# INTRODUCTION

## Research Outline

### Neural Network or Biologically Inspired Modeling

By definition, any system that tries to model the architectural details of the neocortex is a biologically inspired model or neural network [54][55]. Computers cannot accomplish human-like performance for many tasks such as visual pattern recognition, understanding spoken language, recognizing and manipulating objects by touch, and navigating in a complex world. After decades of research, there exist no significant viable algorithms to achieve human-like performance on a computer or special hardware accelerator. So far, there has been not much research and development in hardware for the biologically inspired software models. The hardware implementation of large-scale neural networks is an excellent candidate application for the high density computation and storage possible with current and emerging semiconductor technologies [84]. Besides, hardware implementation is much faster than software, the primary motivation for this dissertation research is to engineer a system level design in hardware that can be used for many biologically inspired computation and other similar applications.

### Associative Memory

An associative memory (AM) [50] can recall information from incomplete or noisy inputs and as such, AM has applications in pattern recognition, facial recognition, robot vision, robot motion, DSP, voice recognition, and big data analysis. Research on the potential mapping of the AMs onto the nano-scale electronics provides useful insight into the development of non-von-Neumann neuromorphic architectures. A datapath for implementing an AM can be implemented using common hardware elements, such as, adder, multiplier, simple divider, sorter, comparator and counter.

Therefore, providing a methodology for non-von-Neuman architecture with nanoscale circuits and devices is one of the targets of this research.

### Massively Parallel Architecture

Neural network based algorithms generally require massive parallelism. Single Instruction Multiple Data (SIMD) [95], pipelining, and systolic array architecture [95] are typical to DSP, neural network and image processing algorithms.

The goal of this research is to propose a design methodology for a complete system that can handle large number of wide vectors with a series of SIMD (Single Instruction Multiple Data) type processing elements and pipelined architecture.

### Neuromorphic Circuits and Devices

The emergence of many novel nanotechnologies has been primarily driven by the expected scaling limits in conventional CMOS processes. Through such efforts many new and interesting novel neuromorphic circuits and devices have been discovered and invented. Memristor is an example of such a new technology.

A memristor feature size of F = 50 nm (where, F is the lithographic feature size or half-pitch i.e. half of center-to-center nanowire distance) yields a synaptic density of 1010 memristive synapses per square centimeter, which is comparable to that of the human cortex [89][90]. Therefore, memristor technology shows the prospect of scaling up the capacities of DSP and Image Processing architectures, and associative memories. Hybrid CMOS-Memristor design could be used for architectures which due to their complexity cannot be designed and simulated in real-time in hardware-software using conventional CMOS based design.

As such, this research undertakes the implementation of a complete system level design using binary memristors with IMPLY logic and using a new variant of a CMOL crossbar nano-grid array.

### Design Methodology Development

The essence of this dissertation work is to develop a new methodology to design a massively parallel and pipelined architecture at a system level using binary memristors for biologically inspired Associative Memory and other similar application areas as mentioned before. The research proposed here will involve the design of an IMPLY-memristor based massively parallel reconfigurable architecture at a system and logic levels.

## Research Background and Motivation

### Part 1: Research Groundwork

#### Defining Associative Memory

Associative memory (AM) [53][62] is a system that stores mappings from input representations to output representations. When the input pattern is given, the output pattern can be reliably retrieved. When the input is incomplete or noisy, the AM is still able to return the output result corresponding to the original input based on a *Best Match* procedure where the memory selects the input vector with the closest match, assuming some metric, to the given input, then returns the output vector for this closest matched input vector.

In *Best Match* associative memory, vector retrieval is done by matching the contents of each location to a key. This key could represent a subset or a corrupted version of the desired vector. The memory then returns the vector that is *closest* to the key. Here, *closest* is based on some metric, such as ***Euclidean Distance*** [19][36][37][38][39][40][41][42][43][44][45]. Likewise, the metric can be conditioned so that some vectors are more likely than others, leading to Bayesian-like inference.

As in associative memories (AM) the information is retrieved through a search: given an input vector one wants to obtain the stored vector that has been previously associated with the input. In a parallel hardware implementation of a large-scale associative memory the memory is searched to find the minimum distance between the new vector and the stored memory vector using the Euclidean distance formula.

On the other hand, the *Exact Match* association, as in the traditional content addressable memory (CAM), returns the stored value that corresponding to the exactly matched input. A CAM holds a list of vectors which are distinguished by their addresses, when a particular vector is needed, the exact address of the vector must be provided.

#### History of Associative Memory Algorithm Development

Associative memories can be of different types. The first associative memory model called *Die Lernmatrix* was introduced by Steinbuch and Piske in 1963. Willshaw model and modified versions (1969-1999) [53], Palm model (1980) [73], and iterative Palm model (1997), Brain-state-in-a-box (BSB) by Anderson et al. (1977, 1993, 1995, 2007), Hopfield network model (1982) [64], Self-Organizing Map (SOM) proposed by Kohonen (1982, 1989) [76][82], Dynamical Associative Memory (DAM) by Amari (1988, 1989), Bidirectional Associative Memory (BAM) by Kosko (1988) [68], Sparse Distributed Memory (SDM) by Kanerva (1988) [56][65][66], Bayesian Confidence propagation Neural Network (BCPNN) by Lansner et al. (1989) [75], Cortronic networks by Hecht-Nielsen (1999), Correlation matrix memories (CMM) [77], Furber model (2007) implemented using Spiking Neural Network (SNN) [51][52][60][72], Spatial and Temporal versions of Self-Organizing Incremental neural Network (SOINN, ESOINN, GAM) by Shen and Hasegawa (2006-2011) [57][79][80], Cortical Learning Algorithms (CLA) by Numenta (2010) are examples of some associative memories [61].

#### System Input Data Encoding

The input data into a neural network system can be received in any form, e.g. binary data, real-valued data etc. Input data then gets encoded as wide vectors. Different neural network models follow different encoding mechanism.

After generating the vectors, the similarity between the vectors is measured using the Euclidean distance calculation or calculating the dot product of the two vectors. The similarity is measured through a *distance threshold* value. A distance threshold constant is used to control the classification of a new node to a new class or to an existing class. During the experimentation, the values of distance threshold are changed several times. A small value of distance threshold may result in a large number of classes. With further experimentation, it is possible to obtain even fewer classes at the output by iterating on the distance threshold constant.

#### Evaluation of Associative Memory Algorithm

The purpose of the research was to provide hardware directions for biologically inspired associative memory models. Many groups have developed biologically inspired software based algorithms [61][79][80] to address such problems. A few groups are looking into creating increasingly more sophisticated hardware based models of neural circuits [63][67][87][88][89][93][96], and then apply those models to real applications.

Finding a suitable associative memory algorithm was the initial task for this dissertation work. Through a detailed literature search, some of the most promising models were identified. First, the performance of the associative memory model was evaluated. Next, the capability of sequential or temporal pattern prediction was checked. Based on all the results published by other authors and my own experimentation with software models, one suitable model was identified for this research.

As a part of this dissertation work, the Kanerva (Furber) SDM Associative memory model and the Palm Associative memory models were implemented by me in Matlab. After evaluation of the two models using the same datasets [69], it was not possible to prove the superiority of one model over the other, as both of these models showed some capability as well as some inaccuracies. The CLA Model [61] used for these experiments is a commercial model by Numenta, Inc. 2010. The Furber Model is a model that was coded in Matlab as a part of my dissertation research, and the coding required certain assumptions based on the original published work by Furber et al. [51][52]. In addition, although the CLA model has the *variable order sequence prediction* feature [61], the experimental results did not show performance superiority of the CLA model over the Furber Model. As such, we were unable to justify that the CLA model is performing any better than the Furber model and I rather concluded that both models have similar performance and none of the models are completely error free.

These conclusions were the motivation behind an additional literature search to find more models that can provide better solutions to the problem. I found a more promising biologically inspired associative memory model for spatial and temporal pattern recognition by Shen and Hasegawa [57][79][80] through further literature search. This led to the study of their SOINN model, ESOINN Model and finally GAM model, which is the most promising algorithms among all of the models studied. For the purpose of this research, the SOINN [57], and ESOINN algorithm [79] were coded in Matlab for *spatial pattern recognition*. Later, the complete GAM [80] algorithm was also coded by me [Appendix A], which algorithm does both spatial and temporal pattern recognition. For the *spatial pattern recognition* experiments, input data was collected from Lecun’s MNIST hand-written digit database [70] both for training and test purposes. The input data was further processed for the purpose of my dissertation research. Upon completion of the training, a different set of images were used to test the performance of the algorithm.

