frontiers in Neuroscience

ORIGINAL RESEARCH
published: 13 March 2018
doi: 10.3389/fnins.2018.00115

Computing Generalized Matrix Inverse on Spiking Neural Substrate

Rohit Shukla[1], Soroosh Khoram[1], Erik Jorgensen[2], Jing Li[1], Mikko Lipasti[1] and Stephen Wright[3]*

[1] Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI, United States,
[2] Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, United States,
[3] Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, United States

Emerging neural hardware substrates, such as IBM's TrueNorth Neurosynaptic System, can provide an appealing platform for deploying numerical algorithms. For example, a recurrent Hopfield neural network can be used to find the Moore-Penrose generalized inverse of a matrix, thus enabling a broad class of linear optimizations to be solved efficiently, at low energy cost. However, deploying numerical algorithms on hardware platforms that severely limit the range and precision of representation for numeric quantities can be quite challenging. This paper discusses these challenges and proposes a rigorous mathematical framework for reasoning about range and precision on such substrates. The paper derives techniques for normalizing inputs and properly quantizing synaptic weights originating from arbitrary systems of linear equations, so that solvers for those systems can be implemented in a provably correct manner on hardware-constrained neural substrates. The analytical model is empirically validated on the IBM TrueNorth platform, and results show that the guarantees provided by the framework for range and precision hold under experimental conditions. Experiments with optical flow demonstrate the energy benefits of deploying a reduced-precision and energy-efficient generalized matrix inverse engine on the IBM TrueNorth platform, reflecting 10× to 100× improvement over FPGA and ARM core baselines.

Keywords: spiking neural networks, TrueNorth, matrix inversion, Hopfield neural network, neuromorphic computing, stochastic computing

OPEN ACCESS

Edited by:
Arindam Basu,
Nanyang Technological University, Singapore

Reviewed by:
Guillaume Garreau,
IBM Research Almaden, United States
Subhrajit Roy,
IBM Research, Australia

****Correspondence:***
Rohit Shukla
rshukla3@wisc.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *02 December 2017*
Accepted: *13 February 2018*
Published: *13 March 2018*

Citation:
Shukla R, Khoram S, Jorgensen E, Li J, Lipasti M and Wright S (2018) Computing Generalized Matrix Inverse on Spiking Neural Substrate. Front. Neurosci. 12:115. doi: 10.3389/fnins.2018.00115

1. INTRODUCTION

Recent advances in neuromorphic engineering (Schuman et al., 2017) have motivated the development of neural hardware substrates that are tailored to loosely emulate computations that happen in a human brain with extremely low power and efficiency. Examples include IBM TrueNorth Neurosynaptic System (Merolla et al., 2014), NeuroFlow (Cheung et al., 2016), Neurogrid (Benjamin et al., 2014), SpiNNaker (Furber et al., 2014), and the BrainScaleS project (Schemmel et al., 2008), all of which are implemented using Si CMOS. While Si CMOS is the prevailing technology, the slowdown in transistor scaling has led to broad interest in spiking neural network substrates that exploit the unique properties of emerging nonvolatile memory such as Narayanan et al. (2017) and RRAM (Goux et al., 2013). Due to the close match between the algorithmic requirements and the underlying hardware architecture, such designs have the potential to achieve much better computational efficiency than the conventional Si-CMOS based designs.

In spite of the radically differing hardware implementations of these neural network substrates, many of them share an inherent design principle: converting input signal amplitude information into a rate-coded spike train and performing parallel operations of dot-product computations on these spike trains, based on synaptic weights stored in the memory array. These similarities also result in a set of common challenges during practical implementation, especially when using them as computing substrates for applications with a mathematical algorithmic basis. These challenges include a restricted range of input values and the limited precision of synaptic weights and inputs. Since a value is encoded in unary spikes over time (i.e., as a firing rate), each individual input and variable must take a value in the range $[0, 1]$. Furthermore, the precision of the encoded value is directly proportional to the size of the evaluation window, which, for reasons of efficiency, is typically limited to a few hundred spikes. Finally, because of hardware cost, synaptic weights can be implemented only by a limited number of memory bits, resulting in limited precision. For instance, IBM's TrueNorth supports 9-bit weight values, where most significant indicates sign of the weights.

Mapping existing algorithms to these substrates requires the designer to choose a strategy for quantizing inputs and weights carefully, so that the range limitations are not violated (i.e., values represented by firing rates do not saturate), while maintaining sufficient precision. Prior work notes these challenges, but typically presents only *ad hoc* solutions that choose scaling factors and quantization strategies based on empirical measurements that can guarantee correct operation for the tested scenarios, but provides no guarantees in the general case (Jin et al., 2008; Shukla et al., 2017). Error analysis for feedforward networks appear in Hopkins and Furber (2015), but omits recurrent networks and range analysis.

In contrast, this paper develops a rigorous mathematical model that enables a designer to map numerical algorithms to these substrates and to reason quantitatively about the range and precision of the computation taking place in the neural substrate. Our mathematical framework can be applied to a wide range of problems in linear optimization running on neural substrates with diverse constraints. The model is validated empirically by constructing input matrices with random values and computing matrix inverse using a recurrent Hopfield neural-network-based linear solver. Our results show that the scaling factor and error bounds derived by the mathematical model hold for this application under a broad range of input conditions. We report the computing resources and power numbers for real-time applications, and quantify how the errors and inefficiencies can be addressed to enable practical deployment of the Hopfield linear solver.

