# Introduction

The idea of a “digital first” neural network has a long history starting (perhaps) with John von Neumann’s cellular automata (CA). Cellular Automata are a great starting point since they are very close to what we can make most easily in digital hardware: a homogenous array of simple processing elements with local communication. CA map naturally to a systolic array type computation that keeps all processing elements running at full capacity.

Note, as the Cellular Automata size increases there is more incentive to look at asynchronous communication patterns. Asynchronous dataflow is a great match for computational neuroscience, graphical models and a wide range of applications. Recent Spiking Neural Network accelerators have evolved into dedicated devices for asynchronous packet-based communication.

That said, we suggest there is still plenty of room at the bottom and still many opportunities for machine learning to make better use of the digital systolic machines that we have available. These lower-level synchronous devices are most likely to benefit applications close to streaming high sample rate, high density sensors and data sources.

# Background

The main component in the OHM node, is a bit-serial circuit for Stack Filters introduced by Chen [1]. They describe their circuit as an extension, or generalization, of Successive Approximation analog-to-digital conversion. We try to summarize their narrative here because it's so cool, but check out the paper.

![image](https://github.com/pookaPlay/ohm/assets/17442142/f724bc27-cc0a-4223-be71-662077bc14e3)

The idea behind the successive approximation ADC on the left is to perform a binary search to match the analog input value to an incrementally constructed representation as a binary string. This process is controlled by the successive approximation register and proceeds most significant bit (MSB) first. The analog input signals must be sampled-and-held for the K clock cycles required to produce a K-bit binary encoding. In the middle of the figure is a hybrid analog/digital multi-channel A/D converter and Stack Filter described in [1]. In this case we have multiple analog comparators and instead of using the comparator output directly, the successive approximation register uses the output from a Positive Boolean Function (PBF). On the right, is the all digital version of the idea which is the bit-serial Stack Filter. In this case successive approximation implements the MSB-first binary search for the Stack Filters output value. The red feedback loop contains conditional latching of inputs based on the result so far. The delta symbol components represent "sticky" latches - an important piece of logic in the Stack Filter bit-serial circuit.

At the time of publication, Chen observed that successive approximation A/D converters were the most widely used of all known A/D conversion techniques, providing relatively high speeds with a small circuit size. This still *\*seems\** true today but successive approximation A/D converters are available in large numbers in most Field Programmable Gate Arrays - a 2020 Xilix FPGA had 64 successive approximation ADCs providing 14-bit accuracy at 5G samples/s [don't trust me!].

**### Stack Filters**

Stack Filters are generalizations of the median and other order statistics and provide a useful connection between binary domain signal processing and integer domain signal processing. There is a one-to-one correspondence between the set of Positive Boolean Functions (PBF) and the set of Stack Filters.

![image](images/StackFilter.png)

The figure shows the relationship for a Stack Filter with three inputs 7, 5 and 2. First, we compare each input to a set of monotonically increasing threshold (or quantization) levels. This produces the stacks of 1's on the left where the height of the stack is equivalent to the integer input. Second, we process each slice (threshold level) of the inputs independently using the same PBF. This produces the stack of ones on the right. The final integer domain output is the height (sum) of the output stack which is guaranteed to be one of the inputs. Note, its guaranteed by the restriction to Positive Boolean Functions.

**### Weighted Order Statistics**

In the figure below we show the Stack Filter Integer <-> Binary Domain relationship for the subset of linearly separable positive Boolean functions. This is a linear function with positive weights, applied to binary inputs, and then thresholded, which we also call a positive threshold function (PTF).

![image](images/WosFilter.png)

**## Bit-Serial Implementation**

The figure shows the main components in the OHM Node.

The number of bits in the binary encoding fixes the maximal extents of the data. In experiments we typically start with real-valued data where each dimension has the same extents -3.0 -> 3.0 and we map this to K bits:

$$-2^{k-1}-1 ... 2^{k-1}-1$$

In practice, predicting "extents" (the dynamic range) of data can be hard, and in many cases, the crux of the problem. Data is filled with low-probability outliers and 3 sigma is not enough! Getting these extents right beforehand is particularly important for Analog-to-Digital Conversion at the sensor. In this case, the extents could be set much higher at the cost of underutilizing the MSBs. At the other extreme, reducing the extents too much, means increasing the amount of data that gets clipped.

![image](images/Node1.png)

The inputs to the OHM node are LSB-first two's complement bitstreams. The LSB computation adds a k-bit weight to a k-bit input, and produces a (k+1) bit output, increasing the bit-width of the representation. Not shown in the figure is the fact that each input is in fact duplicated as it enters the OHM node. This duplicate input is multiplied by -1 before entering a second addition circuit. In summary, for D inputs there are 2*\*D additions with 2\**D individual weights. The (k+1)-bit outputs from the 2*\*D additions are "reversed" to provide the 2\**D inputs of the Stack Filter in MSB-first order. Another figure might help:

![image](images/Node2.png)

The MSB computation implements a Stack Filter with 2\*D inputs and produces one of these inputs as an output. Depending on the data, the MSB computation can sometimes identify which is the correct input in fewer than k+1 bits and stop early. The lsb2msb and msb2lsb components handle the "reversing" and are implemented with 2 time multiplexed First-In-Last-Out (FILO) stacks shown in the figure below.

