CogniPat Toolkit

Pattern recognition development kit for CogniMem networks

User's Manual

Version 1.0.0

Revised 08/03/2012

COGNIMEM

Technologies, Inc.

This License Agreement is a legal agreement between you (Licensee) and CogniMem Technologies Inc. (LICENSOR) for the enclosed computer software, and any associated media, printed materials, and "online" or electronic documentation ("Product").

License, rights and limitations

Licensor grants to the Licensee a revocable, non-transferable, and non-exclusive license to use the Product. This license may not be shared or used concurrently on different computers. You may install one copy of the Product, or any prior version, on a single computer for use by a single user. Licensee may make one copy of Product for backup purposes. No other copies shall be made without Licensor 's prior written consent.

Ownership and Copyright

The Product, including all supporting documentation, is a proprietary product of Licensor. Licensor retains all title and copyrights in and to the Product, the accompanying printed materials or electronic documentation and any complete or partial copies of the Product.

Limited Warranty

Licensor warrants that the Product will perform substantially in accordance with the accompanying written materials or electronic documentation and Licensor will make reasonable efforts to solve any problem issues arising out of the intended use of the Product. To the extent allowed by applicable law, implied warranties on the Product, if any, are limited to ninety (90) days. Licensor disclaim all other warranties and conditions, either express or implied, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, with regards to the Product.

Limitation of Liability

To the maximum extent permitted by law, in no event shall Licensor be liable for any special, incidental, indirect, or consequential damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use the Product or the provision of or failure to provide support services, even if Licensor has been advised of the possibility of such damages. In any case, Licensor' entire liability under any provision of the license agreement shall be limited to the amount actually paid by you for the Product.

Terms and termination

This license agreement is effective from the date of receipt and shall remain in full force until terminated. Without prejudice to any other right, Licensor may terminate this LICENSE AGREEMENT if you fail to comply with any of the terms and conditions of this license agreement. In such event, you must destroy all complete and partial copies of the Product in your possession.

Distribution

The distribution of an application written with the Product and not interfacing with any CogniMem hardware (chip, module, board or system) requires the purchase of one run-time license per system. The distribution of an application including the simulation of the CogniMem neurons requires a distribution license from CogniMem Technologies.

Contact Information

www.cognimem.com

1 Table of Contents

2 Introduction

Using your preferred programming environment, take advantage of a CogniMem hardware to match and classify patterns at high speed. Match or classify patterns in a constant 1.3 microseconds per pattern. Learning of your reference patterns can be made by writing them to the CogniMem chips as to standard memory cells or by using the CogniMem built-in model generator which automatically discards duplicates and decides if similar patterns represent novelty or not.

The CogniPat SDK interfaces seamlessly with the CogniMem evaluation boards featuring a single CM1K chip (i.e. 1024 neurons), 4 CM1K chips (i.e. 4096 neurons), and even stacks of chips with virtually unlimited capacity. In the event that your evaluation platform has an insufficient capacity to hold the reference patterns, a simulation of the cognitive memories or neurons can be instantiated and sized accordingly. In such event, speed performance is reported theoretically though a cycle accurate clock counter.

What can I do with the CogniPat SDK?

- Input patterns can derive from any type of discrete feature vectors and well as digital sensor outputs such as vibration, sound, signal, image, video, etc
- Build a knowledge base through supervised and unsupervised learning of reference patterns
- Exact or fuzzy matching of patterns against a knowledge base
    - Exact matching, K Nearest Neighbors
    - Fuzzy matching, anomaly detection, novelty detection
    - Search, clustering, de-duplication, modeling
- Evaluate the size, throughput and accuracy of a knowledge
- Save and Restore of a knowledge base
- Single and batch operations

3 SDK Contents

3.1 Files and Folders

- Bin folder
  - DLLs compiled for a variety of hardware platforms featuring a single CM1K chip or a chain of multiple CM1K chips, including one DLL simulating a user-defined capacity of chips.
- Examples folder
  - Test_CPlusPlus
  - Test_CSharp
  - Test_MatLab
- Doc folder
  - TM_CogniPat_SDK.pdf, this technical manual
  - TM_CM1K_Hardware_Manual
  - TM_CogniMem_Reference_Guide
  - TM_CogniMem_Decision_Space_Mapping
- Demos
  - TBD

4 Definitions

4.1 Vector byte array

A vector is a sequence of bytes with a length between 1 and 256. This sequence makes a pattern and can derive from any type of data source such as measurements, signal, sound, images, videos, etc. The pattern can be composed of raw data or represent a signature extracted from raw data and also called a feature.

Each byte is called a Component and the neurons of the CogniMem chip receive each component simultaneously, calculate their distance to the same indexed component in their memory and add it to their internal distance register.

4.2 Length

Number of components of the input vector broadcasted to the neurons

4.3 K

K represents a number of matches to retrieve from the cognitive memories or neurons after the broadcast of a pattern. One of the key feature of a CogniMem network is that the matches (whether exact or fuzzy) are automatically read out per decreasing order of confidence (i.e. increasing order of distance).

