Implementing Stochastic Hopfield-Network-based Linear Solvers on a Hardware-Constrained Neural Substrate

Erik Jorgensen, Rohit Shukla, and Mikko Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Email: ejorgensen2@wisc.edu, rshukla3@wisc.edu, and lipasti@engr.wisc.edu

***Abstract* - IBM's TrueNorth neurosynaptic system provides an appealing platform for deploying numerical algorithms for ultra-low power, real time, and mobile applications. A recurrent Hopfield neural network is used to solve for the Moore-Penrose matrix pseudoinverse to solve a broad class of linear optimizations. The TrueNorth hardware platform is heavily constrained through weight quantization and severely limits range and precision of numerical representation and computation. We show that a flexible, robust, and real-time implementation of an optical flow algorithm can be deployed on TrueNorth with minimal resource allocation and high energy-efficiency. These results show promising potential for TrueNorth as an ultra-low power generalize matrix inverse calculator.**

I. INTRODUCTION

Spiking neural network substrates such as IBM's TrueNorth neurosynaptic processor offer an ultra-low power platform on which to solve a host of numerical problems. Two compelling real-time visual problems are object tracking and optical flow. Computer vision algorithms can be used to track invariant features across different variations in position, scale, and rotation. By computing invariant transforms that map the objects in an image to the known objects in memory, objects and their motion between images can be tracked [1], [2], [3], [4], and [6]. This real-time tracking and motion capture can be useful for traffic monitoring, drone maneuvering, autonomous driving, or many other applications. Systems can benefit from the ability to track an object's motion while offshoring the computation locally to sensors to reduce power consumption of the main processor. Unfortunately, the matrix inverse computations required to map motion between images can be very expensive for a normal CPU. The ability to make computations local to sensors with an ultra-low power neurosynaptic processor could provide a less power-hungry solution for mobile applications.

Substrates such as TrueNorth are promising low-power processors [5], but require careful design mapping to ensure numerical accuracy. A major strength of the TrueNorth chip is its ultra-low power consumption – on average about 100mW. Constrained neuron weights help TrueNorth reduce power consumption, but pose a challenge for mitigating computation error. Additionally, the requirement of inputting stochastic bit-streams for numerical representation inherently introduces time-varying precision. We present a practical implementation of the Hopfield neural-network on the TrueNorth substrate used as a linear solver for object localization and optical flow.

II. IBM TRUENORTH

IBM's Neurosynaptic System "TrueNorth" is a spiking neural network consisting of 4096 cores; each containing 256 input axons and 256 output neurons that have individually configurable synapse connections. These axons and neurons are connected across a 256 x 256 crossbar of 65536 synapse connections. The 1 million individual neurons integrate and fire as determined by their more than 20 programmable features such as threshold, leak rate, and reset. TrueNorth operates at a rate of 1 KHz, updating the state of the neurons update every millisecond, or 'tick'. A major distinction from floating-point processors is the time-variation in numerical representation. A stochastic rate-coded stream of spikes is used to represent numbers between zero and one. With a processor speed of 1 KHz, an input or output of 1000 spikes in a second represents the value 1, whereas 500 spikes in the same period represents 0.5, and so on. This coding scheme states that the probability of a spike occurring at time *t* is the same as the input value. As time passes, the precision of a numerical representation increases proportionally. Numbers are represented more precisely over the span of 10 seconds (10,000 ticks) rather than 1 second (1000 ticks). Additionally, a population encoding approach can be used to increase the precision of numerical representation. By grouping a set of neurons to represent a single number, the same or higher precision can be attained over a shorter time span than by using a single neuron.

The membrane potential of each neuron is updated every tick *t* as is governed by the following equations.