![image](images/Node3.png)

Code for this example is in test\_OHM\_WOS.py

**### Dataflow**

The Bit-Serial implementation of OHM Neural Networks reduces to a very simple data-flow architecture where data bit-streams alternate between least-significant-bit (LSB) and most-significant-bit (MSB) computation.

![image](images/DataFlow1.png)

The figure shows one way to think about and/or implement a network of OHM nodes. In this figure we

put all the local FILO memory stacks in one place and compute the OHM Node outputs in two passes:

1. LSB pass: You can think of the LSB computation as feature extraction, or input embedding. It can increase the number of bits used in the representation. For example, adding two K-bit numbers produces a (K+1)-bit sum. The figure also highlights that there are two different types of data involved in the LSB computation: input data and weights. Our initial work treats these differently (as is done in machine learning) but this distinction may not be necessary. Note, we use the term weights but the parameters we are currently focused on might be better described as bias terms, or offsets.

2. MSB pass: This implements the non-linear part of the OHM node (similar to the ReLU in other neural networks). It implements a Stack Filter which selects one of the inputs as the output. For K-bit inputs, the Stack Filter can sometimes identify which input in fewer than K-bits and stop early. This could also be used to reduce the number of bits in the output representation. For example, if the input selected by the filter was [1 0 1 1 0 1 1 0] and this was determined after only 5 bits, the output would be [1 0 1 1 0] and we throw away the 3 LSBs - the selection filter did not need them for that particular set of inputs.

It is interesting to compare (at a high level) the computational blocks used in the LSB (addition) and MSB (Stack Filter) circuits.

![image](images/NodeComparison.png)

For 2 inputs both circuits can be described by the functional diagram in the top-left. Z is a 1-bit local state and f and g are Boolean functions. For the LSB adder, Z is the carry-bit. For the MSB computation, Z is the "sticky latch" and f is any Positive Boolean Function.

On the right in the figure is what happens when you add more inputs to the MSB computation: the local state is associated with a particular input and each new input adds an additional bit (O(D)). This is different to how the LSB computation scales with fan-in which requires O(logD) additional bits for the adder tree.

**### Weighted Order Statistics**

The figure below shows bit-serial Stack Filters for two different linearly separable Positive Boolean Function (PBF) classes.

![image](images/NodePTF.png)

On the left, each node has a single parameter $thresh$ that controls which order statistic is selected (from the maximum to the minimum).

On the right is a more general class of PBFs defined by introducing $D$ additional parameters (or weights). In the figure we set the original threshold to zero (we ignore it for now).

The picture needs to be fixed: the 'X' multiplication symbol is, in fact, a binary AND (with no internal state) while the $+$ addition symbol represents a bit-serial adder with $1$-bit internal state! The fundamental problem with figure is that it doesn't show what happens with time...

**### Understanding Early Stopping**

The bit-serial Stack Filter of D inputs is applied to binary encoded integers that are K-bits long. For each tick (of the clock) we run the PTF, starting with the MSB. The PTF itself is a linearly separable binary function that could (should) be computed in 1 clock step [\*], but this does get harder to do as D increases. The PTF is a weighted sum with binary inputs which can be implemented with a bit-serial conditional adder tree.

![image](images/MsbLsb.png)

In the figure, the input (and blue circuit) run MSB first, and for each bit, we run a LSB inner loop. We can then count how many clock ticks for different parameter settings to get a better idea of when the Stack Filter can stop early.

| Function | Weights   | Threshold | MSB Ticks | LSB Ticks |

| -------- | --------  | --------- | --------- | --------- |

| MAX      | W=1       | T=1      | 1..D        | log(D)+1    |

| MED      | W=1       | T=D/2    | D           | log(D)+1    |

| MIN      | W=1       | T=D      | 1..D        | log(D)+1    |

| MUX      | W=D       | T=D      | 0           | log(D)+log(D) |

The MUX can be identified in the more general weight setting pretty easily. This could be used for lossless structural pruning of the network. We could also generalize the notion of the MUX to a BI-MUX, TRI-MUX, in which case we look for subsets of inputs for which the minimum weight is larger than the maximum weight of the remaining inputs. With the right threshold, this means the Stack Filter output is guaranteed to be one from the subsets. To analyze/detect this type of structure we use an edge (graph) representation for the Positive Threshold Function. This makes it easy to compute the increasing set of connected components in weight space.

[\*] opportunity for the analog/memristor/unconventional "sub-threshold" device.

**## placeholder**

![image](images/DataFlow2.png)

In this figure we imagine a fully connected set of OHM nodes, where the number of nodes is equal to the input dimension. In the figure, the fully connected layer is operating on the same input embedding.