####  ESOINN/ GAM

Shen and Hasegawa proposed several models on pattern recognition, such as the Self-Organizing Incremental Neural Network (SOINN) [57] based on an unsupervised learning technique [58], and the Enhanced Self-Organizing Incremental Neural Network (ESOINN) [79], which is a modification of SOINN. Both of these algorithms have applications in spatial pattern recognition. Shen and Hasegawa also published a General Associative Memory (GAM) algorithm [80], which is an associative memory based algorithm, and a temporal version of the SOINN algorithm. The GAM model is constructed as a three-layer network structure. The input layer inputs key vectors, response vectors, and the associative relation between vectors. The memory layer stores input vectors incrementally to corresponding classes. The associative layer builds associative relations between classes. The method can incrementally learn key vectors and response vectors; store and recall both static information and temporal sequence information; and recall information from incomplete or noise-polluted inputs. Using the GAM model, Shen and Hasegawa demonstrated promising results of pattern recognition experiments using binary data, real-value data, and temporal sequences.

##### **GAM Architecture**

The input layer accepts any data which is encoded as a sparse distributed vector. These input vectors are called *key* and *response* vectors. The input layer receives the key vectors and response vectors. *Response vectors* are the outputs of the *key vectors*. The memory layer classifies the vectors into separate classes based on the similarity of the vectors falling within a threshold limit. The similarity between two vectors is measured through a distance calculation using normalized ***Euclidean distance***. The memory layer stores the input vectors incrementally to the corresponding classes as it receives the input vectors. If the input vector does not belong to an existing class in the memory layer, the GAM builds a new subnetwork in the memory layer to represent the new class. The GAM sends the class labels of subnetworks in the memory layer to the associative layer, and the associative layer builds relationships between the class of the key vector (the key class) and the class of the response vector (the response class) by using arrow edges. One node exists in the associative layer corresponding to one subnetwork in the memory layer. The arrow edges connecting these nodes represent the associative relationships between the classes. The beginning of an arrow edge indicates the key class; and the end of the arrow edge indicates the corresponding response class. The associative layer builds associative relationships among the classes. The GAM can store and recall binary or non-binary information, learn key vectors and response vectors incrementally, realize many-to-many associations with no predefined conditions, store and recall both static and temporal sequence information, and recall information from incomplete or noise-polluted inputs. Experiments using binary data, real-value data, and temporal sequences show that GAM is an efficient system.

GAM at first realizes auto-association, and then hetero-association as humans initially recognize or recall a class with a garbled or incomplete key vector, and then associate it with other classes according to the recognition of the key vector. A pattern recognition or pattern completion process uses auto-associative information and association between classes uses hetero-associative information.

##### **GAM Analysis**

The complete General Associative Memory (GAM) algorithm was analyzed by me as a baseline algorithm for this dissertation research. I observed that the GAM algorithm has an advantage as its datapath can be designed using the SIMD concepts. Also this algorithm fits well for a hybrid system level design as the control logic of the algorithm can be designed in CMOS, while the datapath and memory operations can be designed with a nanotechnology.

Since the goal of this dissertation was set early to develop a methodology for hardware design, we realized that there is no need to design the complete GAM system. We rather identified one most common and critical component that is widely used in GAM and many other similar associative memory architectures. Thus Euclidean Distance Calculator was identified for this methodology development work. Also, the reason the example of the Euclidean Distance calculator was used for this research is that it is widely applied by many Neural Network and similar algorithms in software. However, there is no hardware implementation available or even published. Moreover, for the application areas of pattern recognition, facial recognition, robot vision, Digital Signal Processing, voice recognition, big data analysis, and database mining, all of those algorithms require to process massively parallel large number of wide-word input vector/data and therefore, we need a hardware system that can handle those large number of wide input vectors or neurons efficiently. Conventional CMOS technology is not enough for handling any such massively parallel applications, and as a result, this dissertation proposes an alternate, memristor-based nanotechnology using stateful IMPLY based FPGA design, MsFPGA (Memristive stateful logic Field Programmable Gate Array).

This proposed MsFPGA is the new idea and development by itself, only motivated by the previous research on associative memories. It can be used for many other applications, the same way as CMOS-based FPGA architectures are being used now. However, to illustrate the proposed new device, we use the Euclidean Distance calculator, which can be applied as an important component in any of the above application areas listed. Besides, in this dissertation several potential applications of the proposed FPGA architecture and its associated design methodology are mentioned, such as pipelined and SIMD architectures, which are typical for neural network, machine learning, robot vision, and control related applications.