$$V(t) = V(t-1) + \sum_{i=0}^{255} A_i(t) w_i s^{G_i} \qquad (1)$$

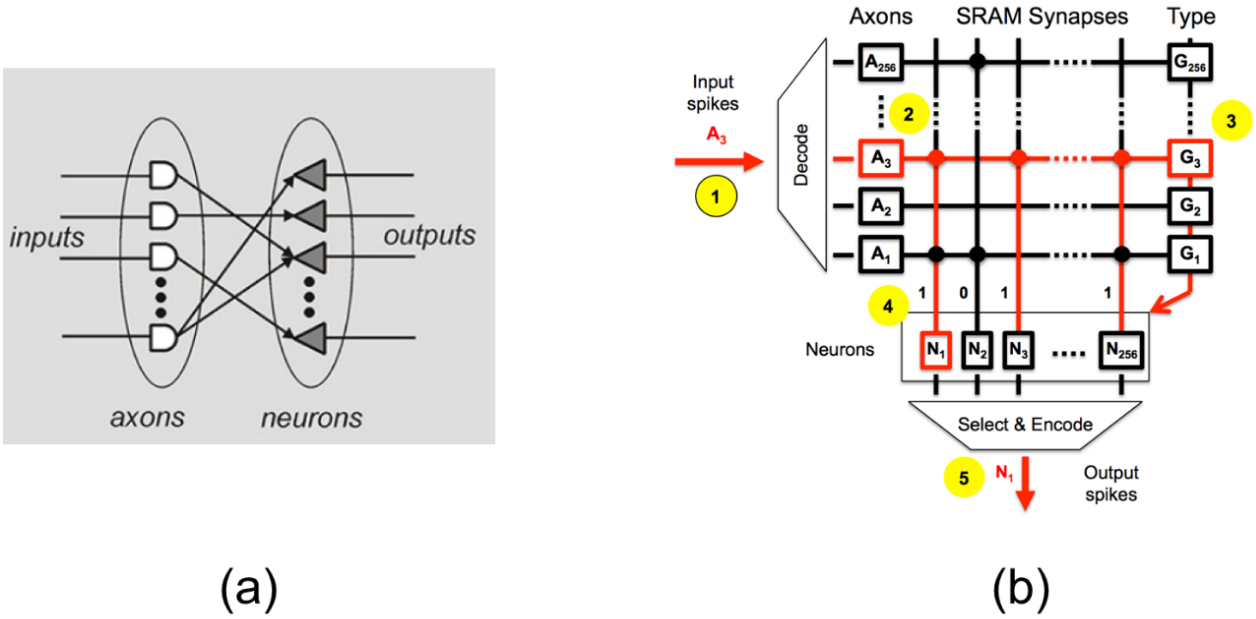

Fig. 1 Abstractions of TrueNorth showing, (a) Axons serving as inputs to and neurons as outputs from the core. (b) Synaptic connections are programmable on a 256 x 256 crossbar with weight values associated with each connection.

$$V(t) = V(t) + \lambda \tag{2}$$

$$if\ V(t) < 0 :\ V(t) \leftarrow 0$$
$$if\ V(t) \geq \alpha : Spike\ and\ set\ V(t) \leftarrow V(t) - \alpha \tag{3}$$

Equation 1 defines the integration of the neuron membrane potential at after each tick *t*. The neuron integrates the binary spike input $A_i(t)$ from 256 axons. The synaptic connection defined for each of the 256 axons connected to the neuron on the crossbar is represented by the binary-valued w_i. The synaptic weight term is s^{G_i} where G_i signifies which of the four axon types is used. Each core with 256 axons can assign as many as four unique axon types. Each assigned axon type corresponds to a 9-bit signed integer weight with the first bit being a sign bit, taking values from -255 to 255.

Equation 2 integrates the neuron's leak λ to the neurons membrane potential $V(t)$. This biologically-inspired leak parameter reduces the neuron potential after each time step without receiving an input spike. For less biologically-realistic application, a positive leak can increase the membrane potential over time. Finally, equation 3 is used to maintain a positive membrane potential when the neuron is in the linear reset mode. If $V(t)$ meets or exceeds the threshold potential α, the neuron outputs a spike and the membrane potential is subtracted by α. The 18-bit unsigned threshold of TrueNorth neurons can take on values from 0 to 262143, affording significantly high precision in certain situations.

By ensuring stochastic numerical representation, many simple logical operations can be used to manipulate the numbers represented by spike streams mathematically. A pair of stochastic spike streams can be added or multiplied using combinational logic [10]. These operations are represented by taking advantage of TrueNorth's neuron parameters to represent OR or AND combinational logic respectively with only a single neuron. By inputting streams of stochastically generated spikes (synaptic impulses), these configured neurons can compute addition, subtraction, multiplication, and other functions with time-dependent precision. As we show in the next section, the solution to the Moore-Penrose pseudoinverse of a matrix can be solved iteratively on TrueNorth with these simple functions.