Another decision to be made is the number (and type) of parameters for the Stack Filter. The figure below shows some linearly separable Stack Filter function classes. On the left is a simple case where we sum up all the inputs and then threshold the sum. The one integer parameter corresponds to the threshold of an Order Statistic. In the top figure, we enumerate this parameter by the position of the Stack Filter in the array. This produces a sorting network. That is, the top node $w\_0=0$ implements a maximum and the bottom node $w\_{N-1}=N-1$ implements a minimum. The sorting network example is implemented in test\_RUN\_SORT\_NETWORK.py

![image](images/DataFlow3.png)

On the right in the figure above is a more general class of PBFs defined by introducing $D$ parameters (or weights). In the figure we set the original threshold to zero (we ignore it for now). Note, that instead of using a single parameter to specify a threshold (as we did on the left) we could use it to select which weight should be non-zero. This corresponds to a multiplexer, and in this case it doesn't really matter if the computation happens in MSB or LSB order.

**### Is the Classifier itself Continuous or Discrete?**

The table gives an OHM centric view of machine learning methods for binary classification. The analogy, which starts with generalizations of the median and how they parallel generalizations of the mean, was stolen from [Arce]. The analogy really only applies to real-valued Binary classification problems:  problems where the x inputs are real-valued and the y output is one of two classes {0,1}. For these machine learning problems, classifiers typically fall into one of two types:

1. Continuous classifiers use a real-valued function and then threshold the function to produce a class output.

2. Discrete classifiers use some method to partition the input space and then store, or assign, class labels to each partition.

As per usual with discussions of continuous and discrete things, it's never quite that simple and the approaches are closely related. Successful algorithms often use a combination of both approaches to mitigate different sources of error e.g. random forest type classifiers.

![image](images/DiscreteClassifiers.png)

**## Related Work**

An incomplete list of some of the related work.

**#### Morphological Networks and Tropical Geometry**

Most relevant are Morphological Neural Networks, Min-Max Neural Networks and other methods that came out of non-linear digital signal processing. Tropical geometry (sub-field of “algebraic geometry”) could be a more modern/general view of the lattice and set theory math that was used in non-linear signal processing:

\* [Maragos review paper](https://ieeexplore.ieee.org/abstract/document/9394420) and [talk on the topic](https://www.youtube.com/watch?v=mDheYMtRM28).

\* [Application to ReLU Deep Nets](https://proceedings.mlr.press/v80/zhang18i/zhang18i.pdf)

**#### Binary (or Binarized) Neural Networks**

This is a shorter-term, very active area of research: take existing foundation models and compress them to reduce computation. Maragos et. al. show promising results using tropical geometry for this problem.

**#### Stochastic Bit Stream Neural Networks and Spiking models**

Very similar in spirit to OHM, but typically start with an inefficient representation i.e. O(N) (in time) instead of the O(logN) achieved with binary encoding.

**#### EEXIST and the Cell Matrix**

[EEXIST](https://songlinesystems.com/) focuses on a continuous space and time model. But from a Neural Network perspective it also has an inspirational unsupervised adaptation rule!

**#### Recent and Random**

\* [KAN Networks](https://arxiv.org/html/2404.19756v1) also focus on learning the nonlinearity with fixed summation as the only multi-input operation.

**# References**

[Weighted order statistic classifiers with large rank-order margin](https://github.com/pookaPlay/ohm/blob/main/doc/pubs/wosc.pdf), R. Porter, D. Hush, D. Eads, J. Theiler, International Conference on Machine Learning, ICML-2003.

[Stack Filter Classifiers](https://github.com/pookaPlay/ohm/blob/main/doc/pubs/OhmTechReport.pdf), R. Porter, B.G. Zimmer and D. Hush, International Symposium on Mathematical Morphology. 2009.

[Ordered Hypothesis Machines](https://github.com/pookaPlay/ohm/blob/main/doc/pubs/OhmJMIV.pdf), B.G. Zimmer, D. Hush, and R. Porter, Journal of Mathematical Image and Vision, DOI:10.1007/s10851-011-0293-z, 2011.

[Error Minimizing Algorithms for Nearest Neighbor Classifiers](https://github.com/pookaPlay/ohm/blob/main/doc/pubs/OhmNN.pdf), R. Porter, D. Hush, and B.G. Zimmer. in Proceedings of the SPIE. San Francisco. 2011.

[Links Between Binary Classification and the Assignment Problem in Ordered Hypothesis Machines](https://github.com/pookaPlay/ohm/blob/main/doc/pubs/OhmAssignment.pdf), R. Porter, B. G. Zimmer, Proceedings of SPIE, Image Processing: Machine Vision Applications VIII, 2014.

[1] Bit-Serial Realizations of a Class of Nonlinear Filters Based on Positive Boolean Functions, Chen K, IEEE Transactions on Circuits and Systems, 36: 785-794, 1989.