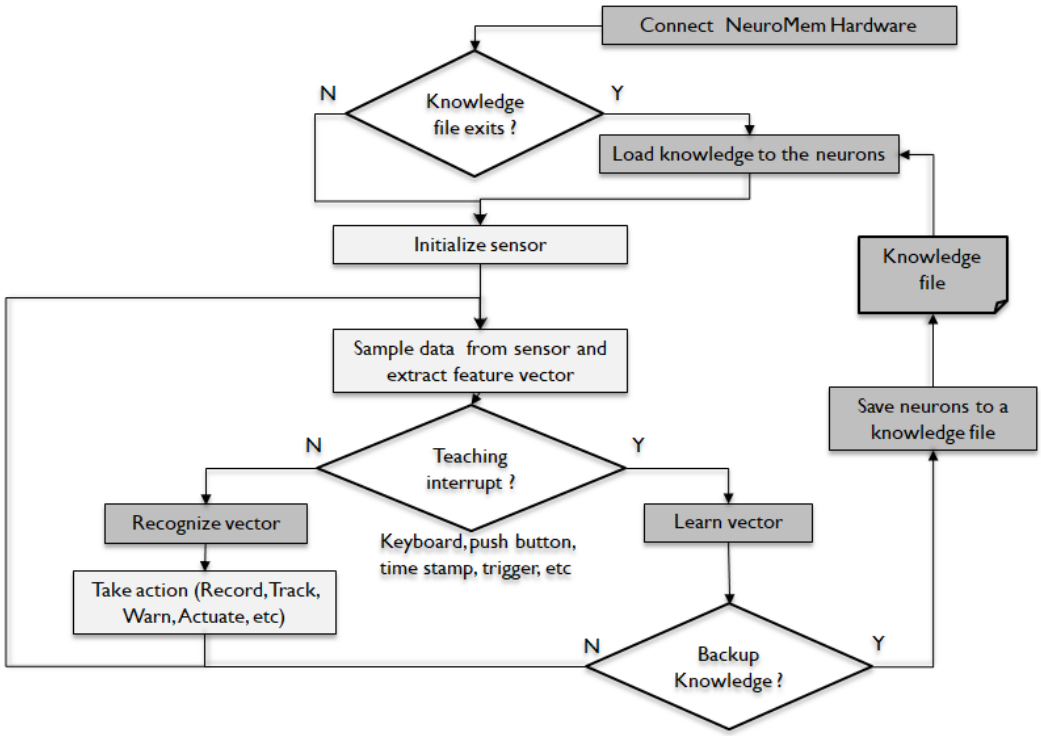


The functions supplied in the NeuroMem API are all based on a Register Transfer Level (RTL) protocol described as the NeuroMem Smart Protocol. Higher level functions manipulating agnostic or specific data types can be built on this protocol. Depending on your hardware platform, this RTL protocol can also be used to address other components such as memory, on-board sensors, and more.

API OVERVIEW

The NeuroMem API includes the basic functions to initialize the NeuroMem network, teach and recognize patterns, save, and restore the knowledge base built and stored in the neurons through the learning process.

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?

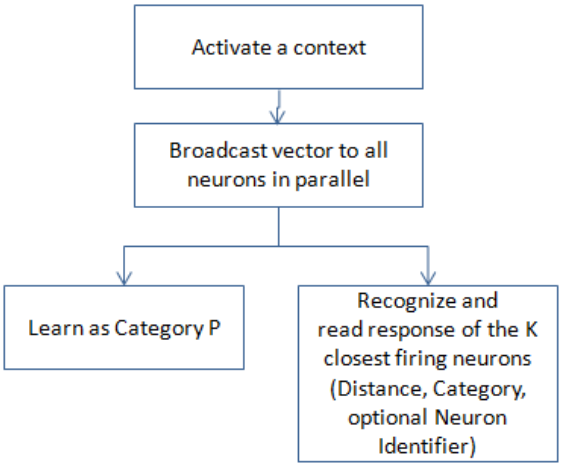
A feature vector represents a dimension (different from its length) which we associate to the notion of context. For example, a color histogram extracted from a pixel area may have the same length as a monochrome histogram, but its dimensionality is richer because it will contain information about the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