III. HOPFIELD NEURAL NETWORK SOLUTION TO MOORE-PENROSE PSEUDOINVERSE

By arranging sets of TrueNorth's neurons, we can implement a recurrent Hopfield network to solve for the Moore-Penrose pseudoinverse of a matrix [7]. This pseudoinverse is useful when solving for the transformation matrix X that maps the elements of matrix A to the elements of matrix B as in equation 4. In the case of object tracking, matrix A would hold the initial features and matrix B would represent the new set of observed features.

$$AX = B \tag{4}$$

$$X = A^{-1}B \tag{5}$$

The solution matrix X to equation 4 is ordinarily computed by inverting matrix A and multiplying with matrix B as in equation 5. However, many situations arise in which the $m \times n$ matrix A has no definite inverse; most commonly when A is not square or if $rank(A) < m$. In such situations, a pseudoinverse is computed as a least squares solution to the linear system. Conveniently, the pseudoinverse (as represented by the † symbol) is defined and unique for all matrices and is shown in equation 6. This is due to the fact that $A^T A$ is always square and invertible.

$$X = A^{\dagger}B = (A^T A)^{-1}A^T B \tag{6}$$

This left-inverse computation is used in the applications presented in this paper, however the Hopfield neural network architecture can also compute a right-inverse of matrix A if the row rank was greater than the column rank. The iterative approach to solving this equation with the Hopfield neural network using the pseudoinverse of matrix A is given in equation 7 and modeled in Figure 2 [7], [8].

$$X_{k+1} = (I_n - \alpha A^T A)X_k + \alpha A^T B \tag{7}$$

Or more simply in equation 8:

$$X_{k+1} = W_{hop}X_k + W_{ff}B \tag{8}$$

The Hopfield neural network model of a linear equation solver can be solved by assembling the following:

1) A feedforward layer with input matrix B and weight matrix $W_{ff} = \alpha A^T$.
2) A recurrent layer with weight matrix $W_{hop} = (I_n - \alpha A^T A)$ whose inputs are the outputs of the feedforward layer.
3) The term α ensuring convergence, determines the rate of convergence, and defined in equation 9.

$$0 < \alpha < \left(\frac{2}{trace(A^T A)}\right) \tag{9}$$

Thus, the iterative solution to update the value for X_{k+1} is given in equation 10.

$$X_{K+1} = \sum_{i=0}^{k+1} (W_{hop})^i W_{ff}B \tag{10}$$

Where W_{hop} is always a symmetric matrix with spectral radius less than 1. Because of these properties, every element of W_{hop} will always be less than 1 in absolute value - as needed for stochastic computation [9].

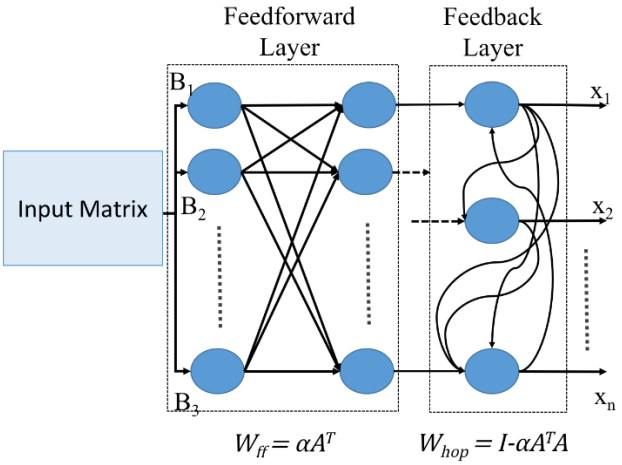

Fig. 2 Depiction of Hopfield neural network connections taking input matrix B, and outputting matrix X.

IV. HOPFIELD NEURAL NETWORK ON TRUENORTH

To perform the matrix multiplications of W_{ff} with B and W_{hop} with X_k, one method is to hard-code the floating-point values of the two weight matrices as a ratio of weights to neuron thresholds. To ensure numerical accuracy within the 8-bit range of possible weights allowed by TrueNorth, the weights must be scaled within the range of 0 to 255. This is done by mapping all the W_{ff} or W_{hop} values to a range from 0 to 255. The threshold must be set by mapping the maximum of the weight matrix proportionally. In example: a W_{hop} with a maximum value of 0.5 would be mapped to a threshold of 510 and weights between 0 and 255. That way, the neuron would only output a spike after receiving at least two input spikes – each increasing the membrane potential by up to 255.