If the CogniMem network is set to operate in KNN mode (NSR register bit 5=1), it can be considered as a bank of cognitive memories with a built-in distance calculator. If the CogniMem network is set to operate in RBF mode, it can be considered as a neural network with built-in learning and recognition logic and the ability to model decision spaces with areas of unknown and/or uncertainty. It is possible that in RBF less than K neurons identify a pattern, in which case K represents a maximum number of readout.

4.4 Recognition Status code

The recognition status code is updated after each broadcast of a pattern to a CogniMem network.

- 0, unknown,
- 4, multiple neurons fire and report different categories,
- 8, a single or multiple neurons fire and report the same category

If the CogniMem network is set to operate in KNN mode (NSR register bit 5=1) the notion of “unknown” does not exist, so Status=0 never occurs.

4.5 Neuron content byte array

The entire content of a neuron is stored in a 264-byte array as follow:

- Byte 0= 0 (discarded)
- Byte 1= NCR, Neuron Context Register
- Byte 2-258= 256 components of the pattern vector*
- Byte 259-260: AIF, Active Influence Field
- Byte 261-262: MINIF, Minimum Influence Field
- Byte 263-264: CAT, Category

5 Initialization functions

5.1 int Version();

The last digit of the version number represents the platform as follows:
1= Simulation
2= CogniBlox
3= V1KU
4= CogniStix

Example: Version 12 means version 1 of the CogniPat DLL for the CogniBlox board.

5.2 int Connect(int DeviceID);

Establishes communication with the CogniMem device. This function returns 0 if the connection is successful.

In the case of a USB device, DeviceID is the USB device number connected to your host. Default is 0.

In the case of the Simulation platform, DeviceID is the number of neurons to simulate. If DeviceID is less than 1024, the function automatically instantiates a chain of 1024 neurons which is the capacity of a single CM1K chip. Make sure to size the neural network to the least requested capacity in order to optimize the recognition speed.
Example: Connect(10,000) will instantiate a chain of 10,000 software neurons. Recognition time might turn slow since the simulation features sequential neurons as opposed to the parallel neurons in a chain of CM1K chips.

5.3 int Disconnect();

Closes the communication with the current device.

6 Learning and Recognition functions

6.1 int Status = Broadcast(byte vector[], int length)

This function sends the length components of a vector to the neurons and returns the status of the recognition.

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, or to turn ON/OFF the Save-and-Restore mode.
- Change the Min Influence Field and Max Influence Field

6.2 int Learn(byte vector[], int length, int category)

Learn the input *Vector* composed of *Length* components as belonging to the category *Category*.

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, or to turn ON/OFF the Save-and-Restore mode.
- Change the Min Influence Field and Max Influence Field

6.3 int Recognize(byte vector[], int length, int K, int distance[], int category[], int Nid[])

Recognize the input *Vector* composed of *Length* components and read the response of the K first firing neurons reporting their *Distance*, *Category* and *Identifier* register. Thanks to the CogniMem technology the response of the firing neurons are automatically read in order of decreasing confidence.

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, or to turn ON/OFF the Save-and-Restore mode.

6.4 int CommittedNeurons();

Report the number of committed neurons.

Remark: Calling this function when the network is full might take a few second as explained below. Indeed the NCOUNT register is equal to 0xFFFF when the network is full and no longer reports the number of committed neurons. The function then switches the network to Save and Restore mode, points at the first neuron of the chain and reads their category sequentially until a category 0 or 0xFFFF is reached. The number of iterations is equal to the number of committed neurons. If your platform features 4096 neurons or more, this will not be instantaneous!

6.5 void ClearNeurons();

Clear the contents of all the neurons including the neuron register and memory cells.

6.6 void ReadNeuron(int NeuronID, byte neuron[]);

Output the entire content of a neuron in a 264-byte array (refer to the first chapter for a description of the Neuron byte array).

*Note that if an application is dealing with patterns of a length L lesser than 256, only the first L values of the byte array[2-258] will be significant. If the ClearNeurons function was executed prior to the teaching of the neurons, 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.

6.7 int ReadNeurons(byte *neurons, int ncount);

Output the content of the first *ncount* neurons in a byte array. The function returns a value N equal to the input *ncount* or the number of committed neurons whichever is smaller.

neurons= Byte array of (N*264) with the 264 elements describing the content of a neuron (refer to the first chapter for a description of the Neuron byte array).

*Note that if an application is dealing with patterns of a length L lesser than 256, only the first L values of the byte array[2-258] will be significant. If the ClearNeurons function was executed prior to the teaching of the neurons, 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.

6.8 int WriteNeurons(byte *neurons, int ncount);

Load the content of the *neurons* byte array into the first *ncount* neurons of the chain. The function returns a value N equal to the input *ncount* or the number of committed neurons at the end of the operation whichever is smaller.

neurons= Byte array of (ncount*264) with the 264 elements describing the content of a neuron (refer to the first chapter for a description of the Neuron byte array).

6.9 int SaveNeurons(char *Filename);

Save the contents of all the neurons to a file.