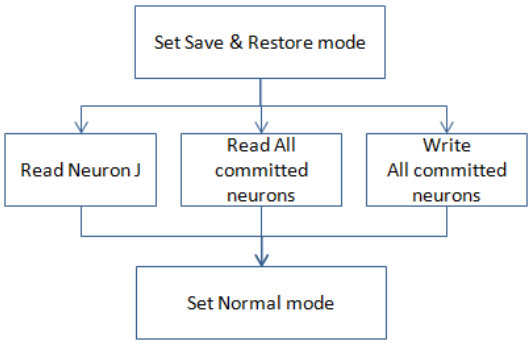
Our NeuroMem RBF tutorial is a simple application demonstrating how the neurons build a decision space autonomously by learning examples and how they recognize new vectors with ability to report cases of unknown and uncertainties too. For the sake of a simplicity, the decision space is limited to a 2D 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:

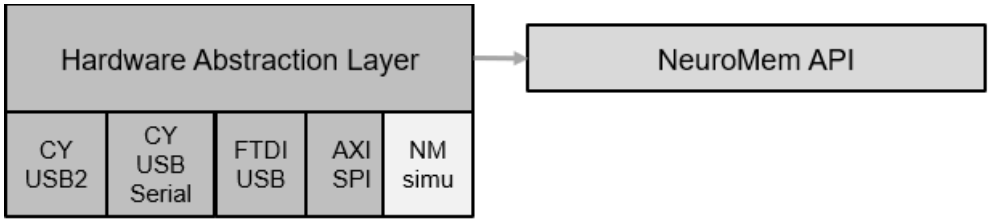


For more information of the NeuroMem technology, please refer to the NeuroMem Technology Reference Guide.

FILE AND FOLDERS

The NeuroMem API is intended for both Windows and Linux machines and can access a NeuroMem network on a variety of connected devices including but not limited to:

- NeuroMem simulation (run-time license)
- NeuroShield as USB device (CyPress USB Serial)
- NeuroShield interfaced to ZYNQ AXI_SPI
- NeuroShield interfaced to Python on Raspberry PI
- NeuroMem USB dongle or Brilliant or CogniSight Sensor (CyPress USB2)
- NeuroStack (FDTI USB)



The folder contains a lib folder and several app folders with examples written under specific IDE such as C++, C++ Visual Studio, Python, etc.

All examples have a dependency to the lib\neuromem folder which contains the core C++ function library accessing the neurons and described in this manual.

Depending on your NeuroMem xyz hardware, your project must link the neuromem.cpp file to one of the GVComm.cpp file stored in the Comm_xyz folders.

All platforms are tested running the SimpleScript described in the last chapter of this manual.

FUNCTION LIBRARY

InitializeNetwork()

Connect to the NeuroMem hardware platform establishing SPI or USB communication through the NeuroMem Smart Protocol, clears the entire content of the neurons (registers and memory) and return the neural network capacity.

Forget()
Forget(int Maxif)

Set all the neurons to idle mode. The first neuron of the chain is ready to learn. The global context, minimum and maximum influence fields are reset to their default values of 1, 2 and 0x4000.
Note that this function does not clear the memory of the neurons.

The Maximum Influence Field can be changed. The smaller Maxif, the more conservative the generalization capability of the neurons while learning. Maxif can range between 2 and 0xFFFF.

void clearNeurons();

Clears the memory of all the neurons and reset their context, minimum influence fields and categor

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

Learning consists of broadcasting a vector and assigning its category-to-learn. Depending on the application, the category-to-learn can be defined by the user in a supervised mode. It can also be assigned programmatically in an unsupervised mode. After the broadcast, the firing neurons with a category other than category-to-learn reduce their Influence Fields to exclude the input vector from their similarity domain. This is also when a new neuron is committed if no firing neuron has the category equal to category-to-learn.

Inputs

A feature vector is an array of integer values with a dimension less than or equal the memory capacity of the NeuroMem neurons. Read more>>

Category can range between 1 and 32767. Category of 0 is used to teach a counter example with the intend to correct neurons firing erroneoulsy, but without commiting a new neuron.