Prior work by authors in Shukla et al. (2017) showed how these linear solvers can be used in a variety of robotic applications, such as computing transformation matrices. They have discussed how weights can be encoded on TrueNorth, as in section 2.3.1 of this paper, and have reported experiments using a weight-encoded scheme. However, Shukla et al. (2017) does not show any mathematical model of the Hopfield linear solver (section 2.2 of this paper) and does not contain results about error bounds (section 2.5). There was no discussion about Hopfield weights being encoded as spiking inputs (section 2.3.2) and no description regarding algorithmic steps required to implement the linear solver on a spiking neural substrate like TrueNorth (section 2.4). All these issues are addressed in this paper. Additionally, section 3 of this paper presents a more thorough analysis of experiments with respect to dynamic inputs, where the Hopfield network weights are represented as spiking inputs.

2. MATERIALS AND METHODS

2.1. Background

This section describes how a system of linear equations can be solved using a recurrent Hopfield neural network, and shows how such a solver can be used in applications such as target tracking and optical flow. These example applications have been successfully deployed on TrueNorth (Shukla et al., 2017), by applying the analytical framework presented in this paper to probe its range and precision requirements.

2.1.1. Solving Linear Systems With a Hopfield Network

A linear equations solver is used to solve matrix equations of the form $AX = B$. More generally, to accommodate the case of infeasible systems, we obtain X from the following linear least squares problem:

$$\min_X \frac{1}{2}\|AX - B\|_F^2, \tag{1}$$

where A is an $M \times N$ matrix with $M \geq N$, B is a matrix of dimension $M \times P$, and $\|\cdot\|_F$ is the Frobenius norm. Note that the problem decomposes by columns of B, so we can write Equation (1) equivalently as

$$\min_{[X]_{\cdot j}} \frac{1}{2}\|A[X]_{\cdot j} - B_{\cdot j}\|_2^2, \quad j = 1, 2, \ldots, P, \tag{2}$$

where $[X]_{\cdot j}$ denotes the jth column of the matrix X. By setting the the gradient of Equation (1) to zero, we obtain the following "normal equations:"

$$A^T AX = A^T B. \tag{3}$$

This system can be solved by the following stationary iterative process:

$$X_{k+1} = X_k + \alpha(-A^T AX_k + A^T B) \tag{4}$$

$$= (I - \alpha A^T A)X_k + \alpha A^T B, \tag{5}$$

$$\text{where} \qquad X_0 \coloneqq \alpha A^T B, \tag{6}$$

and α is a positive steplength. (This process can also be thought of as a steepest descent method applied to the optimization problem Equation (2). For convergence of this process, we require

$$0 < \alpha < \frac{2}{\lambda_{\max}(A^T A)}, \tag{7}$$

where $\lambda_{\max}(A^TA)$ denotes the maximum eigenvalue of A^TA. This condition ensures that all eigenvalues of $(I - \alpha A^TA)$ lie in the interval $(-1, 1]$. (If A is rank deficient, A^TA is singular, so some eigenvalues of $(I - \alpha A^TA)$ will be 1 in this case.) See Ben-Israel and Charnes (1963) for a proof of convergence.

We can map this process Equation (4) to a recurrent Hopfield network cleanly by rewriting it as follows:

$$X_{k+1} = W_{\text{hop}}X_k + W_{\text{ff}}B, \quad k = 0, 1, 2, \ldots, \tag{8}$$

where

$$W_{\text{hop}} \coloneqq I - \alpha A^TA, \tag{9a}$$

$$W_{\text{ff}} \coloneqq \alpha A^T. \tag{9b}$$

The Hopfield neural network architecture for implementing Equation (8) is shown in **Figure 1**.

This elementary derivation shows that we can solve arbitrary systems of linear equations (which we refer to also as "matrix division") directly in a recurrent neural network by loading synaptic weight coefficients W_{ff} and W_{hop} derived from A and α into the neural network, and connecting the inputs b and recurrent outputs X_{k+1} appropriately. The weight matrix W_{ff} serves as the feedforward weight for the input matrix B, while W_{hop} serves as the weight for the recurrent part for the values X_k.

We have implemented prototypes for two different classes of applications. The first prototype is for applications in which Hopfield network weights are hard-coded on TrueNorth, while the second prototype is for applications in which Hopfield network weights are encoded as dynamic spike trains.

2.1.2. Hopfield Network Weights Hard-Coded on TrueNorth

For the first class of applications, we consider a typical target tracking scenario, shown in **Figure 2A**. We are repeating the same steps that was published in Shukla et al. (2017) to replicate the target tracking experiment. Here, the features do not change often, remaining fixed for long time periods, during which the matrix A in Equation (3) stays the same. The Hopfield network weights W_{ff} and W_{hop} can be precomputed and hard-coded onto TrueNorth board for use.

In target tracking, a real-time video input is preprocessed to extract features (e.g., edges of particular orientations) to form a feature set. This feature set is then compared against a set of templates to identify objects of interest, with the goal of tracking the objects in the image frame as they move in three dimensions. As a proof of concept, we have chosen a very simple image whose feature set consists of just three edges similar in appearance to the letter H. To determine size and placement of the bounding box for the tracked image, we utilize the theory of affine transforms which shows that a current image B can be matched to its template A via an affine transformation X using a matrix multiplication $AX = B$, as long as the image has been transformed only with respect to the template in scale, rotation, or 2D translation. (A similar self-learning visual architecture was investigated in Shukla and Lipasti, 2015). By employing matrix division implemented in the recurrent Hopfield network

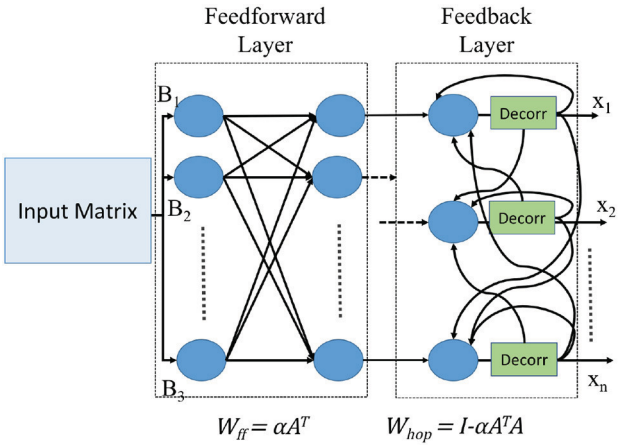

FIGURE 1 | Neural network architecture of Hopfield linear solver.