Figure 3 shows the synaptic connections in TrueNorth that compute a single dot product of the matrix multiplication of the 3×3 Hopfield network input H_k with the 3×3 weight matrix W_{hop}. Each of the three values in H_k are assigned a different axon type so that they multiply with the corresponding weight from W_{hop}.

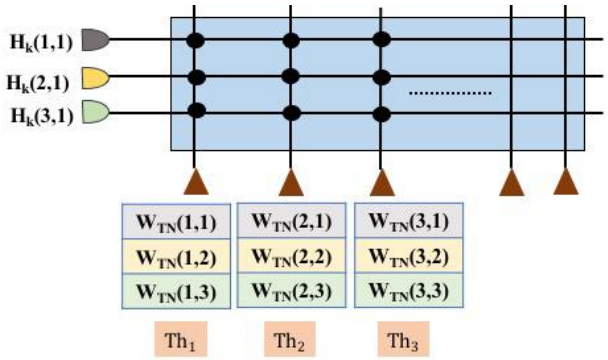

Fig. 3 Synapse connections on a TrueNorth crossbar showing the dot product of the first column of H_k with the weight matrix W_{hop}.

The crossbar architecture of a single 3×3 matrix multiplication is depicted in Figure 4. In this case, each synapse connection computes one product of the 27 in the 3×3 matrix multiplication, and each set of three synapses is integrated down one of the nine neurons. It is important to again note that numbers can only be represented between 0 and 1, so negative values in the W_{hop} matrix must be dealt with separately. In this case, we must first separate values into 3×3 positive and 3×3 negative domains. In the positive case, all negative numbers of the 3×3 matrix are zeroed and the opposite is true for the negative case. Then we compute each 3×3 matrix multiplication separately but with the same inputs. We now have two 3×3 matrices with positive values, but the second represents the magnitude of the negative values.

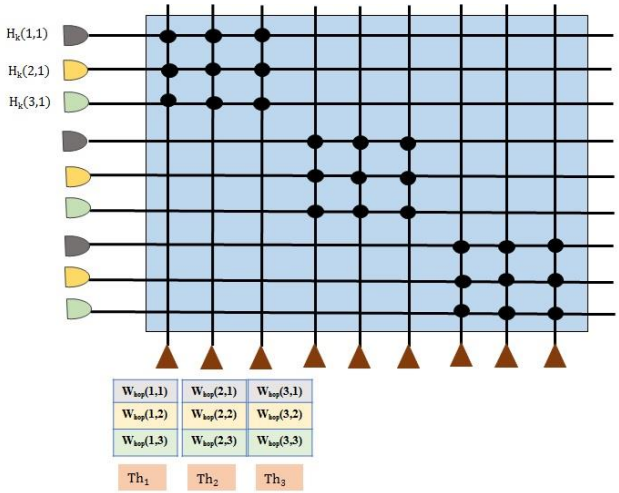

Fig. 4 Synapse connections on a TrueNorth core crossbar used to compute an entire 3×3 matrix multiplication. Inputs to the axons on the left represent the input H_k, while the neuron weights represent W_{hop} or W_{ff}.

This implementation holds for static A matrix values, but must be redrawn more flexibly when solving a more elaborate problem with multiple different A matrices such as in the case of optical flow. While the simple practice of adding together multiple rate coded spike trains is well-defined on TrueNorth, multiplication of two spike trains is necessary for a flexible implementation. By setting a TrueNorth neuron's parameters appropriately they can act as multipliers. This takes advantage of the stochastic nature of the input spike trains by simply operating as a logical AND. On average, a stochastic rate coded spike train that represent the number 0.5 will correlate to another spike train representing the number 0.6 with a rate of 0.3. As can be expected, the precision of this method would converge more slowly. This can be attributed to the fact that each stochastic spike train more precisely represent a number as more time steps pass, resulting in an increase in precision over time of this multiplication.