Outputs

The function returns the total number of committed neurons.

Considerations before calling this function:

- Forget to reset the network
- Forget and decrease the Max Influence Field to learn in a more conservative manner
- 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.

Int(nsr) Classify(int vector[], int length);
Int(nsr) Classify(int vector[], int length, int distance, int category, int nid);
Int(respNbr) Classify(int vector[], int length, int K, int distance[], int category[], int nid[]);

Three versions of the Classify function exist depending on the level of details required by your application and the speed constraints you have.

- Level 1: Recognition Status
    - Reports the status of the recognition including if it is positive or not, and in the former case if it is identified with certainty or not.
    - Takes 1 clock cycle after the broadcast of the vector
- Level 2: Best or Closest Match
    - Reports the response of the neuron with the closest match, if any, and the status of the recognition.
    - Takes 19 clock cycles after the broadcast of the vector
- Level 3: K closest matches
    - Reports the response of the K neurons with the closest matches sorted per increasing order of distances, if applicable.
    - Takes 19 clock cycles times K after the broadcast of the vector

Inputs

A feature vector is an array of integer values with a dimension less than or equal the memory capacity of the NeuroMem neurons.

Length is the dimension of the vector.

K indicates the maximum number of responses to read out, if applicable. K is a maximum if the network is set to RBF mode since there is no warranty that K neurons will fire. K is always applicable if the network is set to KNN mode.

Output Level 1: Recognition Status:

Nsr or 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 disagree with its category (UNCertain)

Output Level 2: Closest Match:

- Distance register of the firing neuron, or the neuron with the smallest distance to the input vector. 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.

- If the NSR indicates a case of uncertainty and you are interested to analyze it, you can immediately execute a series of Read(DIST) + Read(CAT) + Read(NID) to obtain the response of the next closest firing neurons without broadcasting the vector again, and this until you read DIST=0xFFFF which notifies that all firing neurons have been queried.

Output Level 3: K Closest Matches:

- 3 output arrays with a dimension K words
- Distances of up the K closest firing neurons or 0xFFFF if non-existant. It is calculated per the selected norm (Bit 7 of the Global Context Register).
- 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.
- Identifiers of up the K closest firing neurons or 0 if inexistant. NID can range between 1 and the number of neurons available in the network.
- The function returns the number of responses, RespNbr, which can be as high as the input value K. Note that if the network is set to KNN mode, RespNbr is always equal to K. This is not true in RBF mode, since the neurons only fire if the input vector falls in their influence field.

Considerations before calling this function:

- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier. This is done by setting bit 4 of the NSR to 1 (Write NSR, 32). Do not forget to set it back to 0 prior to the next learning operation so you can benefit from the RBF mode to detect and learn novelty.

void ReadNeuron(int nid, int* context, int model[], int* aif, int* category);
void ReadNeuron(int nid, int neuron[]);

Saving the content of a specific neuron consists of switching the network to Save-and-Restore mode, pointing to the specific neuron of the chain and reading sequentially its memory and registers.

Inputs

NID is the index of the neuron in the chain starting at 1. NID can range between 1 and the number of committed neurons. If greater, the content of the last committed neuron is returned.

Outputs

Neuron[] is an array of NEURONSIZE+4 integer as described below.
NEURONSIZE is the memory capacity of each neuron and returned by the getNeuronsInfo function (ex: 256 in the CM1K chip, 128 in the Intel® QuarkSE chip)

- Neuron[0] = NCR, Neuron Context Register
- Neuron [1-NEURONSIZE+1] = NEURONSIZE components of neuron memory
- Neuron[NEURONSIZE+2] = AIF, Active Influence Field
- Neuron[NEURONSIZE+3] = MINIF, Minimum Influence Field
- Neuron[NEURONSIZE+4] = CAT, Category

*Note that if a model stored in a neuron has a length L lesser than NEURONSIZE, only the first L values of the model[] are significant. The remaining values might be irrelevant or noisy unless the Init function was executed prior to starting the learning.