REFERENCES

1. Chua, L. O. (1971). Memristor-the missing circuit element. *Circuit Theory, IEEE Transactions on*, *18*(5), 507-519.
2. Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). The missing memristor found. *Nature*, *453*(7191), 80-83.
3. Borghetti, J., Snider, G. S., Kuekes, P. J., Yang, J. J., Stewart, D. R., & Williams, R. S. (2010). ‘Memristive’ switches enable ‘stateful’ logic operations via material implication. *Nature*, *464*(7290), 873-876.
4. Kuekes, P. (2008, November). Material implication: digital logic with memristors. In *Memristor and memristive systems symposium* (Vol. 21).
5. Kvatinsky, S., Satat, G., Wald, N., Friedman, E. G., Kolodny, A., & Weiser, U. C. (2014). Memristor-based material implication (IMPLY) logic: design principles and methodologies. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, *22*(10), 2054-2066.
6. Strukov, D. B., & Likharev, K. K. (2005). CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, *16*(6), 888.
7. Lehtonen, E., Tissari, J., Poikonen, J., Laiho, M., & Koskinen, L. (2014). A cellular computing architecture for parallel memristive stateful logic. *Microelectronics Journal*, *45*(11), 1438-1449.
8. Likharev, K. K., & Strukov, D. B. (2005). CMOL: Devices, circuits, and architectures. In *Introducing Molecular Electronics* (pp. 447-477). Springer Berlin Heidelberg.
9. Kim, K., Shin, S., & Kang, S. (2011). Field programmable stateful logic array. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *30*(12), 1800-1813.
10. Kim, K., Shin, S., & Kang, S. (2011, May). Stateful logic pipeline architecture. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on* (pp. 2497-2500). IEEE.
11. Kvatinsky, S., Kolodny, A., Weiser, U. C., & Friedman, E. G. (2011, October). Memristor-based IMPLY logic design procedure. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on* (pp. 142-147). IEEE.
12. Lehtonen, E., & Laiho, M. (2009, July). Stateful implication logic with memristors. In *Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures* (pp. 33-36). IEEE Computer Society.
13. Cong, J., & Xiao, B. (2011, June). mrFPGA: A novel FPGA architecture with memristor-based reconfiguration. In *Nanoscale Architectures (NANOARCH), 2011 IEEE/ACM International Symposium on* (pp. 1-8). IEEE.
14. Snider, G. S., & Williams, R. S. (2007). Nano/CMOS architectures using a field-programmable nanowire interconnect. *Nanotechnology*, *18*(3), 035204.
15. <http://www.xilinx.com/> (2015, February 23). “XA Kintex-7 FPGAs Overview”. DS182 (v2.13).
16. Biolek, D., Di Ventra, M., & Pershin, Y. V. (2013). Reliable SPICE simulations of memristors, memcapacitors and meminductors. *arXiv preprint arXiv:1307.2717*.
17. Mazady, A. (2014). Modeling, Fabrication, and Characterization of Memristors.
18. Torrezan, A. C., Strachan, J. P., Medeiros-Ribeiro, G., & Williams, R. S. (2011). Sub-nanosecond switching of a tantalum oxide memristor. *Nanotechnology*,*22*(48), 485203.
19. Furao, S., Ogura, T., & Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks*, *20*(8), 893-903.
20. Borghetti, J., Li, Z., Straznicky, J., Li, X., Ohlberg, D. A., Wu, W., ... & Williams, R. S. (2009). A hybrid nanomemristor/transistor logic circuit capable of self-programming. *Proceedings of the National Academy of Sciences*, *106*(6), 1699-1703.
21. Manem, H., Rajendran, J., & Rose, G. S. (2012). Design considerations for multilevel CMOS/nano memristive memory. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *8*(1), 6.
22. International technology roadmap for semiconductors. URL <http://www.itrs.net/>
23. Razavi, B., Lee, K. F., & Yan, R. H. (1995). Design of high-speed, low-power frequency dividers and phase-locked loops in deep submicron CMOS. *Solid-State Circuits, IEEE Journal of*, *30*(2), 101-109.
24. Sun, Y., & Herzel, F. (2006). A fully differential 60 GHz receiver front-end with integrated PLL in SiGe: C BiCMOS. In *2006 European Microwave Integrated Circuits Conference* (pp. 198-201).
25. Pinel, S., Sarkar, S., Sen, P., Perumana, B., Yeh, D., Dawn, D., & Laskar, J. (2008, February). A 90nm cmos 60ghz radio. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International* (pp. 130-601). IEEE.
26. Tsai, K. H., & Liu, S. I. (2012). A 104-GHz phase-locked loop using a VCO at second pole frequency. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, *20*(1), 80-88.
27. Lin, Y., & Kotecki, D. E. (2012, August). A 126.9–132.4 GHz wide-locking low-power frequency-quadrupled phase-locked loop in 130nm SiGe BiCMOS. In*Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on* (pp. 754-757). IEEE.
28. Chen, W. Z., Lu, T. Y., Wang, Y. T., Jian, J. T., Yang, Y. H., & Chang, K. T. (2014). A 160-GHz Frequency-Translation Phase-Locked Loop with RSSI Assisted Frequency Acquisition.
29. <http://www.orcad.com/> (2015). “OrCAD 16.6 Lite Demo Software (OrCAD Capture and PSpice)”. Cadence Design Systems, Inc.
30. Xia, Q., Robinett, W., Cumbie, M. W., Banerjee, N., Cardinali, T. J., Yang, J. J., ... & Williams, R. S. (2009). Memristor− CMOS hybrid integrated circuits for reconfigurable logic. *Nano letters*, *9*(10), 3640-3645.
31. Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., & Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, *10*(4), 1297-1301.
32. Kim, K. H., Gaba, S., Wheeler, D., Cruz-Albrecht, J. M., Hussain, T., Srinivasa, N., & Lu, W. (2011). A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. *Nano letters*, *12*(1), 389-395.
33. Yang, J. J., Pickett, M. D., Li, X., Ohlberg, D. A., Stewart, D. R., & Williams, R. S. (2008). Memristive switching mechanism for metal/oxide/metal nanodevices. *Nature nanotechnology*, *3*(7), 429-433.
34. Raghuvanshi, A., & Perkowski, M. (2014, November). Logic synthesis and a generalized notation for memristor-realized material implication gates. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design* (pp. 470-477). IEEE Press.
35. Kime, C. R., & Mano, M. M. (2004). Logic and computer design fundamentals.
36. Breu, H., Gil, J., Kirkpatrick, D., & Werman, M. (1995). Linear time Euclidean distance transform algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *17*(5), 529-533.
37. Liberti, L., Lavor, C., Maculan, N., & Mucherino, A. (2014). Euclidean distance geometry and applications. *SIAM Review*, *56*(1), 3-69.
38. Parhizkar, R. (2013). *Euclidean distance matrices: Properties, algorithms and applications* (Doctoral dissertation, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE).
39. Coeurjolly, D., & Vacavant, A. (2012). Separable distance transformation and its applications. In *Digital Geometry Algorithms* (pp. 189-214). Springer Netherlands.
40. Dokmanić, I., Parhizkar, R., Walther, A., Lu, Y. M., & Vetterli, M. (2013). Acoustic echoes reveal room shape. *Proceedings of the National Academy of Sciences*, *110*(30), 12186-12191.
41. Ye, Q. Z. (1988, November). The signed Euclidean distance transform and its applications. In *Pattern Recognition, 1988., 9th International Conference on* (pp. 495-499). IEEE.
42. Chu, D. I., Brown, H. C., & Chu, M. T. (2010). On least squares euclidean distance matrix approximation and completion. *Available at February*, *16*.
43. Barrett, P. (2006). Euclidean distance: Raw, normalised, and double-scaled coefficients. *Unpublished paper retrieved from http://www. pbmetrix. com/techpapers/Euclidean\_Distance. pdf*.
44. Krislock, N., & Wolkowicz, H. (2012). *Euclidean distance matrices and applications* (pp. 879-914). Springer US.
45. Dokmanic, I., Parhizkar, R., Ranieri, J., & Vetterli, M. (2015). Euclidean Distance Matrices: A Short Walk Through Theory, Algorithms and Applications. *arXiv preprint arXiv:1502.07541*.
46. Kannan, S., Karimi, N., Karri, R., & Sinanoglu, O. (2015). Modeling, Detection, and Diagnosis of Faults in Multilevel Memristor Memories.*Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *34*(5), 822-834.
47. Lee, C. H., Perkowski, M. A., Hall, D. V., & Jun, D. S. (2000). Self-repairable EPLDs: design, self-repair, and evaluation methodology. In *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on* (pp. 183-193). IEEE.
48. Lee, C. H., Perkowski, M. A., Hall, D. V., & Jun, D. S (2001). Self-Repairable GALs. [Journal of Systems Architecture](https://www.researchgate.net/journal/1383-7621_Journal_of_Systems_Architecture), 02/2001; 47(2):119-135.
49. Li, H., Gao, B., Chen, Z., Zhao, Y., Huang, P., Ye, H., ... & Kang, J. (2015). A learnable parallel processing architecture towards unity of memory and computing. *Scientific reports*, *5*.
50. Austin, J., & Stonham, T. J. (1987). Distributed associative memory for use in scene analysis. *Image and Vision Computing*, *5*(4), 251-260.
51. Bose, J. (2007). *Engineering a Sequence Machine Through Spiking Neurons Employing Rank-order Codes* (Doctoral dissertation, University of Manchester).
52. Bose, J., Furber, S. B., & Shapiro, J. L. (2005). An associative memory for the on-line recognition and prediction of temporal sequences. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on* (Vol. 2, pp. 1223-1228). IEEE.
53. Buckingham, J., & Willshaw, D. (1992). Performance characteristics of the associative net. *Network: Computation in Neural Systems*, *3*(4), 407-414.
54. Carpenter, G. A., & Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, *21*(3), 77-88.
55. Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179-211.
56. Flynn, M. J., Kanerva, P., & Bhadkamkar, N. (1989). Sparse distributed memory: Principles and operation.
57. Furao, S., & Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, *19*(1), 90-106.
58. Ghahramani, Z. (2004). Unsupervised learning. In *Advanced lectures on machine learning* (pp. 72-112). Springer Berlin Heidelberg.
59. Govoreanu, B., Kar, G. S., Chen, Y. Y., Paraschiv, V., Kubicek, S., Fantini, A., ... & Jossart, N. (2011, December). 10× 10nm 2 Hf/HfO x crossbar resistive RAM with excellent performance, reliability and low-energy operation. In *Electron Devices Meeting (IEDM), 2011 IEEE International* (pp. 31-6). IEEE.
60. Guyonneau, R., VanRullen, R., & Thorpe, S. J. (2005). Neurons tune to the earliest spikes through STDP. *Neural Computation*, *17*(4), 859-879.
61. Hawkins, J., Ahmad, S., & Dubinsky, D. (2010). Hierarchical temporal memory including HTM cortical learning algorithms. *Techical report, Numenta, Inc, Palto Alto http://www. numenta. com/htmoverview/education/HTM\_CorticalLearningAlgorithms. pdf*.
62. Hebb, D. O. (2005). *The organization of behavior: A neuropsychological theory*. Psychology Press.
63. Holleman, J., Mishra, A., Diorio, C., & Otis, B. (2008, September). A micro-power neural spike detector and feature extractor in. 13μm CMOS. In*Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE* (pp. 333-336). IEEE.
64. Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, *79*(8), 2554-2558.
65. Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. *Advances in psychology*, *121*, 471-495.
66. Kanerva, P. (1992). Sparse distributed memory and related models.
67. Kim, Y., Zhang, Y., & Li, P. (2012, September). A digital neuromorphic VLSI architecture with memristor crossbar synaptic array for machine learning. In*SOC Conference (SOCC), 2012 IEEE International* (pp. 328-333). IEEE.
68. Kosko, B. (1988). Bidirectional associative memories. *Systems, Man and Cybernetics, IEEE Transactions on*, *18*(1), 49-60.
69. Levenshtein, V. I. (1966, February). Binary codes capable or ‘correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady* (Vol. 10, No. 8).
70. LeCun, Y., Cortes, C., & Burges, C. J. (1998). The MNIST database of handwritten digits.
71. Likharev, K. K. (2011). CrossNets: Neuromorphic hybrid CMOS/nanoelectronic networks. *Science of Advanced Materials*, *3*(3), 322-331.
72. Loiselle, S., Rouat, J., Pressnitzer, D., & Thorpe, S. (2005, July). Exploration of rank order coding with spiking neural networks for speech recognition. In*Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on* (Vol. 4, pp. 2076-2080). IEEE.
73. Palm, G., Schwenker, F., Sommer, F. T., & Strey, A. (1997). Neural associative memories. *Associative processing and processors*, 307-326.
74. Pierzchala, E., & Perkowski, M. A. (1999). *U.S. Patent No. 5,959,871*. Washington, DC: U.S. Patent and Trademark Office.
75. Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257-286.
76. Sakurai, N., Hattori, M., & Ito, H. (2002, May). SOM associative memory for temporal sequences. In *Proceedings of the 2002 international joint conference on neural networks* (pp. 950-955).
77. KP, S. S., Turner, A., Sherly, E., & Austin, J. (2014). Sequential data mining using correlation matrix memory. *arXiv preprint arXiv:1407.2206*.
78. Sharad, M., Augustine, C., Panagopoulos, G., & Roy, K. (2012, June). Cognitive computing with spin-based neural networks. In *Proceedings of the 49th Annual Design Automation Conference* (pp. 1262-1263). ACM.
79. Shen, F., Yu, H., Kasai, W., & Hasegawa, O. (2010, July). An associative memory system for incremental learning and temporal sequence. In *Neural Networks (IJCNN), The 2010 International Joint Conference on* (pp. 1-8). IEEE.
80. Shen, F., Ouyang, Q., Kasai, W., & Hasegawa, O. (2013). A general associative memory based on self-organizing incremental neural network. *Neurocomputing*, *104*, 57-71.
81. Snider, G., Kuekes, P., Hogg, T., & Williams, R. S. (2005). Nanoelectronic architectures. *Applied Physics A*, *80*(6), 1183-1195.
82. Yamada, T., Hattori, M., Morisawa, M., & Ito, H. (1999). Sequential learning for associative memory using Kohonen feature map. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on* (Vol. 3, pp. 1920-1923). IEEE.
83. Yang, J. J., Strukov, D. B., & Stewart, D. R. (2013). Memristive devices for computing. *Nature nanotechnology*, *8*(1), 13-24.
84. Zaveri, M. S., & Hammerstrom, D. (2011). Performance/price estimates for cortex-scale hardware: a design space exploration. *Neural Networks*, *24*(3), 291-304.
85. Zaveri, M. S., & Hammerstrom, D. (2010). CMOL/CMOS implementations of bayesian polytree inference: Digital and mixed-signal architectures and performance/price. *Nanotechnology, IEEE Transactions on*, *9*(2), 194-211.
86. Strukov, D. B., Stewart, D. R., Borghetti, J., Li, X., Pickett, M. D., Medeiros-Ribeiro, G., ... & Xia, Q. (2010, May). Hybrid CMOS/memristor circuits. In*ISCAS* (pp. 1967-1970).
87. Pershin, Y. V., & Di Ventra, M. (2010). Experimental demonstration of associative memory with memristive neural networks. *Neural Networks*,*23*(7), 881-886.
88. Snider, G. S. (2007). Self-organized computation with unreliable, memristive nanodevices. *Nanotechnology*, *18*(36), 365202.
89. Likharev, K., Mayr, A., Muckra, I., & Türel, Ö. (2003). CrossNets: High‐Performance Neuromorphic Architectures for CMOL Circuits. *Annals of the New York Academy of Sciences*, *1006*(1), 146-163.
90. Snider, G., Amerson, R., Gorchetchnikov, A., Mingolla, E., Carter, D., Abdalla, H., ... & Patrick, S. (2011). From synapses to circuitry: Using memristive memory to explore the electronic brain. *Computer*, (2), 21-28.
91. Coleman, J. N., Chester, E. I., Softley, C. I., & Kadlec, J. (2000). Arithmetic on the European logarithmic microprocessor. *Computers, IEEE Transactions on*, *49*(7), 702-715.
92. Taylor, F. J., Gill, R., Joseph, J., & Radke, J. (1988). A 20 bit logarithmic number system processor. *Computers, IEEE Transactions on*, *37*(2), 190-200.
93. Eshraghian, K., Cho, K. R., Kavehei, O., Kang, S. K., Abbott, D., & Kang, S. M. S. (2011). Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, *19*(8), 1407-1417.
94. Lehtonen, E., Poikonen, J. H., & Laiho, M. (2012, August). Applications and limitations of memristive implication logic. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on* (pp. 1-6). IEEE.
95. Patterson, D. A., & Hennessy, J. L. (2013). *Computer organization and design: the hardware/software interface*. Newnes.
96. Hu, X., Duan, S., & Wang, L. (2012, November). A novel chaotic neural network using memristive synapse with applications in associative memory. In *Abstract and Applied Analysis* (Vol. 2012). Hindawi Publishing Corporation.
97. Chu, P. P. (2006). *RTL hardware design using VHDL: coding for efficiency, portability, and scalability*. John Wiley & Sons.
98. Barrabés Castillo, A. (2012). Design of single precision float adder (32-bit numbers) according to IEEE 754 standard using VHDL.
99. Zidan, M. A., Fahmy, H. A. H., Hussain, M. M., & Salama, K. N. (2013). Memristor-based memory: The sneak paths problem and solutions. *Microelectronics Journal*, *44*(2), 176-183.
100. Rahman, K. C. (2014). Proposal Dissertation: Complete Methodology for a massively parallel Biologically Inspired Associative Memory architecture design: An FSMDM and pipelined-SIMD architecture -Implementation with CMOS-CMOL Binary memristor design with Stateful IMPLY gates. Department of Electrical and Computer Engineering, Portland State University, Portland, OR, USA.
101. Rahman, K. C., Xiong, H., Sachindran, S., & Godbole, A. (2014). ECE590 Report: CMOS FPGA Implementation of Euclidean Distance Pipeline using VHDL. Department of Electrical and Computer Engineering, Portland State University, Portland, OR, USA.
102. Rahman, K. C., & Shankar, V. (2015). Memristor Device Modeling. Department of Electrical and Computer Engineering, Portland State University, Portland, OR, USA.