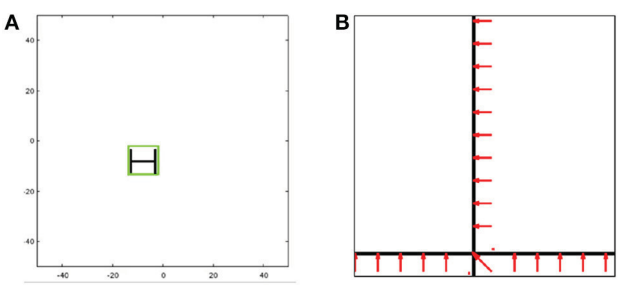

FIGURE 2 | Illustration of **(A)** target tracking and **(B)** optical flow.

(Equation 8), we can derive the affine transform X that maps the current image input b to the template A, thus determining the scale and the horizontal and vertical transformations from the matrix X.

2.1.3. Hopfield Network Weights as Dynamic Spiking Input

We investigated optical flow as our second application, shown in **Figure 2B**. We are repeating the same steps that was published in Shukla et al. (2017) to replicate the optical flow experiment. In optical flow, the matrix A might change at certain (frequent and regular) time intervals, so it is not viable to hard-code weight assignment on TrueNorth. To implement a more dynamic pseudoinverse calculator, we make use of concepts from stochastic computing (Gaines, 1967; Alaghi and Hayes, 2013) and demonstrate the versatility of spiking neural substrates.

Our demonstration is similar to the one reported in Esser et al. (2013) where the horizontal bar is continuously moving upwards and the vertical bars are moving to the left of the screen. In this prototype, we demonstrate that the direction of movement of the bars can be reported without any error, and the speed at which the two bars move apart can be determined approximately. This is done by solving for X in the equation $AX = B$, where the matrix A contains partial derivatives of initial image frame with respect to x and y directions, and the vector B contains partial derivatives

of pixel positions between the initial image frame and the image frame at time t. The solution matrix X indicates the speed and direction of the image pixels.

2.1.4. Computing in Spikes

IBM TrueNorth is a biologically-inspired architecture that performs computations using spiking neurons. Input values are represented in a stochastic time-based coding, in which the probability of occurrence of a spike at a particular time tick is directly proportional to the input value. Since the computation values are represented as spike trains, designers are faced with two key issues in mapping algorithms to these spiking neural substrate.

1. Signed computations on spiking neural network substrates that have input values represented as rate based encoding must be performed by splitting all numbers (and intermediate results) into positive and negative parts.
2. Data representation is limited by maximum frequency of spikes. To represent different values within a matrix, we need to scale all quantities so that no number exceeds this maximum frequency.

To repeat the argument presented in Shukla et al. (2017), **Figure 3** illustrates the importance of selecting a correct scaling factor to represent multiple values in an input vector or an input matrix. Three values are given as inputs to TrueNorth (2, 4, and 5) and represented as spike trains. In **Figure 3A**, all values have been scaled by the maximum-magnitude element 5, so all values can be represented within the available range of spiking rate. In **Figure 3B**, the inputs are scaled by a value smaller than the maximum magnitude, so saturation occurs: two elements (4 and 5) are represented by the same spike rate. Selecting the correct scaling factor is important when spike based arithmetic operations may produce results that are larger than any of the inputs. **Figure 3C** shows addition between two values represented as spike trains. Although both operands can be represented exactly with a scale factor of 4, the result of the addition is greater than the chosen scale factor, so the representation saturates and the result is inaccurate.

In implementating algorithms on TrueNorth, therefore, we must choose a scale factor that ensures that the intermediate computations never saturate. On the other hand, the scale factor should not be much larger than necessary, as this will result in loss of precision for the spike-train representations.

2.2. Range Analysis to Determine Input Scaling Factor

A Hopfield linear solver (Lendaris et al., 1999; Shukla et al., 2017) can be used to compute the Moore-Penrose generalized matrix inverse based on the mathematical principles proposed by Ben-Israel and Charnes (1963). This section derives scaling factors that must be applied to the inputs to the system to guarantee that the vectors X_k that arise in the stationary iterative process Equation (4) [equivalently, Equation 8] have no elements greater than 1 in absolute value, for all k. This requirement is achieved by means of a scaling factor η applied to the right-hand side B in Equation (1).