The previous method outperforms this method in convergence speed. In the prior method, products were determined by multiplying input spike trains with definite weights whereas this method varies precision based on two independent input spike trains. Additionally, the resource utilization is much higher in the second method since a single neuron is needed for each multiplication in a dot product as well as another to add those products together. In the 3×3 case, the first method needs only one neuron and three axons per dot product, where the second method requires four neurons and twice as many axons. Even more, we input two rate coded spike trains per multiplication, but each matrix being multiplied can have positive and negative values. When we split each matrix into positive and negative domains to be represented stochastically, we are now more than quadruple the number of computations necessary. Matrix multiplication like that of W_{hop} with X_k is separated into four parts: $W_{hop\_pos}X_{k\_pos}$, $W_{hop\_pos}X_{k\_neg}$, $W_{hop\_neg}X_{k\_pos}$, and $W_{hop\_neg}X_{k\_neg}$. We then manipulate these four domains arithmetically to output the overall positive and negative domains of $W_{hop}X_k$. All of these domain and arithmetic combinations result in a much greater total resource utilization of TrueNorth neurons.

It is important to realize, however, that the second implementation allows for flexibility by allowing any two matrices as input without reconfiguring neuron weights. The first implementation would require the user to manually hard-code weights for one of the matrices. Withstanding the losses in computation time efficiency and resource utilization, this second method is preferred in many cases for its generalized nature.

V. OPTICAL FLOW EXPERIMENTAL SETUP

A popular algorithm in applications like drone control and object tracking is optical flow. This algorithm uses the Jacobian between pixel intensities in subsequent image frames. Across short time intervals, it is assumed that only small changes in object location occur and that overall intensity of an image does not change. The optical flow algorithm takes as input two subsequent image frames and

outputs the direction and speed at which an object is moving. It is common to break up each image into a grid of smaller windows on which to operate the algorithm. This allows the algorithm to track multiple objects moving at different velocities.

$$A = \begin{bmatrix} I_x(q1) & I_y(q1) \\ \vdots & \vdots \\ I_x(qn) & I_y(qn) \end{bmatrix} \quad (11)$$

$$B = \begin{bmatrix} -I_t(q1) \\ \vdots \\ -I_t(qn) \end{bmatrix} \quad (12)$$

In our experiment, we implemented the Lucas-Kanade algorithm of optical flow. As inputs, we assemble matrices A and B from $I_x(qi)$, $I_y(qi)$, and $I_t(qi)$ which represent the derivatives across the x direction, y direction, and time respectively around pixel qi. These matrices are assembled as shown in equations 11 and 12. In our experiments, we computed the optical flow vector on 5×5 windows of pixels – resulting in a matrix A of size 25×2 and matrix B of size 25×1. We again solve the set of linear equations given in equation 4 by using the pseudoinverse, with matrix X representing the output optical flow vector. In this case, rather than outputting a 3×3 affine transformation mapping as in the object tracking case [12] and [13], we simply return a 2-element vector accounting for the velocity in the x direction and y direction.

Our experiment operated on a short set of frames involving two bars in a blank frame – one moving horizontally and the other moving vertically similar to that in [13]. The size of the frames was 50 cm by 50 cm and the lines were each 3 cm thick. The bars both moved at 12 cm per second in their respective directions. This experiment is visualized in the resultant screenshot in figure 5.

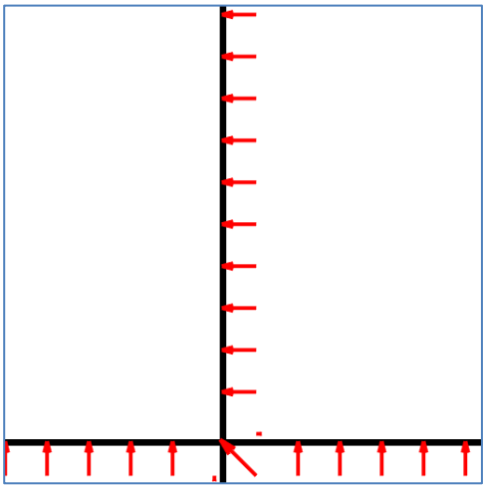

Fig. 5 Optical flow vector field showing directional movement of horizontal and vertical lines.