# Appendix A

# **SOFTWARE – MATLAB CODES FOR ESOINN & GAM (PATTERN RECOGNITION ALGORITHMS)**

The motivation of this work is explained in detail in Chapter 1 of this dissertation. For this research, various biologically inspired i.e. neural network based pattern recognition algorithms were studied. In order to understand and compare the performance, some of these promising algorithms were coded in MATLAB as a part of this research. The pseudo codes were published in many papers [51][52][61][73][79][80] and most of those algorithms were coded for this research. These algorithms were then simulated and a comparative performance for the pattern recognition application was conducted. Based on the performance comparison, GAM (General Associative Memory) [80] was found to be the best algorithm for both spatial and temporal pattern recognition.

 Although the original goal of this dissertation work was to develop a hardware design methodology for this best performing algorithm for its complete system, however, later we realized that the implementation of the complete system is unnecessary for providing the design methodology. Therefore, a common critical hardware component was selected to develop the design methodology that is used by most of the neural network and machine learning based algorithms. This component is the Euclidean Distance (ED) Processor/Calculator. ED calculator can be used in a massively parallel and pipelined datapath systems and thus it can have applications in pattern recognition, robot motion, big data analysis, image processing, voice recognition, DSP, database mining and may other hardware systems where large number of wide vectors need to be processed simultaneously.

 The ESOINN [79] and GAM [80] codes are presented in Appendix A.

ESOINN (Enhanced Self-Organizing Incremental Neural Network) Model:

The ESOINN algorithm is an *Unsupervised Learning* algorithm for Spatial Pattern Recognition. Unsupervised learning [79] studies how a system can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns. By contrast with supervised learning or reinforcement learning, in unsupervised learning there are no explicit target outputs or environmental evaluations associated with each input.

GAM (General Associative Memory) Model: This algorithm is an improved version of ESOINN for temporal pattern recognition.

**Handwritten digit database**

This training dataset used for the algorithm was derived from the original MNIST database available at <http://yann.lecun.com/exdb/mnist/> [70]

The training data file for each class 0 to 9 was generated.