For purposes of this section we define the max-norm of a matrix to be its largest element in absolute value, that is,

$$\|Y\|_{\max} := \max_{i,j} |[Y]_{ij}|. \tag{10}$$

(Note that when Y is a vector, the max-norm is the same as the ∞-norm.) Suppose that the (i,j) element of Y is the one that achieves the maximum norm. We have that

$$\|Y\|_2 \geq \frac{\|Ye_j\|_2}{\|e_j\|_2} \geq |[Y]_{ij}|, \quad \text{for any } i,$$

where e_j is the vector whose elements are all zero except for a 1 in position j. Thus

$$\|Y\|_2 \geq \|Y\|_{\max}. \tag{11}$$

We write the singular value decomposition of A as follows:

$$A = U\Sigma V^T, \tag{12}$$

where U is an $M \times N$ matrix with orthornormal columns, Σ is an $N \times N$ diagonal matrix with nonnegative diagonals, and V is an $N \times N$ orthogonal matrix. In fact, the diagonals of Σ are the singular values of A:

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_N), \tag{13}$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_N \geq 0$. We further use notation

$$\sigma_{\max} := \sigma_1, \quad \sigma_{\min} := \min_{\sigma_i > 0} \sigma_i. \tag{14}$$

In this notation, we have that $\lambda_{\max}(A^TA) = \sigma_{\max}^2$, so that condition Equation (7) becomes

$$0 < \alpha < \frac{2}{\sigma_{\max}^2}. \tag{15}$$

Note too that $\|A\|_2 = \|A^T\|_2 = \sigma_{\max}$.

The following claim shows how we can scale the elements of B to ensure that $\|X_k\|_{\max} \leq 1$ for all k.

CLAIM 1. *For the iterative process defined by Equation (4), and supposing that condition Equation (15) holds, we have that*

$$\|X_l\|_{\max} \leq \frac{2}{\sigma_{\min}}\sqrt{MN}\|B\|_{\max}, \quad \textit{for } l = 0, 1, 2, \ldots. \tag{16}$$

Proof: By applying Equation (4) recursively, we have for all l that

$$X_l = \alpha \sum_{k=0}^{l} (I - \alpha A^TA)^k A^TB. \tag{17}$$

From Equation (12) we have that, $I - \alpha A^TA = V(I - \alpha\Sigma^2)V^T$, so that, $X_l = \alpha \sum_{k=0}^{l} V(I - \alpha\Sigma^2)^k \Sigma U^TB$. By multiplying both sides by V^T, we have that

$$V^TX_l = \alpha \left[\sum_{k=0}^{l} (I - \alpha\sigma_i^2)^k \sigma_i U_{:i}^T B \right]_{i=1,2,\ldots,N},$$

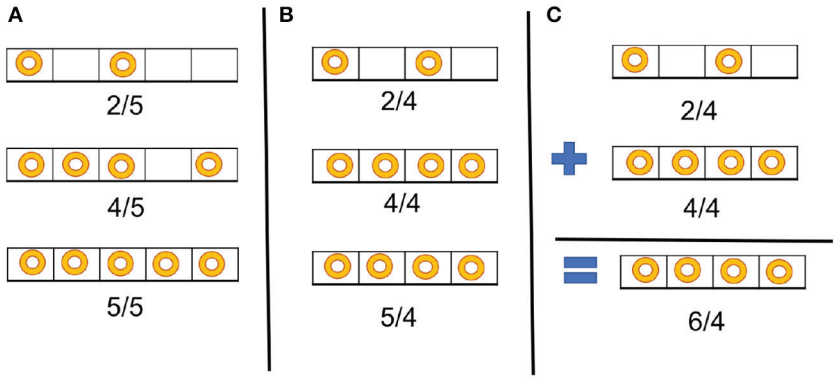

FIGURE 3 | Example illustrating the importance of proper scaling for spike-based computation. **(A)** All three values of a vector are scaled properly. **(B)** Inappropriate scaling: two values (4 and 5) are represented by the same spike rate. **(C)** Addition of two numbers that are scaled properly, but the scale factor is too small to allow proper storage of the result of the addition, leading to saturation.

where $U_{\cdot i}$ is the ith column of U. By carrying out the summation, we have

$$[V^T X_l]_{i\cdot} = \alpha \frac{1-(1-\alpha\sigma_i^2)^{l+1}}{1-(1-\alpha\sigma_i^2)} \sigma_i U_{\cdot i}^T B, \quad i = 1, 2, \ldots, N,$$

so that

$$[V^T X_l]_{i\cdot} = \begin{cases} 0 & \text{for } \sigma_i = 0; \\ \frac{1}{\sigma_i}[1-(1-\alpha\sigma_i^2)^{l+1}] U_{\cdot i}^T B & \text{for } \sigma_i > 0. \end{cases}$$

Since $1-\alpha\sigma_i^2 \in (-1, 1)$ for all $i = 1, 2, \ldots, N$ with $\sigma_i > 0$, we have for such i that

$$\begin{aligned} \|[V^T X_l]_{i\cdot}\|_{\max} &\leq \frac{2}{\sigma_i} \|U_{\cdot i}^T B\|_{\max} \leq \frac{2}{\sigma_i} \|U_{\cdot i}\|_1 \|B\|_{\max} \\ &\leq \frac{2}{\sigma_{\min}} \sqrt{M} \|B\|_{\max} \end{aligned} \tag{18}$$

where the last inequality follows from $\|U_{\cdot i}\|_2 = 1$, the standard inequality that relates $\|\cdot\|_2$ to $\|\cdot\|_1$, and the definition (14). By considering $V^T X_l$ one column at a time, we have

$$\begin{aligned} \|X_l\|_{\max} &= \|VV^T X_l\|_{\max} \leq \|V\|_1 \|V^T X_l\|_{\max} \\ &\leq \sqrt{N} \frac{2}{\sigma_{\min}} \sqrt{M} \|B\|_{\max}, \end{aligned}$$

and by applying Equation (14), we obtain the result. □

An immediate corollary of this result is that if we re place B by $B/(\eta \|B\|_{\max})$ in Equation (1), where