6.10 int LoadNeurons(char *Filename);

Load the contents of all the neurons from a file.

7 Vector Batch Processing

7.1 int LearnVectors(byte *vectors, int vectNbr, int vectLen, int *categories);

Learn vectors with the associated categories. This function does not clear the existing knowledge. It returns the total number of committed neurons.

- vectors= Byte Array[vectNbr,vectLen]
  - vecNbr, number of vectors to learn. Must be less than or equal to the number of non committed neurons in the network.
  - vectLen, length of the vectors to learn. Must be less than or equal to 256.
- categories= Int Array[vectNbr]
  - If the category[i] == 0xFFFF, the vector I is learned as belonging to the last taught category+1
  - If at least one category !=0xFFFF and NSR=0, the learning is automatically iterative. Refer to the CogniMem Reference Guide for information about the advantages of an Iterative Learning method.

Considerations before calling this function:
- Clear the neurons
- 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.

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

Recognize the vectors and store the response of the first K firing neurons in the 3 arrays *distances*, *categories* and *nids*.

- vectors= Byte Array[vectNbr,vectLen]
  - vecNbr, number of vectors to learn. Must be less than or equal to the number of non committed neurons in the network.
  - vectLen, length of the vectors to learn. Must be less than or equal to 256.
- distances= Int Array[K]
- categories= Int Array[K]
- nids= Int Array[K]

Considerations before calling this function:
- Set the NSR register to 16 or 0
  - If NSR=32, the vectors are recognized using the KNN classifier
  - If NSR=0, the vectors are recognized using the RBF classifier
- Change the Global Context Register

7.3 int MatchVectors(byte *vectorsRef, int vectRefNbr, byte *vectors, int vectNbr, int vectLen, int maxDistance, int *indexPairs, int *matchDistances);

Match two sets of vectors of the same length and produces the pairs of matching indexes and their distances. This fnction clears the neurons prior to execution. It returns the number of pairs found.

- vectorsRef= Byte Array[vectRefNbr,vectLen]
  - vectors to learn or load into the neurons
- vectors= Byte Array[vectNbr,vectLen]
  - vectors to match against the *vectorsRef*
- maxDistance
  - distance not to exceed to consider a match between a *vector* and a *vectorRef*
- indexPairs= Array[matches_found,2]
  - the first value of the pair is the index of *vector*
  - the second value of the pair is the index of the closest *vectorRef*
- matchDistances[matches_found]
  - distance between the two vectors of an indexPairs

Considerations before calling this function:

- Clear the neurons
- 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.

8 Network Register Access Level

On all CogniMem evaluation boards, a chain of CM1Ks is referred to as the module number 1. Depending on the platform, additional modules may be accessible such as a bank of memory, a sensor, etc. More to come on this subject.

8.1 The CogniMem registers

A CogniMem neural network is fully controlled through the set of 15 registers listed below.

CM_NCR	0x00	Neuron Context register
CM_COMP	0x01	Neuron Component (or memory index)
CM_LCOMP	0x02	Neuron Last Component
CM_DIST	0x03	Neuron Distance register
CM_INDEXCOMP	0x03	Neuron Component Index
CM_CAT	0x04	Neuron Category
CM_AIF	0x05	Neuron Active Influence Field
CM_MINIF	0x06	Neuron Minimum Influence Field
CM_MAXIF	0x07	Global Maximum Influence Field
CM_NID	0x0A	Neuron Identifier
CM_GCR	0x0B	Network Global Context register
CM_RESETCHAIN	0x0C	Network Reset Chain (Write only)
CM_NSR	0x0D	Network Status register
CM_NCOUNT	0x0F	Network committed neurons (read only)
CM_FORGET	0x0F	Network Clear

For information about the neuron registers, refer to the CM1K Hardware Manual and CogniMem Reference Guide.

8.2 int Write(byte 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.

8.3 int Read(byte module, byte reg)

Return the value *of* the register *reg* of the module *module*.

Example: Read(1, 15) returns the number of committed neurons

8.4 int Readout(byte module, byte reg, int *data)

Read the value *of* the register *reg* of the module *module*.
Example: Readout(1, 15, ncount) write the number of committed neurons to the variable ncount.

8.5 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 CogniMem 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 CogniMem 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.

8.6 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 CogniMem 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 CogniMem 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.

9 Cycle Accurate timings

In the event that your evaluation platform has an insufficient capacity to hold the reference patterns, the CogniPat SDK let you instantiate a simulation of the cognitive memories or neurons of a user-defined capacity. In such event, speed performance will greatly deteriorate if more than hundreds of neurons are committed.

The SDK provides for a cycle-accurate report so you can estimate the number of clock cycles executed by the chain of CM1K chips. For details about the clock cycles of each instruction you can refer to the CM1K hardware manual.

9.1 void EnableClockCounter()

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

9.2 void DisableClockCounter()

Turn OFF the usage of the cycle accurate counter.

9.3 int ReadClockCounter()

Read the current value of the counter.

9.4 void ClearClockCounter()

Clear the counter.

9.5 Example:

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