VLSI Architecture for Hierarchical Temporal Memory

Wim J. C. Melis and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
6-6-05, Aoba, Aramaki, Aoba-ku, Sendai 980-8579, Japan

Abstract: A large number of real world applications, such as image recognition and understanding, can still not be performed easily by conventional algorithms in comparison with the human brain. Implementing applications that require such intelligence, might therefore require a different approach, for which Hierarhical Temporal Memory (HTM) seems a promising framework. Currently HTM exists as a software model. However, this software implementation has its performance limitations and a distinct learning and operating mode. This paper proposes a possible architecture for VLSI implementation that also allows learning during normal operation.

1. Introduction

Real world applications have formed a continuous drive towards finding better algorithms. However there are still a large number of applications that cannot be performed easily in comparison with the way the human brain performs them.

In the past, the human brain has inspired and driven several research areas that wanted to improve the intelligence of computing systems. These research areas are better known as Artificial Intelligence [1] and Neural Networks [2], [3]. Although the amount of research in these areas is numerous, the results have not reached what was hoped for. Possibly because they were focusing on which operations to perform to be intelligent, instead of making the system intelligent in its roots. One framework which seems promising in offering similar intelligence as the human brain is Hierarchical Temporal Memory (HTM) [4], [5].

This paper starts of with describing HTM, which achieves intelligence by being trained with the data it should remember. Currently this intelligent machine is however only available as a software model, which has its performance limitations, but also a distinct difference between learning and normal operation. Therefore, this paper proposes a possible architecture for VLSI that also allows learning during normal operation.

2. Hierarchical Temporal Memory

Hierarchical Temporal Memory (HTM) is in essence a method of modelling the intelligence of the human brain. As the name implies it is a hierarchical structure (Figure 1 a) of nodes (Figure 1 b) which mainly consist of memory. Each node performing the same algorithm.

The hierarchical structure is supposed to be similar to the way neurons are connected within the human brain. At the bottom of the hierarchy, the nodes are fed with the sensory input data, which could e.g. be visual or auditory information. At the top of the hierarchy, an output is provided. This output gives an indication of which category of data was provided at the input. For example if the system is trained to recognise

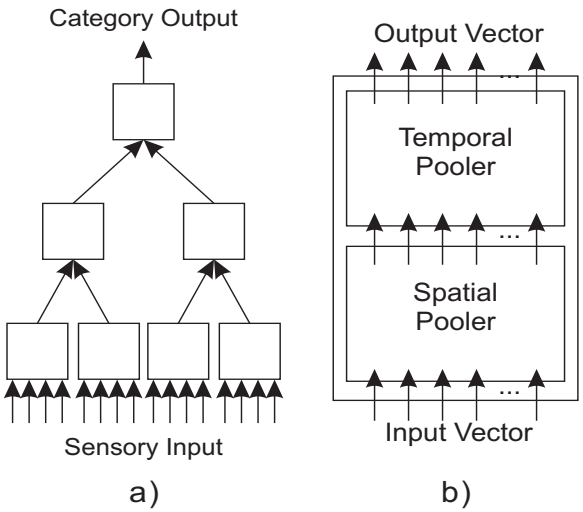


Figure 1. Hierarchical structure of HTM (a), and structure of each individual node (b).

animals, then the output would be whether for example a cat or dog was recognised from the input data. Although each node performs the same algorithm, it performs it on different data. While the lowest level operates on the input data, when moving up the hierarchy the abstraction of the processed data increases. The actual node algorithm consists of two parts namely a spatial and a temporal pooler. The spatial pooler is designed to deal with common patterns in space, whereas the temporal pooler stores the common sequences among these spatial patterns.

Just like a human brain, the system needs to be trained before it is capable of performing its required function. Therefore, the system knows two exclusive modes, namely: learning mode and inference/normal operation mode. When a bottom node is in learning mode, the spatial pooler stores any patterns that commonly occur in the learning set within its local memory. Each incoming pattern is compared with the already stored patterns to determine whether it is sufficiently similar to an already stored pattern, and only when not sufficiently similar to an existing pattern it is stored as a new entry. Once the spatial pooler has gone through the full set of training data, it swaps to normal operation. In normal operation, the input provided to the spatial pooler is compared with the stored patterns, and a vector indicating similarity is generated as output. Each value within the vector represents similarity with one of the patterns stored during learning phase. At this point the temporal pooler starts to learn. Since the temporal pooler stores sequences of the spatial pooler patterns, it initially stores data about the sequence of occurrence of these patterns, and this information is then eventually used to group

the frequently occurring sequences. During normal operation, the temporal pooler also provides a vector as output, where each value within the vector indices its resemblance with a specific group of the temporal pooler. This output then combines with the output of another node to become input to a higher level node. Consequently a next level of nodes can only learn when a lower level of nodes has finished learning, and therefore swapped to normal operation.