$$\eta := \frac{2}{\sigma_{\min}} \sqrt{MN}, \tag{19}$$

then the matrices X_l produced by the iterative process Equation (8) have $\|X_l\|_{\max} \leq 1$ for all $l = 0, 1, 2, \ldots$. We note too that from the definition Equation (9b) of W_{ff} and Equation (19), we have by setting $B = I$ and $l = 0$ in Claim 1 that

$$\|W_{\text{ff}}\|/\eta \leq 1. \tag{20}$$

By applying this scaling, and writing the solution X of Equation (1) as an infinite sum, we have

$$\begin{aligned} X &= (\eta \|B\|_{\max}) \sum_{k=0}^{\infty} (I - \alpha A^T A)^k \alpha A^T \frac{B}{\eta \|B\|_{\max}} \\ &= (\eta \|B\|_{\max}) \sum_{k=0}^{\infty} (I - \alpha A^T A)^k \alpha A^T B_n, \end{aligned} \tag{21}$$

where $B_n := B/(\eta \|B\|_{\max})$. We use H_j to denote the jth scaled partial summation in Equation (21), that is

$$H_{j+1} = \sum_{k=0}^{j} (W_{\text{hop}})^k W_{\text{ff}} \frac{B}{\eta \|B\|_{\max}} \tag{22a}$$

$$= \sum_{k=0}^{j} (W_{\text{hop}})^k W_{\text{ff}} B_n \tag{22b}$$

$$= W_{\text{hop}} H_j + W_{\text{ff}} B_n. \tag{22c}$$

By setting $j = 0$ in Equation (22c), we obtain

$$H_1 = W_{\text{ff}} B_n. \tag{23}$$

Note that H_l and X_l differ from each other only by the scaling factor $\eta \|B\|_{\max}$, so we have from Claim 1 and Equation (19) that

$$\|H_l\|_{\max} = \frac{1}{\eta \|B\|_{\max}} \|X_l\|_{\max} \leq 1, \quad l = 1, 2, \ldots, \tag{24}$$

and thus

$$\|H_l\|_2 \leq \sqrt{NP}, \quad l = 1, 2, \ldots. \tag{25}$$

2.3. Weight Assignment

We present two techniques to encode weights for the Hopfield neural network based linear solver of section 2.1. In the first subsection, we consider hardcoding the Hopfield neural network weights as TrueNorth neuron parameters. The extracted features

of the image are used to compute weight matrices W_{hop} and W_{ff}, and these are converted further as TrueNorth neuron weight and threshold parameters. This scheme is suitable when the initial features do not change, as in 2-D image tracking. In the second subsection, we see how the computations are performed when weights are introduced as spikes. This scheme is appropriate for scenarios in which the initial conditions may vary frequently, such as optical flow and inverse kinematics.

2.3.1. Hopfield Neural Network Features Encoded as TrueNorth Weights and Threshold

To perform matrix multiplication with weight matrices W_{ff} and W_{hop}, the floating point values of these two weight matrices are encoded as a ratio of TrueNorth weights to thresholds; see Algorithm 1. Here, a single synapse is used for each term in the dot product computation. In TrueNorth, each neuron can have up to four axon types as input, each of which can be assigned a unique synaptic weight in the range $[-255, 255]$. **Figure 4A** shows the synaptic connections in TrueNorth that implement dot product between the vector $[H_k(1,1); H_k(2,1); H_k(3,1)]$ and the columns of the 3×3 weight matrix W_{hop} (which can have either positive or negative values). Each of the three values in H_k have been assigned a different axon type, so that they are multiplied with a corresponding weight value to compute a dot product of the form $w_1 H_k(1,1) + w_2 H_k(2,1) + w_3 H_k(3,1)$. Each neuron i, has its reset mode set to linear reset ($\gamma_i = 1$) and rest of the parameters of the LIF neuron have the default initial value.

Algorithm 1 Computes the weights and threshold values for performing dot product on TrueNorth

Input: Floating point values in the i^{th} row of weight matrices $(W_{i,.})$

Output: Assigned TrueNorth weight and threshold

procedure WEIGHTTHRESHOLDASSIGNMENT

$\text{Threshold} = \text{Round}\left(\frac{255}{\max_i(|W_{i,.}|)}\right)$

$\text{Weights} = \text{Round}\left(\frac{255}{\max_i(|W_{i,.}|)} \times W_{i,.}\right)$

end procedure

Figure 4A presents the scenario where all weights in the Hopfield neural network (both W_{ff} and W_{hop}) can be encoded on a single TrueNorth neuron. Using all of the four axon types available in a single TrueNorth neuron, we can encode a Hopfield neural network that has four W_{ff} and W_{hop} neurons. For scenarios where the Hopfield neural network might have more than four neurons in either W_{ff} or W_{hop}, then the matrix multiplication would have to be divided as partial sums across multiple TrueNorth neurons. **Figure 4B** presents the setup for multiplying vector H_k by a single column of the matrix W_{hop}. Multiple neurons would be required to handle partial sums in matrix multiplication. Partial summation of matrix dot product are computed in neurons N_1 and N_2. Both of these neurons have linear reset mode, and their weight and threshold values are computed using Algorithm 1. Once the partial sums have been computed, the results would go through a separate adder neuron where all of the intermediate sums would be computed. LIF neuron parameters for an adder neuron is shown in **Figure 5C**.

2.3.2. Hopfield Neural Network Features Using Spiking Inputs

For applications such as optical flow and inverse kinematics, where the initial input conditions may change dynamically, the hard-coding of TrueNorth weights discussed in previous subsection is not appropriate. We need an algorithm in which TrueNorth neurons can be used as arithmetic computation units and operate over spiking inputs. Cassidy et al. (2013) show that when the data is represented as stochastic rate-based coding, the theory of stochastic computing (as presented in the survey paper of Alaghi and Hayes, 2013) shows that neurons can perform such arithmetic operations as multiplication, addition, subtraction, and division. Algorithm 2 shows the computation scheme for representing the Hopfield neural network weight matrices W_{hop} and W_{ff} using spikes. **Figure 5A,C,D** shows the LIF parameters of TrueNorth neurons that needs to be set to perform arithmetic operations such as multiplication, addition and subtraction, respectively.