Int(count) ReadNeurons(int neurons[]);

The NeuroMem neurons can be saved for backup purposes (in between training sessions for example) or for export and use on other devices.

Saving the content of the neurons consists of switching the network to Save-and-Restore mode, pointing to the first neuron of the chain and reading sequentially their memory and registers until a neuron with a null category is reached.

Outputs

The Neurons output array is an array of ncount * Neuron[]

Neuron[] is an array of NEURONSIZE+4 integer as described below.
NEURONSIZE is the memory capacity of each neuron and returned by the getNeuronsInfo function (ex: 256 in the CM1K chip, 128 in the Intel® QuarkSE chip)

- Neuron[0] = NCR, Neuron Context Register
- Neuron [1-NEURONSIZE+1] = NEURONSIZE components of neuron memory
- Neuron[NEURONSIZE+2] = AIF, Active Influence Field
- Neuron[NEURONSIZE+3] = MINIF, Minimum Influence Field
- Neuron[NEURONSIZE+4] = CAT, Category

*Note that if a model stored in a neuron has a length L lesser than NEURONSIZE, only the first L values of the model[] are significant. The remaining values might be irrelevant or noisy unless the Init function was executed prior to starting the learning.

The function returns the number of committed neurons.

Int(ncount) WriteNeurons(int neurons[], int ncount);

The NeuroMem neurons can be restored from backup files or files exported from another hardware platform. This operation erases any previous knowledge built and residing in the neurons.

Loading the content of the neurons consists of switching the network to Save-and-Restore mode, pointing to the first neuron of the chain and writing sequentially to their memory and registers. Execution of this function erases any previous knowledge held by the neurons.

Inputs

The Neurons input array is an array of ncount * Neuron[]

Neuron[] is an array of NEURONSIZE+4 integer as described below.
NEURONSIZE is the memory capacity of each neuron and returned by the getNeuronsInfo function (ex: 256 in the CM1K chip, 128 in the Intel® QuarkSE chip)

- Neuron[0] = NCR, Neuron Context Register
- Neuron [1 : NEURONSIZE+1] = NEURONSIZE, size of each neuron memory
- Neuron[NEURONSIZE+2] = AIF, Active Influence Field
- Neuron[NEURONSIZE+3] = MINIF, Minimum Influence Field
- Neuron[NEURONSIZE+4] = CAT, Category

Outputs

The function returns the number of committed neurons once the input array has been loaded into the silicon neurons. This value is equal to 0xFFFF is the network is full meaning that the neurons array exceeds the network capacity.

Remark:

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. This function is similar to ReadNeurons but adds a header before the data documenting the file format and size. It returns the number of committed neurons saved to the file.

File header:

Header[0]= format version number;
Header[1] = Neuron Size;
Header[2] = Neuron Count;
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. This function is similar to WriteNeurons but first decodes a header documenting the file format and size. It returns the number of committed neurons loaded from the file.

File header:

Header[0]= format version number;
Header[1] = Neuron Size;
Header[2] = Neuron Count;
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

ADVANCED FUNCTIONS

The following functions should be used with cautious and full understanding of their behavior. For more details refer to the NeuroMem Technology Reference Guide.

void setContext(int context, int minif, int maxif);

Select the context and associated minimum and maximum influence fields for the next neurons to be committed.

Inputs

The Context value allows to assign the neurons dynamically to the modeling of decision space based on different feature extraction methods.
It is coded as follows in the Global Context Register (GCR) and Neuron Context Register (NCR):

- Bit [6-0] = context value, or a range of [0, 127]
- Bit 7 = Norm used to calculate the distance; 0=L1, 1=LSUP;
- The neurons with an NCR lesser than 128 calculate L1 distances, while the neurons with an NCR greater than 128 calculate LSup distances.
- The context value 0 is special and enables all committed neurons to react to an input pattern regardless of their individual context value.

The Minimum and Maximum Influence Fields are also global variables of the NeuroMem network. Their values should be set in relation to the dimension represented by the feature vectors and the distance calculation in use (L1 or LSup).