3. Architecture for VLSI

Due to the fact that HTM is a system that learns instead of being programmed, a VLSI implementation requires high flexibility, as well as dynamic configuration. Initially when the system is defined, only the number of inputs to the sensor node as well as the maximum number of coincidences and groups for spatial and temporal pooler respectively are defined, together with a value indicating the maximum distance within which two vectors shall be treated as similar.

Hence each node should contain a certain amount of memory for spatial and temporal patterns. However, extra overflow memory is provided as a shared resource among several nodes. To reduce the possibility of memory access bottlenecks to this shared memory, the most frequently used items should be stored in the memory local to the node. This requires the stored spatial and temporal patterns to be sorted within the memory according to their frequency of occurrence. For the spatial patterns this can easily be obtained during learning stage, though for the temporal patterns this would be the most common time sequence possibly combined with which sequence occurred last. The sorting of coincidence data stored in the memory, should be performed at the end of the learning phase for the spatial pooler, whereas the last occurred sequence in time for the spatial pooler could be implemented as a pointer. This pointer should then be used as an indication of which location should be checked first when new data comes in. In addition to the number of entries for storage determined at design time, the extra available memory can be used for learning during normal operation. Namely any pattern during normal operation will be compared with the previously stored patterns, which will determine the values for the output vector of that block. If this new pattern does not match any of the previously stored patterns, then it should be stored in the extra memory, together with its frequency of occurrence. If there is a pattern during normal operation that occurs more frequently than the patterns stored during learning phase, then this should trigger the system to update its learned data with this frequently occurring pattern.

A node does not only consist of memory, but also has to perform comparisons between newly incoming data and already stored data to determine how similar they are to one another. In the software model, this is performed using Euclidean distance calculation, however this is a computational expensive manner of performing distance calculation. For VLSI implementation, a less computational expensive method would be more beneficial. And since the contents of every memory block needs to be compared, combining it into the memory, like in [6] would be optimal. Before doing this replacement, it is however essential to first determine the accuracy required for this distance calculation. The method in [6] makes use of Manhattan distance calculation, however it is currently not clear how this distance calculation influences the accuracy and performance of HTM.

The hierarchical structure of this system, allows for a globally asynchronous locally synchronous implementation, since only when all data from the lower levels is received, the next level can start computation. Each level however requires approximately the same amount of time for computation. Therefore local clocks can be used to achieve an efficient and fast design. Communication can be performed using asynchronous communication techniques. The communication scheme between the nodes, requires flexibility, especially towards the width of the channels, but the actual source and destination nodes could be linked from the design of the network, due to the hierarchical structure.

4. Conclusions

This paper described Hierarchical Temporal Memory, which is a very promising scheme to deal with real world applications that cannot easily be performed using current algorithms. The scheme has a hierarchical structure, consisting of several nodes. Each of these nodes perform the same algorithm.

The current software model of HTM allows for easy testing and modification of the algorithm, but suffers from performance limitations, especially for larger applications. Furthermore, the software model knows a distinct learning and operating mode. Therefore a possible architecture for VLSI implementation is being proposed. This VLSI architecture is a highly flexible architecture including features like: dynamic reconfiguration. In comparison to the software model, the proposed architecture allows for an extended learning also during normal operation, by using the memory normally not required for the design, to store newly incoming common patterns. In order to find the maximum distance between stored patterns and newly incoming patterns, algorithm optimisations would also be possible, but this requires to determine the required accuracy of the algorithm, which forms part of the current future work.

References

[1] C. Williams, "Expert systems, knowledge engineering, and AI tools - an overview," *IEEE Expert*, vol. 1, no. 4, pp. 66–70, 1986.

[2] S. Kulkarni, G. Lugosi, and S. Venkatesh, "Learning pattern classification - a survey," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2178–2206, 1998.

[3] A. Michel, "Recurrent neural networks: Overview and perspectives," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, vol. III, Bangkok, Thailand, 2003, pp. 474–477.

[4] J. Hawkins and D. George, "Hierarchical temporal memory: Concepts, theory, and terminology," p. 20, 2007. [Online]. Available: http://www.numenta.com/Numenta_HTM_Concepts.pdf

[5] D. George and B. Jaros, "The HTM learning algorithms," p. 44, 2007. [Online]. Available: http://www.numenta.com/for-developers/education/Numenta_HTM_Learning_Algos.pdf

[6] T. Hanyu, S. Kaeriyama, and M. Kameyama, "Logic-in-memory VLSI circuit for fully parallel nearest pattern matching based on floating-gate-MOS pass-transistor logic," *Journal of Multi-Valued Logic & Soft Computing*, vol. 11, pp. 619–632, 2005.