Algorithm 2 Computes weight matrices W_{ff} and W_{hop} using spiking inputs

Input: Spiking coding of the elements in the matrices $\frac{I}{2}$, $(\sqrt{\frac{\alpha}{2}})A^T$, $(\sqrt{\frac{\alpha}{2}})A$, $\frac{\alpha A^T}{\eta}$. Before giving these elements as input to Truenorth, separate the elements into positive and negative domains.

Output: Assigned TrueNorth weight and threshold

procedure COMPUTEWEIGHTSUSINGSPIKES

$(\alpha A^T A)^+ = \max(\alpha A^T A, 0)$;

$(\alpha A^T A)^- = \max(-\alpha A^T A, 0)$;

$(\alpha A^T)^+ = \max(\alpha A^T, 0)$;

$(\alpha A^T)^- = \max(-\alpha A^T, 0)$;

$P_1 = \max\left(\frac{I}{2} - \frac{(\alpha A^T A)^+}{2}, 0\right)$;

$P_2 = \max\left(\frac{(\alpha A^T A)^+}{2} - \frac{I}{2}, 0\right)$;

$P_3 = \left(\frac{(\alpha A^T A)^-}{2}\right)$;

$W^+_{\text{hop}} = 2(P_1 + P_3)$;

$W^-_{\text{hop}} = 2(P_2)$;

$W^+_{\text{ff}} = (\alpha A^T)^+/\eta$;

$W^-_{\text{ff}} = (\alpha A^T)^-/\eta$;

end procedure

Since the matrix computations for W_{hop} and W_{ff} will be done in the hardware itself, we need to reconstruct the iteration formula in such a way that every term that serves as an input to the hardware has magnitude < 1, otherwise the computations might saturate and give us the wrong result. We rewrite Equation (22a) as follows, to ensure that each bracketed term has all its

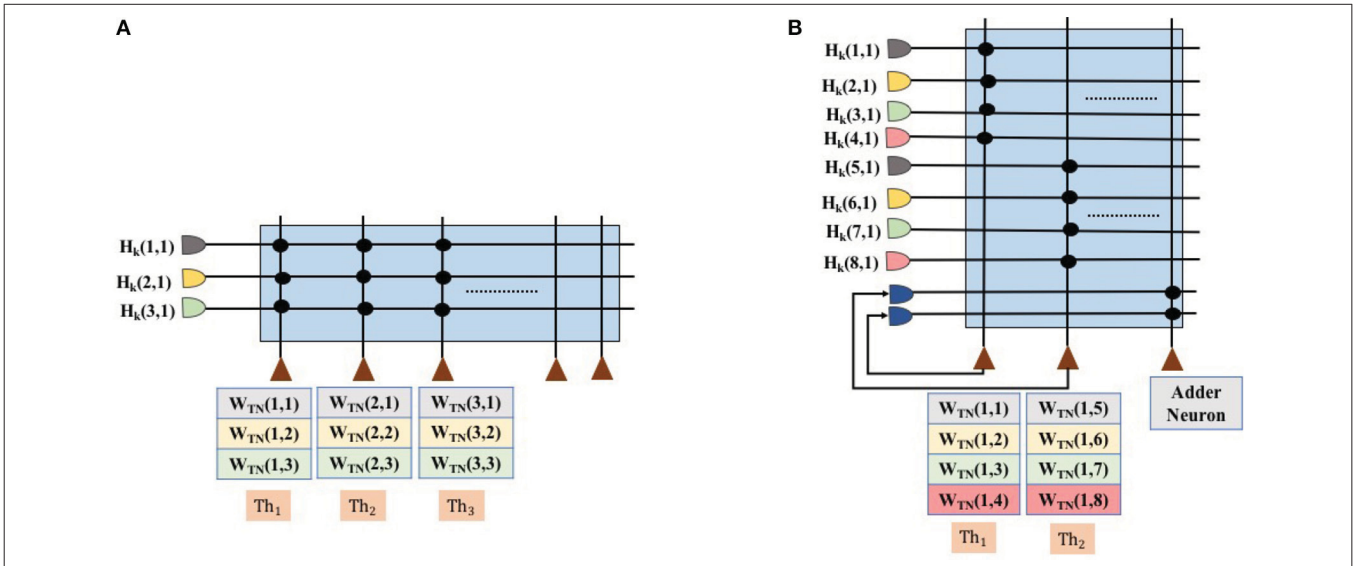

FIGURE 4 | Synapse connection showing the dot product between first column of H_k and the weight matrix W_{hop}, and the corresponding threshold values for each neuron. **(A)** Shows the matrix dot product for the scenario in which W_{ff} and W_{hop} can be encoded using a single neuron. **(B)** Shows the matrix dot product for the scenario where W_{ff} and W_{hop} cannot be encoded using a single neuron. We would need multiple neurons to compute partial sums and later add them up together.