File format:

Each file has 1000 training examples. Each training example is of size 28x28 pixels. The pixels are stored as unsigned chars (1 byte) and take values from 0 to 255. The first 28x28 bytes of the file correspond to the first training example, the next 28x28 bytes correspond to the next example and so on.

Algorithm Organization:

* Many classes
* Many sub-classes under each class
* Many nodes under each sub-class

**Data Structure:**

Data of 10 classes - 0 – 9

In each class there are 1000 samples

So potentially there are 10\*1000 = 10K nodes

Node = 28x28 = 784 elements with values between 0 to 255

Node = A vector of 784 elements with values from 0 to 255

Each image is node -> sub-class -> class

When an image is received, first its class is found and then its subclass is identified.

Class will be indexed/identified by numbers 0 - variable

Sub-class will be indexed/identified by numbers 0 - variable

Nodes will be indexed/identified by numbers 0 - variable

Image can be catalogued --> NODE[CI][SCI][NI]

CI -> Class index

SCI -> Sub-class index

NI -> Node index

Node has a local maximum density -> apex of a sub class

**Image Database:** The image database is populated with 10,000 images, of which, the node distribution for digits 0 through 9 is shown in Appendix A-1. Each digit between 0 through 9 represents a *Class*. Each category in Appendix A-1 has 1000 images and each of these images represents a *Subclass* under the *Class*. Appendix A-2 shows MNIST handwritten digits. Each digit has a pixel size of 28x28. The pixels are stored as unsigned chars (1 byte) and take gray-scale values from 0 to 255. The first 28x28 bytes of the training file correspond to the first training example, the next 28x28 bytes correspond to the next example and so on. As such, 10,000 lines were concatenated in one training file.

Table A- 1: Node Representation for Various Digits.

|  |  |
| --- | --- |
| Nodes | Represent Image for Digit |
| 1-1000 | 0 |
| 1001-2000 | 1 |
| 2001-3000 | 2 |
| 3001-4000 | 3 |
| 4001-5000 | 4 |
| 5001-6000 | 5 |
| 6001-7000 | 6 |
| 7001-8000 | 7 |
| 8001-9000 | 8 |
| 9001-10000 | 9 |



Figure A- 1: MNIST Handwritten Digits used in the experiments. Upper left: Classes 2, 4, 5, 8; Right: Subclasses of digit 4; Lower left: 3-D image of digit 4.

**Training database:** There are 400 images randomly picked from the image database and used for training, of which nodes 1-100 represent image 2; nodes 101-200 represent image 4; nodes 201-300 represent image 5 and nodes 301-400 represent image 8.

**Test database:** There are 200 images randomly picked from the image database and used for testing, of which nodes 1-50 represent image 2; nodes 51-100 represent image 4; nodes 101-150 represent image 5 and nodes 151-200 represent image 8. The number of test images is a smaller set compared to the number of training images.

**Distance Threshold constant:**

A distance threshold constant is used to control the classification of a new node to a new class or to an existing class. During the experimentation, the values of distance threshold are changed several times. A small value of distance threshold may result in a large number of classes. For example, after some trial and error, for the four broader input classes (digits 2, 4, 5, 8) as mentioned above, a large number of classes can be obtained at the output. With further experimentation, it is possible to obtain even fewer classes at the output by iterating on the distance threshold constant.

ESOINN MODEL:

readdata.m

clear all

%open the file corresponding to digit

k=1;

l=1;

for j=[1 4 5 8]

 filename = strcat('MNIST\data',num2str(j),'.txt');

 [fid(k) msg] = fopen(filename,'r');

 filename

 %read in the first training example and store it in a 28x28 size matrix t1

 for i=1:100

 [data28x28,N]=fread(fid(k),[28 28],'uchar');

 data(l,:) = reshape(data28x28,1,28\*28);

 dataX = reshape(data28x28,1,28\*28);

 l = l+1;

 %imshow(data28x28');

 %pause(0.5)

 end

 k = k+1;

end

save ('numimagedat4\_1.mat','data');

distcalc.m

function z = distcalc(w,p)

%DIST Euclidean distance weight function.

% Algorithm

% The Euclidean distance D between two vectors X and Y is:

% D = sqrt(sum((x-y).^2))

[S,R] = size(w);

[R2,Q] = size(p);

if (R ~= R2), error('Inner matrix dimensions do not match.'),end

z = zeros(S,Q);

if (Q<S)

 p = p';

 copies = zeros(1,S);

 for q=1:Q

 z(:,q) = sum((w-p(q+copies,:)).^2,2);

 end

else

 w = w';

 copies = zeros(1,Q);

 for i=1:S

 z(i,:) = sum((w(:,i+copies)-p).^2,1);

 end

end

z = sqrt(z)/R;

findthreshold.m

% given a set of nodes, find maximum & minimum sim\_threshold of each of the nodes.

function [TMax, TMin] = findthreshold(a,DIST\_THRESH)

[NRow,MCol] = size(a);

for i=1:NRow % assuming I have 100 nodes

 TMax(i) = 0;

 TMin(i) = 9999;

 for j=1:NRow

 dist = distcalc (a(i,:), a(j,:)');

 %fprintf('%f %f\n',DIST\_THRESH, dist);

 if(dist < DIST\_THRESH)

 if dist > TMax(i)

 TMax(i) = dist;

 end

 if dist < TMin(i)

 TMin(i) = dist;

 end

 end

 end

end

return

findwinners.m

% given a set of nodes, find winner and second winner.

function [winner, winner2, DWinner, DWinner2] = findwinners(a,x)

[NRow,MCol] = size(a);

for i=1:NRow % assuming I have 100 nodes

 dist(i) = distcalc (x, a(i,:)');

end

if dist(1) < dist(2)

 winner = 1;

 winner2 = 2;

else

 winner = 2;

 winner2= 1;

end

for i= 3:NRow

 if dist(i) < dist(winner)

 temp = winner;

 winner = i;

 if dist(winner2) > dist(temp);

 winner2 = temp;

 end

 else

 if dist(i) < dist(winner2)

 winner2 = i;

 end

 end

end

DWinner = dist(winner);

DWinner2 = dist(winner2);

return

find winnersX.m

% given a set of nodes, find winner and second winner.

function [winner, winner2, DWinner, DWinner2] = findwinnersX(a,x)

[NRow,MCol] = size(a);

for i=1:NRow % assuming I have 100 nodes

 dist(i) = distcalc (x, a(i,:)');

end

if dist(1) < dist(2)

 winner = 1;

 winner2 = 2;

else

 winner = 2;

 winner2= 1;

end

for i= 3:NRow

 if dist(i) < dist(winner)

 temp = winner;

 winner = i;

 if dist(winner2) > dist(temp);

 winner2 = temp;

 end

 else

 if dist(i) < dist(winner2)

 winner2 = i;

 end

 end

end

DWinner = dist(winner);

DWinner2 = dist(winner2);

% if DWinner == 0

% DWinner

% sparse(a)

% sparse(x)

% end

Return

find\_neighbors.m

function [nghbrs] = find\_neighbors(winner, W, DIST\_THRESH)

% find how many nodes in the sub space

[SR SC] = size( W);

cnt = 1;

for i=1:SR

 dist = distcalc(W(winner,:), W(i,:)');

 if(dist < DIST\_THRESH)

 nghbrs(cnt) = i;

 cnt = cnt + 1;

 end

end

end

return

soinn\_subclass.m

clear all

load soinn\_400.mat

% pick class

[RC SC] = size(class\_of\_node);

% initialize

for i = 1:SC

 visited(i) = 0;

 subclass(i) = 0;

end

% now do the classification

% "Connections matrix" is tracking all the connected nodes of a given node

for i = 1:SC

 k = 1;

 for j = 1:SC

 if(i ~= j)

 if (Conn(i,j) == 1)

 Connections(i,k) = j; % Connection recorded

 k = k + 1;

 end

 end

 end

end

% Find density of each node

for p = 1:NClass

 scindx = 1;

 for i = 1:SC

 if ((visited(i) == 0) && (class\_of\_node(i) == p))

 k=1;

 clear visited\_t;

 %fprintf ('class = %d node = %d\n',p,i);

 marker = 99;

 max = h(i);

 max\_node = i;

 visited\_t(k) = i; % Keepingtrack of visited tree

 visited(i) = 1; % Keeping track of the nodes that are already worked on

 current\_node = i;

 new\_marker = marker + 1 ; % this is a way to flag the last node of the tree

 [max, max\_node, new\_marker, visited, visited\_t, k] = search\_node\_tree(Connections, max, max\_node, marker, current\_node, k, h, visited\_t, visited);

 while (new\_marker > marker)

 marker = new\_marker;