VI. RESULTS AND DISCUSSION

In the optical flow experiment our implementation used only 712 neurons out of the total 1 million neurons available on TrueNorth. This architecture uses only 11 or 0.2% of TrueNorth's available cores. We compute relative and absolute error between the TrueNorth Hopfield linear solver output and the output obtained using MATLAB's pseudoinverse function. We note a preference toward lower absolute error because it indicates the precision of our result compared to the predicted result.

Our optical flow application which uses dynamic matrix multiplication shows slower convergence rates than when using definite neuron weights, much as expected. Table 1 summarizes the errors obtained when simulating for 3 million clock ticks for 100 randomly chosen values from two image frames. We also ensured that the ideal output would not have both velocity components equal to zero to avoid incorrect results when no motion is present. We note that our results were always correct in terms of direction of motion or velocity and the correct polarity converged rather quickly.

TABLE 1
ERROR IN REPORTING OPTICAL FLOW VECTORS

Attribute	Mean relative Error (%)	Standard deviation of relative error (%)	Mean absolute error (cm)	Standard deviation of absolute error (cm)
Magnitude of horizontal velocity	18.39	36.56	0.1649	0.5485
Magnitude of vertical velocity	7.65	18.27	0.2057	1.0

VII. CONCLUSIONS AND FUTURE WORK

This work, to the best of our knowledge, is the first to implement a generalized matrix pseudoinverse calculator with a precision-constrained neural substrate. Our results show the plausibility of using a linear solver on-chip, rather than offloading the computations to the power-hungry CPU operations. We find that these algorithms as implemented on TrueNorth are accurate but take a relatively long time to converge to usefully-precise solution.

It is noted, that these algorithms would likely perform very well on neural substrates or FPGA that operate at a much higher frequency than the 1 KHz of IBM's TrueNorth. We suppose that power savings would still be significant at a higher operating frequency due to the incredibly low resource utilization of the neural substrate. In addition, slow convergence could be mitigated by using less conservative scaling factors to keep computations from saturating on the [0,1] range of stochastically representable values.

Furthermore, gains in computation speed can be had by implementing a population coding scheme for encoding values into spike trains. Separating values into multiple streams would allow us to implement parallel computations with a result of increased precision in the same time range.

REFERENCES

[1] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," Nature Neuroscience, vol. 2, pp. 1019–1025, 1999.

[2] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," in Biological Cybernetics, vol. 36, 1980, pp. 193–202.

[3] D. W. Arathorn, Map-Seeking Circuits in Visual Cognition: A Computational Mechanism for Biological and Machine Vision. Stanford, CA, USA: Stanford University Press, 2002.

[4] E. Rolls, "Invariant visual object and face recognition: Neural and computational bases, and a model, visnet," Frontiers in Computational Neuroscience, vol. 35, Jun 2012.

[5] P. A. Merolla, J. V. Arthur, F. Akopyan, and et. al., "A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm," in IEEE Custom Integrated Circuits Conference, CICC 2011, September 2011.

[6] Y. Mroueh, S. Voinea, and T. Poggio, "Learning with group invariant features: A kernel perspective," in Advances in Neural Information Processing Symposium, 2015. [Online]. Available: http://arxiv.org/abs/1311.4158

[7] G. Lendaris, K. Mathia, and R. Saeks, "Linear hopfield networks and constrained optimization." IEEE Trans. Systems, Man and Cybernetics, Part B, vol. 91, pp. 114–118, Feb 1999.

[8] A. Ben-Israel and A. Charnes, "Contributions to the theory of generalized inverses," J. Soc. Indust. Appl. Math., vol. 11, no. 3, pp. 55–60, 1963.

[9] P. J. Olver, "Numerical analysis lecture notes," 2016. [Online]. Available: http://quant-econ.net/ downloads/iteration notes.pdf

[10] A. Cassidy, P. Merolla, J. V. Arthur, and et. al., "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," 2013.

[11] "https://en.wikipedia.org/wiki/Affine transformation," 2016. [Online]. Available: https://en.wikipedia.org/wiki/Affine transformation

[12] R. Shukla and M. Lipasti, "A Self-Learning Map-Seeking Circuit For Visual Object Recognition," in The International Joint Conference on Neural Networks, IJCNN 2015, 2015.

[13] S. Esser, A. Andreopoulos, R. Appuswamy, and et. al., "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," 2013.