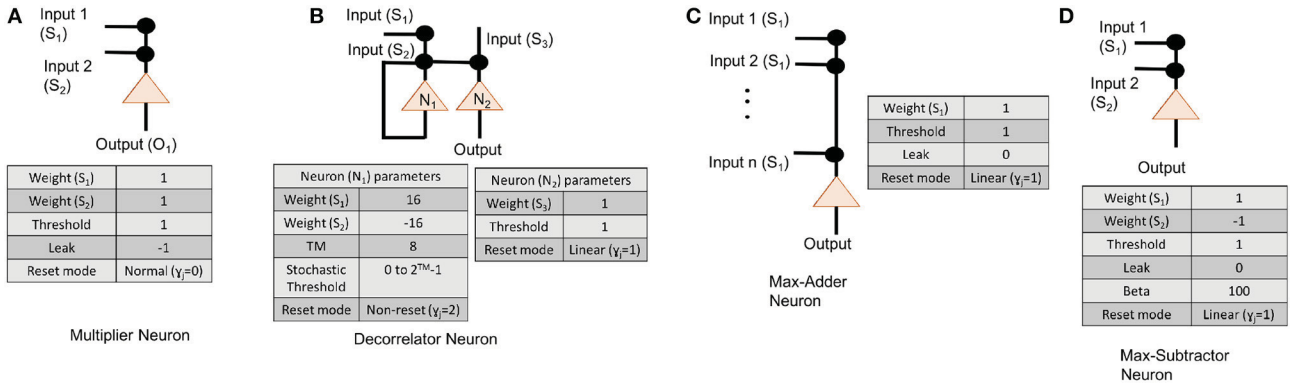

FIGURE 5 | Neuron parameters for **(A)** multiplier neuron that is modeled as AND gate, **(B)** an subtractor neuron that is meant to implement max-sub function, **(C)** an adder neuron, and **(D)** decorrelator neurons.

elements in the range $[-1, 1]$:

$$H_{j+1} = \sum_{k=0}^{j} \left(\frac{I}{2} - \left(\sqrt{\frac{\alpha}{2}}A^T\right)\left(\sqrt{\frac{\alpha}{2}}A\right)\right)^k 2^k \left(\frac{\alpha A^T}{\eta}\right)\left(\frac{B}{\|B\|_{\max}}\right). \tag{26}$$

We state the formal claim as follows.

CLAIM 2. *All elements of the matrices* $\left(\sqrt{\frac{\alpha}{2}}A\right)$, $\left(\sqrt{\frac{\alpha}{2}}A^T\right)$, W_{hop}, *and* $\left(\frac{\alpha A^T}{\eta}\right)$ *lie in the interval* $[-1, 1]$. *That is, the max-norms Equation (10) of these four matrices are all less than* 1.

Proof: Because of Equation (11), it suffices to show that $\|\cdot\|_2 \leq 1$ for all four of the matrices in question.

For the first matrix, note from Equation (15) that $\sqrt{\alpha/2} \leq 1/\sigma_{\max} = 1/\|A\|_2$. Thus $\left\|\sqrt{\frac{\alpha}{2}}A\right\|_2 \leq 1$, as required. The proof for the second matrix is identical.

For the third matrix we note that W_{hop} is a square symmetric matrix with eigenvalues in the range $[-1, 1]$. Thus the eigenvalues of W_{hop}^2 will be in the range $[0, 1]$, so $\|W_{\text{hop}}\|_2 \leq 1$, as required.

For the fourth matrix, we have from Equation (15), the definition of η in Equation (19), and the fact that $\|A\|_2 = \sigma_{\max}$ that

$$\left\|\frac{\alpha A^T}{\eta}\right\|_2 \leq \frac{2}{\sigma_{\max}^2}\sigma_{\max}\frac{\sigma_{\min}}{2\sqrt{MN}} = \frac{\sigma_{\min}}{\sigma_{\max}\sqrt{MN}} \leq 1.$$

■

2.4. Implementation

The spiking neural substrates can operate only for values in the range [0, 1]. Thus, to perform computation on numbers that can be either positive or negative, the computations must be divided into two separate domains, one working with the positive parts of the matrices and one with the negative parts. Algorithms 3 and 4 implement the formula Equation (22b). Algorithm 3 performs the preprocessing step or the feedforward path of the neural network architecture, while Algorithm 4 implements the recurrent part of the Hopfield architecture. The steps shown in Algorithms 3 and 4 ensure that the intermediate computation values never saturate. This is managed by performing subtraction of intermediate results followed by addition in the final step. Since the input values were normalized by the scaling factor, as shown in Equation (19), the addition of partial sums would never saturate. We use the following definition of the positive and negative parts of a matrix:

$$Y^+ := \max(Y, 0), \quad Y^- := \max(-Y, 0), \tag{27}$$

where the max-operation is applied component-wise. The proposed architecture ensures that nonzero elements in the positive-part matrices have zeros in the corresponding elements of the negative-part matrices, and vice versa. We do not have to scale the values in Algorithm 4 while computing matrices M, PS_1, and PS_2 because Claim 1 guarantees that no quantity will exceed 1, by choice of scale factor η. The max function used in Algorithms 2, 3, and 4 can be implemented with a LLIF (linear leaky integrated fire) neuron.