 [max, max\_node, new\_marker, visited, visited\_t, k] = search\_node\_tree(Connections, max, max\_node, marker, current\_node, k, h, visited\_t, visited);

 current\_node = max\_node;

 end

 % done searching that tree

 % assign sub-class here

 [X, TNodesInTree] = size(visited\_t);

 %disp ('visited\_tree')

 visited\_t;

 %disp('visited of current node')

 visited(current\_node);

 for m=1:TNodesInTree

 subclass(visited\_t(m) ) = scindx;

 end

 subclass\_elems{p,scindx,:} = visited\_t;

 subclass\_apex{p,scindx} = max\_node; % Node with highest density of a given subclass

 scindx = scindx + 1;

 end

 end

 p;

 scindxcount(p) = scindx -1;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For testing writing the results to a text file

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fileID = fopen('organize.txt','w');

for i = 1: SC

% %fprintf (fileID, 'class = %d subclass = %d node = %d\n',class\_of\_node(i), subclass(i), i);

 fprintf (fileID, '%d %d %d\n',class\_of\_node(i), subclass(i), i);

 end

 fclose(fileID);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Following is needed for subclass merging

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for p = 1:NClass

 for m = 1:scindxcount(p)

 sum(p,m) = 0;

 count(p,m) = 0;

 end

end

for p=1:NClass

 for m=1:scindxcount(p)

 for i=1:SC

 if( (class\_of\_node(i) == p) && (subclass(i) == m))

 sum(p,m) = sum(p,m) + h(i);

 count(p,m) = count(p,m) + 1;

 end

 end

 end

end

for p=1:NClass

 for m=1:scindxcount(p)

 Avrg(p,m) = sum(p,m)/count(p,m);

 end

end

[dataR dataC] = size(W);

for p=1:NClass

 fprintf('Total elements in class %d is %d\n',p,scindxcount(p));

 for m=1:scindxcount(p)

 clear other\_nodes;

 if(scindxcount(p) > 1) % there is no point of finding winner and second-winners to other subclasses when we have only 1 subclass

 mxnode = subclass\_apex{p,m};

 for j=1:scindxcount(p)

 scwinner(p,m,j) = 0;

 scwinner2(p,m,j) = 0;

 scDWinner(p,m,j) = 0;

 scDWinner2(p,m,j) = 0;

 all\_elems\_of\_subclass = subclass\_elems{p,j,:};

 [A Sz] = size(all\_elems\_of\_subclass);

 other\_nodes = zeros(Sz,dataC);

 for i=1:Sz

 other\_nodes(i,:) = W(all\_elems\_of\_subclass(i),:);

 end

 subclass\_elems{p,j,:}

 if(Sz == 1)

 SnglNode = subclass\_elems{p,j,:};

 scwinner(p,m,j) = subclass\_elems{p,j,:};

 scwinner2(p,m,j) = subclass\_elems{p,j,:};

 scDWinner(p,m,j) = distcalc(W(SnglNode,:), W(mxnode,:)');

 scDWinner2(p,m,j) = scDWinner(p,m,j);

 else

 MoreNodeArray = subclass\_elems{p,j,:};

 [WW1,WW2,scDWinner(p,m,j), scDWinner2(p,m,j)] = findwinnersX(other\_nodes,W(mxnode,:));

 scwinner(p,m,j) = MoreNodeArray(WW1);

 scwinner2(p,m,j) = MoreNodeArray(WW2);

 end

 clear other\_nodes;

 fprintf ('p=%d m=%d, winner=%d, winner2=%d\n',p,m,scwinner(p,m,j), scwinner2(p,m,j));

 end

 end

 end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Check if the two subclasses need to be merged

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for p=1:NClass

 for m=1:scindxcount(p)

 for j=1:scindxcount(p)

 fprintf ('==>[%d %d %d] %d %d %f %f\n',p,m,j,scwinner(p,m,j), scwinner2(p,m,j),scDWinner(p,m,j), scDWinner2(p,m,j));

 end

 end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If nodes from two sub classes are connected -> disconnect

% This is true for even if the two subclasses belong to two different class

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i = 1:SC

 for j = 1:SC

 if((i ~= j) && (subclass(i) ~= subclass(j)))

 if (Conn(i,j) == 1)

 Conn(i,j) = 0;

 end

 end

 end

end

subclass\_test.m

load soinn.mat

[SA SB] = size(class\_of\_node);

for ii = 1:SB

 if(class\_of\_node (ii) == 2)

 point\_density(ii)

 end

end

updt\_winner.m

function [A] = updt\_winner(winner, x, W, M)

[SR SC] = size( W);

for j = 1:SC

 dW(j) = x(j) - W(winner,j);

 A(j) = W(winner,j) + dW(j)/M(winner);

end

return

updt\_neighbors.m

function [W] = updt\_neighbors(winner, nghbrs, x, W, M)

[SR SC] = size( W);

[SNR SNC] = size(nghbrs);

for k = 1: SNC

 if(nghbrs(k) ~= winner) % We do not want to update winner again

 for j = 1:SC

 dW(j) = x(j) - W(nghbrs(k),j);

 W(nghbrs(k)) = W(nghbrs(k),j) + dW(j)/(100\*M(winner));

 %fprintf('neighbor node = %d\n',nghbrs(k));

 end

 end

end

return

updt\_connection\_matrix.m

function [Conn] = update\_connection\_matrix (Conn, CN, value)

[SR, SC] = size(Conn);

for i = 1:SR

 Conn(CN,SR) = value;

end

return;

updt\_conn\_edge\_n\_point\_density.m

function [Conn, Age, point\_density] = update\_conn\_edge\_n\_point\_density(W, Conn, Age, winner)

% Conn -- Connectivity matrix

% W -- Weight vectors of each node

% Age -- age of each connection. So all possible connection edge will have

% an "age" value

% winner - winner node

% Size of connection matrix will determine the

% size of existing node space

%disp('Weight::')

%W

[SR, SC] = size(Conn);

Agemax = 100;

point\_density = zeros(SR,1);

avg\_density = zeros(SR,1);

% Search for all connectivity to winner and update their connection age

for i = 1: SC

 if Conn(winner, i) == 1

 Age(winner, i) = Age(winner, i) + 1;

 if Age(winner, i) > Agemax

 Conn(winner, i) = 0;

 end

 end

end

% Now calculate the point density of ALL the nodes

for i = 1: SR

 dist = 0;

 M=0; % Number of connections with the given node "i"

 for j = 1: SC

 if i ~= j

 if Conn(i, j) == 1

 % W(i,:)

 % W(j,:)

 dist = dist + distcalc(W(i,:),W(j,:)');

 M = M + 1;

 end

 end

 end

 % Calculate Average Density

 if(M > 0)

 avg\_density(i) = dist/M;

 else

 avg\_density(i) = 0;

 end

 if M == 0

 point\_density(i) = 0;

 else

 point\_density(i) = 1/ (1 + avg\_density(i))^2;

 end

end

return

search\_node\_tree.m

function [max, max\_node, new\_marker, visited, visited\_t, k] = search\_node\_tree (Connections, max, max\_node, marker, current\_node, k, h, visited\_t, visited)

% Now lets find the largest connected tree because that will determine the

% final size of the "Connections" matrix

[CR, CC] = size(Connections);

new\_marker = marker;

k;

for jc = 1:CC % checking connections of the nodes connected to i

 j = Connections(current\_node,jc);

 if ( j ~= 0)

 %fprintf ('=> %d %d %d %d %d\n',current\_node, j, k, visited(current\_node), visited(j));

 end

 if ( ( j ~= 0) && (max\_node ~= j))

 if (visited(j) ~= 1)

 k = k+1;

 visited\_t(k) = j;

 visited(j) = 1;

 if (h(j) > h(max\_node))

 max = h(j);

 max\_node = j;

 new\_marker = marker + 1;

 end

 end

 %fprintf ('===> %d %d %d\n',current\_node, j, k);

 visited\_t;

 end

soinn.m

clear all

load numimagedat4.mat

% Select two random entries from the image database to

% initialize the SOINN system

dist = 0;

[DataSize,DataElems] = size(data);

DIST\_THRESH = 3.00; %% used for determining the neighboring nodes

while(dist < 2.5)

 randindx1 = (round(rand(1)\*(DataSize-1)) +1);

 randindx2 = (round(rand(1)\*(DataSize-1)) +1);