MAXIF:

- Range [2, 65534]; Default 16384
- The larger the MAXIF, the more generalization or over-fitting. Tendency to create lesser liberal neurons.
- The lesser the MAXIF, the less generalization. Tendency to create many conservative neurons.

MINIF:

- Range [2, MAXIF]; Default 2
- The larger the MINIF, the more uncertain in the decision space. Tendency to create degenerated neurons which overlap on one another. Can help flag outliers, reveal non discriminating features and more.

void getContext(int* context, int* minif, int* maxif);

Read the values of the current context, min and max influence fields.

void setRBF();

Set the NeuroMem network as a Radial Basis Function classifier. This is the default settings at power-up of the chip.

void setKNN();

Set the NeuroMem network as a K-Nearest Neighbor. This settings does not allow any learning and is a pure distance evaluation mode between stimuli and models.

SIMPLE SCRIPT

The following script is intended to illustrate the behavior of the neurons during learning and recognition operations. It is a reference script supplied in General Vision's SDK and APIs and can consequently be found in a variety of language including but not limited to:

- C/C++
- C#
- Python
- Arduino
- MatLab

Description

The vectors are set to simple and fixed data set such as [11,11,11,11] and [15,15,15,15], such that their relationship is easy to understand and represent graphically. However, in real applications, these vectors are extracted from real data and can be composed of up to 256 different values ranging between 0 and 256. For example, in the case of images, vectors can represent raw pixel values, grey-level, or color histograms, etc. In the case of audio and voice data, vectors can be cepstral coefficients, etc.

Step 1: Learn

Learn vector1 [11,11,11,11] (1) with category 55
Learn vector2 [15,15,15,15] with category 33(2)
Learn 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.

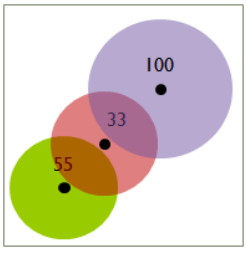
(2) The category of the second neuron is purposely set to a lesser value than the category of the first neuron to verify that if both neurons fire with a same distance, the category of the neuron on the 2nd chip is still the first the be read out

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 following 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.

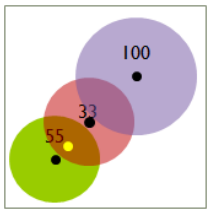


Step2: Recognition

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.

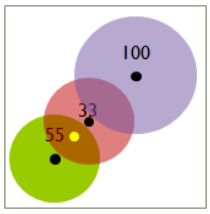
Case of uncertainty, closer to Neuron#1



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= 0xFFFF	Category= 0xFFFF	NeuronID= 0xFFFF

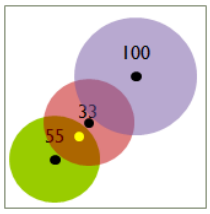
Case of uncertainty, equi-distant to Neuron#1 and Neuron#2



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= 0xFFFF	Category= 0xFFFF	NeuronID= 0xFFFF

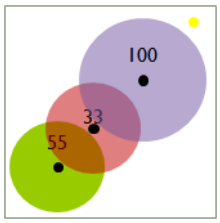
Case of uncertainty, closer to Neuron#2



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= 0xFFFF	Category= 0xFFFF	NeuronID= 0xFFFF

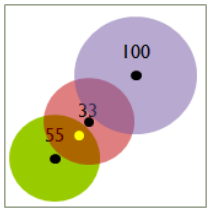
Case of unknown



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

Response#1	Distance= 0xFFFF	Category= 0xFFFF	NeuronID= 0xFFFF

Case of a KNN recognition



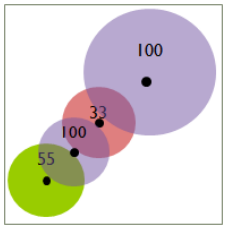
NSR=32
Vector=14, 14, 14, 14,
All neurons fire

Response#1	Distance= 4	Category= 33	NeuronID= 2
Response#2	Distance= 12	Category= 55	NeuronID= 1
Response#3	Distance= 32	Category= 100	NeuronID= 3

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 the AIF=16 to an AIF=8 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.