 W(1,:) = data(randindx1,:);

 W(2,:) = data(randindx2,:);

 sd = 0;

% i stands for row vector and ik stands for column values in each row

 for ik=1:784

 sd = sd + (W(1,ik) - W(2,ik))^2;

 end

 dist = sqrt(sd)/784;

 TMax(1) = dist;

 TMax(2) = dist;

end

% Now the system has two nodes

N= 2;

NClass = 2;

%class(class,node#)=node#

class\_of\_node(1) = 1;

class\_of\_node(2) = 2;

Conn(1,1) = 1;

Conn(1,2) = 0;

Conn(2,1) = 0;

Conn(2,2) = 1;

Age(1,1) = 0;

Age(1,2) = 0;

M(1) = 1;

M(2) = 1;

% Introduce new nodes (i.e. images) to the system

for i = 1: DataSize-2

 indx = i;

 % CN --- index of the nodes as a new input is introduced

 CN = 2 + i;

 x = data(indx, :);

 Conn(CN,CN) = 1;

 Age(CN,CN) = 0;

 [winner, winner2,DWinner, DWinner2] = findwinners(W,x);

 W(CN,:)= x;

 M(CN) = 1;

 % update connection matrix for the new member with no connection to

 % begin with

 [Conn] = update\_connection\_matrix (Conn, CN, 0);

 % W - Weight matrix

 % Conn - Connection matrix

 % Age = Age matrix

 % winner - ID of the winner node

 if DWinner > TMax(winner) % A new class.

 NClass = NClass+1;

 class\_of\_node(CN) = NClass;

 [TMax, TMin] = findthreshold(W,DIST\_THRESH);

 Conn(CN, winner) = 0;

 Age(CN, winner) = 0;

 Conn(CN, winner2) = 0;

 Age(CN, winner2) = 0;

 point\_density(CN) = 0;

 size(Conn);

 else % step4 - member of existing class of the winner node

 class\_of\_node(CN) = class\_of\_node(winner);

 M(winner) = M(winner) + 1;

 [TMax, TMin] = findthreshold(W,DIST\_THRESH);

 Conn(CN, winner) = 1; % establishing a connection between winner and the new node

 Conn(winner, CN) = 1;

 dw1w2 = distcalc(winner, winner2);

 Age(CN, winner) = 0; % setting age to 0

 Age(winner, CN) = 0;

 if(dw1w2 < DIST\_THRESH)

 Conn(winner, winner2) = 1;

 Conn(winner2, winner) = 1;

 Age(winner, winner2) = 0;

 Age(winner2, winner) = 0;

 end

 %%% Update weight of winner and its neighbors

 % find neighbors of winner

 [nghbrs] = find\_neighbors(winner, W, DIST\_THRESH);

 % update weight of winner

 [W(winner,:)] = updt\_winner(winner, x, W, M);

 % update weight of neighbor

 [W] = updt\_neighbors(winner, nghbrs, x, W, M);

 % disp('Weight::');

 %W

 [Conn, Age, point\_density] = update\_conn\_edge\_n\_point\_density(W, Conn, Age, winner);

 % Now that I updated the point density of one node, I need to

 % update the accumulated point density of every one

 end

 size(point\_density);

 point\_density';

 for kk = 1: i-1

 % kk is the row and CN is the column.

 % kk tracks the history of the

 % previous learnings as a row of the

 % "point\_density\_history" matrix.

 % Since each row has to hold same number

 % of columns and as we learn

 % new items, number of columns grow,

 % we have to zero pad the earlier

 % rows to accommodate the size growth for the new entry

 point\_density\_history(kk,CN) = 0;

 end

 point\_density\_history(i,:) = point\_density';

 [sr, sc] = size(point\_density\_history);

 for col = 1:sc

 NN = sum(spones(point\_density\_history(:,col)));

 accum\_point\_density(col) = sum(point\_density\_history(:,col));

 mean\_accum\_point\_density(col) = accum\_point\_density(col)/NN;

 h(col)= mean\_accum\_point\_density(col);

 end

end

save('soinn\_400.mat')

GAM MODEL:

* soinn\_12\_train\_v0: Implementation of algorithm 1 & 2 for training the memory layer and creating the associative layer.

% In algorithm at first we put all nodes into one class

% For training you go with known classes of data as suggested in GAM

% Or you go with unsupervised learning as suggested in SOINN

%

% ALGORITHM 1: Learning of the memory layer

% ALGORITHM 2: Building Associative Layer

clear all

tic

for ClsName=1:10

 FName = strcat('traindata\_p',num2str(ClsName),'.mat');

 FName

 load ( FName );

 [DataSize,DataElems] = size(data);

 % introduce new node - Step 4

 Class(ClsName).Node(1).W = data(1,:);

 Class(ClsName).Node(1).Th = 0;

 Class(ClsName).Node(1).M = 1; % Frequency of winning of that node

 Class(ClsName).Node(1).N = 0; %

 Class(ClsName).NodeCount = 1;

 ClassCount = 1;

 Class(ClsName).ConnMatrix(1,1) = 1;

 Class(ClsName).ConnAge(1,1) = 1;

 for indx = 2: DataSize

 x = data(indx,:);

 DoneClassification = 0; % Reset it every time

 % you processed a new node

 XX= ['Training Class => ',num2str(ClsName),' New data => ',num2str(indx)];

 disp(XX);

 % Find winner and second winner - step 6 - 8

 WinnerNode = 1;

 Winner2Node = 1;

 WinnerDistance = 0;

 Winner2Distance = 0;

 for Ni = 1:Class(ClsName).NodeCount

 dist = distcalcSOINON(Class(ClsName).Node(Ni).W ,x);

 %dd = sprintf ('Now Processing indx: %5d -> Node: %5d dist: %f [Node Th: %f]' , indx, Ni, dist, Class(ClsName).Node(Ni).Th );

 %disp(dd);

 if (dist > Class(ClsName).Node(Ni).Th) % Step 8

 %disp('dist > thr');

 if Class(ClsName).Node(Ni).Th == 0

 %disp('=> Wd = 0');

 WinnerNode = Ni;

 Winner2Node = Ni;

 WinnerDistance = dist;

 Winner2Distance = dist;

 Class(ClsName).Node(Ni).Th = dist;

 else

 if WinnerDistance == Winner2Distance

 %disp( '=> Wd == W2d');

 if WinnerDistance == 0

 Winner2Node = Ni;

 Winner2Distance = dist;

 WinnerNode = Ni;

 WinnerDistance = dist;

 elseif dist > WinnerDistance

 Winner2Node = Ni;

 Winner2Distance = dist;

 else

 WinnerNode = Ni;

 WinnerDistance = dist;

 end

 elseif dist < Winner2Distance

 %disp('=> dist < W2d');

 Winner2Node = Ni;

 Winner2Distance = dist;

 else

 %disp([' > th but ..',WinnerDistance,Winner2Distance]);

 end

 end

 else

 % Update winner and second winner - Step 6

 if dist <= Class(ClsName).Node(Ni).Th

 Winner2Distance = WinnerDistance;

 Winner2Node = WinnerNode;

 WinnerDistance = dist;

 WinnerNode = Ni;

 elseif dist < Winner2Distance

 Winner2Distance = dist;

 Winner2Node = Ni;

 end

 end

 %dd = sprintf ('Node: %5d -> Wd: %5.3f WN: %5d W2d: %5.3f W2N: %5d' , Ni, WinnerDistance,WinnerNode, Winner2Distance, Winner2Node );

 %disp(dd);

 end

 %Class(Ci).NodeCount

 %dd = sprintf('Classification Done for indx: %d, NodeCount: %d, Wd: %f Th: %f', indx,Class(ClsName).NodeCount,WinnerDistance, Class(ClsName).Node(WinnerNode).Th);

 %disp(dd);

 Class(ClsName).Node(WinnerNode).M = Class(ClsName).Node(WinnerNode).M + 1; % step 6

 if WinnerDistance > Class(ClsName).Node(WinnerNode).Th % Step 8

 %disp( ['introduce new node to the class', WinnerDistance, ' > ' ,Class(ClsName).Node(WinnerNode).Th ]);

 NNi = Class(ClsName).NodeCount+1;

 Class(ClsName).NodeCount = Class(ClsName).NodeCount + 1;

 Class(ClsName).Node(NNi).W = x;

 Class(ClsName).Node(NNi).M = 1;

 Class(ClsName).Node(NNi).N = 0;

 % Update thresholds

 Class(ClsName).Node(NNi).Th = dist;

 Class(ClsName).Node(Ni).Th = dist;

 elseif WinnerDistance == Winner2Distance

 %disp( ['introduce new node to the class', WinnerDistance ' == ' ,Winner2Distance]);

 NNi = Class(ClsName).NodeCount+1;

 Class(ClsName).NodeCount = Class(ClsName).NodeCount + 1;

 Class(ClsName).Node(NNi).W = x;

 Class(ClsName).Node(NNi).M = 1;

 Class(ClsName).Node(NNi).N = 0;

 % Update thresholds

 Class(ClsName).Node(NNi).Th = dist;

 Class(ClsName).Node(Ni).Th = dist;

 else % Step 10

 delS1 = 1/Class(ClsName).Node(WinnerNode).M;

 delS2 = 1/Class(ClsName).Node(Winner2Node).M;

 Class(ClsName).Node(WinnerNode).W = Class(ClsName).Node(WinnerNode).W + delS1\*(x-Class(ClsName).Node(WinnerNode).W); % eq 10

 Class(ClsName).Node(Winner2Node).W = Class(ClsName).Node(Winner2Node).W + delS2\*(x-Class(ClsName).Node(Winner2Node).W); % eq 11

 Class(ClsName).Node(WinnerNode).Th = (Class(ClsName).Node(WinnerNode).Th + WinnerDistance)/2; %eq 12

 end

 Class(ClsName).ConnMatrix(WinnerNode,Winner2Node) = 1; %Step 13

 Class(ClsName).ConnAge(WinnerNode,Winner2Node) = 0; %Step 14

 Class(ClsName).ConnMatrix(Winner2Node,WinnerNode) = 1; %Step 13

 Class(ClsName).ConnAge(Winner2Node,WinnerNode) = 0; %Step 14

 %image(reshape((Class(ClsName).Node(WinnerNode).W),28,28)')

 %pause(1)

 % Step 15

 [NS\_1 NS\_2] = size(Class(ClsName).ConnAge(WinnerNode,:));

 for jk = 1:NS\_2

 if Class(ClsName).ConnMatrix(WinnerNode,jk) == 1

 Class(ClsName).ConnAge(WinnerNode,jk) = Class(ClsName).ConnAge(WinnerNode,jk) + 1;

 end

 end

 end

 [Ns1 Ns2] = size(Class(ClsName).Node);

 MostVisNode = 1;

 MostVisNodeM = 1;

 for Mn=1:Ns2

 if Class(ClsName).Node(Mn).M > MostVisNodeM

 MostVisNode = Mn;

 MostVisNodeM = Class(ClsName).Node(Mn).M;

 end

 end

 % Build associative layer

 AssocClass(ClsName).Wb = Class(ClsName).Node(MostVisNode);

 AssocClass(ClsName).Mb = 0;

end

save('soinn\_trained\_assoc.mat')

toc

* soinn\_2\_v0: training the associative layer with temporal sequence.

% Learning of the associative layer

% 2-4-1-3

% key-rwaponse vector

% 2-4

% 4-1

% 1-3

clear all

tic % to measure the CPU time of the algorithm

load('all\_input\_data\_flat.mat');

% load the pre-trained node space

load('soinn\_trained\_assoc.mat');

% Start with a key/control vector

[CDCnt CDLen] = size(Control\_Vec);

AssocClassConnMatrix = zeros (10,10);

RespClass = zeros (10,10);

for j = 1:CDCnt

 % Here we find which class a given Control Vector belongs to

 j

 [MinClassCnt MinNodeCnt MinDistCnt] = memlayer\_classification\_v0(Control\_Vec(j,:),Class)

 [MinClassRes MinNodeRes MinDistRes] = memlayer\_classification\_v0(Response\_Vec(j,:),Class)

 % TBD: Update the node space of the class with the information of the new

 % node

 % Build Association - Step 19,23,26/A-2

 if AssocClassConnMatrix(MinClassCnt,MinClassRes) <= 0

 AssocClassConnMatrix(MinClassCnt,MinClassRes) = 1;

 else

 AssocClassConnMatrix(MinClassCnt,MinClassRes) = AssocClassConnMatrix(MinClassCnt,MinClassRes) + 1;

 end

 % associative index of Node i

 AssocIndxNode(MinClassCnt,MinNodeCnt) = MinNodeRes;

 AssocIndxClass(MinClassCnt,MinNodeCnt) = MinClassRes;

 % Response class of Node i

 RespClass(MinClassCnt,MinClassRes) = RespClass(MinClassCnt,MinClassRes) + 1;

end

toc

Supporting Codes:

* readdata: For creating the training and testing vector for creating memory layer.

% Generating train and test data from MNIST data set

clear all

%open the file corresponding to digit

k=1;

for j=[1 2 3 4 5 6 7 8 9 0]

 filename = strcat('Users/Kamela/Documents/MatLabCodes/Codes\_ESOINN/MNIST/data',num2str(j),'.txt');

 [fid(k) msg] = fopen(filename,'r');

 filename

 l=1;

 %read in the first training example

 % and store it in a 28x28 size matrix t1

 for i=1:2:100

% for i=2:2:100

 [data28x28,N]=fread(fid(k),[28 28],'uchar');

 data(l,:) = reshape(data28x28,1,28\*28);

 dataX = reshape(data28x28,1,28\*28);

 l = l+1;

 %imshow(data28x28');

 %pause(0.5)

 end

 fname = strcat('traindata\_p',num2str(k),'.mat');

% fname = strcat('testdata\_p',num2str(k),'.mat');

 save (fname,'data');

 k = k+1;

end

* prep\_key\_response\_vector\_data: For creating temporal sequence for training and inference.

% Generating train and test data

% from MNIST data set

clear all

%open the file corresponding to digit

k=1;

for j=[1 2 3 4 5 6 7 8 9 0]

 filename = strcat('Users/Kamela/Documents/MatLabCodes/Codes\_ESOINN/MNIST/data',num2str(j),'.txt');

 [fid(k) msg] = fopen(filename,'r');

 filename

 l=1;

 %read in the first training example

 % and store it in a 28x28 size matrix t1

 for i=1:2:100

 % for i=2:2:100

 [data28x28,N]=fread(fid(k),[28 28],'uchar');

 data(k,l,:) = reshape(data28x28,1,28\*28);

 dataX = reshape(data28x28,1,28\*28);

 l = l+1;

 end

 k = k+1;

end

% Create control and response vectors from the training data

l = 1;

for j=1:50

 Control\_Vec(l,:) = data(1,j,:);

 Response\_Vec(l,:) = data(3,j,:);

 l = l+1;

 Control\_Vec(l,:) = data(2,j,:);

 Response\_Vec(l,:) = data(4,j,:);

 l = l+1;

 Control\_Vec(l,:) = data(4,j,:);

 Response\_Vec(l,:) = data(1,j,:);

 l = l + 1;

end

fname = strcat('all\_input\_data\_flat','.mat');

save (fname,'data','Control\_Vec','Response\_Vec');

* memlayer\_classification\_v0: To classify a new incoming node. Used in training temporal sequence.

function [MinClass MinNode MinDist] = memlayer\_classification\_v0(x,Class)

% x = input vector

% Class = Node Space information

% Class =

%

% Node: [1xn struct]

% NodeCount: n

% ConnMatrix: [pxp double]

% ConnAge: [pxp double]

tic

[a b] = size(Class);

MinDist = 99999;

MinClass = 0;

MinNode=0;

for m=1:b

 Class(m).NodeCount;

 [Ns1 Ns2] = size(Class(m).Node);

 MostVisNode(m) = 1;

 for n=1:Ns2

 dist = distcalcSOINON(Class(m).Node(n).W, x);

 if dist < MinDist

 MinDist = dist;

 MinClass = m;

 MinNode = n;

 end

 end

end

toc

distcalcSOINON.m

function z = distcalcSOINON(w,p)

%DIST Euclidean distance weight function.

% Algorithm

% The Euclidean distance D between two vectors X and Y is:

% D = sqrt(sum((x-y).^2))

[S,R] = size(w);

[R2,Q] = size(p);

if (R ~= Q), error('Inner matrix dimensions of R and Q do not match.'),end

if (S ~= R2), error('Inner matrix dimensions of S and R2 do not match.'),end

Tot = 0;

for i = 1:R

 Tot = Tot + (w(i) - p(i))^2;

end

z = sqrt(Tot)/R;