Algorithm 3 Computes the scaled value of input features B based on W_{ff} and the normalizing factor. These scaled values serve as the input for recurrent network

Input: Coordinates of the current input features B
Output: Scaled values of input features for the recurrent network. These values are divided among four domains

procedure PREPROCESSING
  if Weights are hard-coded on TrueNorth **then**
    $B_n = \text{Normalize}(B, \eta \|B\|_{\max})$;
  else if Weights are given as spiking inputs **then**
    $B_n = \text{Normalize}(B, \|B\|_{\max})$;
  end if
  $T^{\langle+,+\rangle} = \max(W_{\text{ff}}^+ B_n^+ - W_{\text{ff}}^+ B_n^-, 0)$;
  $T^{\langle+,-\rangle} = \max(W_{\text{ff}}^+ B_n^- - W_{\text{ff}}^+ B_n^+, 0)$;
  $T^{\langle-,-\rangle} = \max(W_{\text{ff}}^- B_n^- - W_{\text{ff}}^- B_n^+, 0)$;
  $T^{\langle-,+\rangle} = \max(W_{\text{ff}}^- B_n^+ - W_{\text{ff}}^- B_n^-, 0)$;
  $B_s^{\langle+,+\rangle} = \max(T^{\langle+,+\rangle} - T^{\langle-,+\rangle}, 0)$;
  $B_s^{\langle-,+\rangle} = \max(T^{\langle-,+\rangle} - T^{\langle+,+\rangle}, 0)$;
  $B_s^{\langle-,-\rangle} = \max(T^{\langle-,-\rangle} - T^{\langle+,-\rangle}, 0)$;
  $B_s^{\langle+,-\rangle} = \max(T^{\langle+,-\rangle} - T^{\langle-,-\rangle}, 0)$;
end procedure

To implement these arithmetic operations we set the TrueNorth neuron parameters appropriately. A detailed description of individual neuron parameters and their behavior with respect to TrueNorth's spiking neurons can be found in Cassidy et al. (2013). **Figure 5C** shows the neuron parameters and connections for implementing an adder function that is required to compute variables such as M in Algorithm 4. Similarly, **Figure 5B** shows the neuron parameters and connections for max-subtractor neuron that is meant to compute variables such as PS_1, and PS_2 in Algorithm 4, or, variable B_s in Algorithm 3.

Algorithm 4 Solve for system of linear equations defined as $AX = B$. Computations are divided into negative and positive parts

Input: Matrices A and B that have been divided into positive and negative domains
Output: Solution for the system of linear equation, matrix X

procedure HOPFIELDSOLVER_SPLIT
  while $\delta \geq$ Minimum Error **do**
    $M^{\langle+,+\rangle} = (B_s^{\langle+,+\rangle}) + (W_{\text{hop}}^+ H_k^+)$;
    $M^{\langle+,-\rangle} = (B_s^{\langle+,-\rangle}) + (W_{\text{hop}}^+ H_k^-)$;
    $M^{\langle-,-\rangle} = (B_s^{\langle-,-\rangle}) + (W_{\text{hop}}^- H_k^-)$;
    $M^{\langle-,+\rangle} = (B_s^{\langle-,+\rangle}) + (W_{\text{hop}}^- H_k^+)$;
    $PS_1^{\langle+,+\rangle} = \max(M^{\langle+,+\rangle} - M^{\langle+,-\rangle}, 0)$;
    $PS_1^{\langle+,-\rangle} = \max(M^{\langle+,-\rangle} - M^{\langle+,+\rangle}, 0)$;
    $PS_1^{\langle-,-\rangle} = \max(M^{\langle-,-\rangle} - M^{\langle-,+\rangle}, 0)$;
    $PS_1^{\langle-,+\rangle} = \max(M^{\langle-,+\rangle} - M^{\langle-,-\rangle}, 0)$;
    $PS_2^{\langle+,+\rangle} = \max(PS_1^{\langle+,+\rangle} - PS_1^{\langle-,+\rangle}, 0)$;
    $PS_2^{\langle-,+\rangle} = \max(PS_1^{\langle-,+\rangle} - PS_1^{\langle+,+\rangle}, 0)$;
    $PS_2^{\langle-,-\rangle} = \max(PS_1^{\langle-,-\rangle} - PS_1^{\langle+,-\rangle}, 0)$;
    $PS_2^{\langle+,-\rangle} = \max(PS_1^{\langle+,-\rangle} - PS_1^{\langle-,-\rangle}, 0)$;
    $\tilde{\mathbf{H}}_{k+1}^+ = (PS_2^{\langle+,+\rangle} + PS_2^{\langle-,-\rangle})$;
    $\tilde{\mathbf{H}}_{k+1}^- = (PS_2^{\langle+,-\rangle} + PS_2^{\langle-,+\rangle})$;
    if Weights are hard-coded on TrueNorth **then**
      $H_{k+1}^+ = \tilde{\mathrm{H}}_{k+1}^+$;
      $H_{k+1}^- = \tilde{\mathrm{H}}_{k+1}^-$;
    else if Weights are given as spiking inputs **then**
      $H_{k+1}^+ = \text{Decorrelate}(\tilde{\mathrm{H}}_{k+1}^+)$;
      $H_{k+1}^- = \text{Decorrelate}(\tilde{\mathrm{H}}_{k+1}^-)$;
    end if
    $\delta = \left\| (H_{k+1}^+ - H_{k+1}^-) - (H_k^+ - H_k^-) \right\|$;
    $k = k + 1$;
  end while
  $X_k = \text{Rescale}(H_k, \eta \|B\|_{\max})$;
end procedure

2.4.1. Computation With Spiking Weights

For applications in which matrix A might change dynamically, it is not possible to hard-code the weights on TrueNorth. Instead, we borrow concepts from stochastic computing (Gaines, 1967; Alaghi and Hayes, 2013) to perform multiplication between input

streams using a single neuron. In stochastic computing, if the inputs are represented as independent streams of bits, then the multiplication between these two values can be implemented with just one AND gate. In our implementation, the values are represented as stochastically rate coded spikes, similar to bit streams mentioned earlier. The AND gate can be modeled with an LLIF neuron as shown in **Figure 5A**.

Since the computation involves sending the values through a recurrent path, it is crucial to maintain independence of spike occurrence between the inputs from feedforward path and inputs from recurrent path. Therefore, the inputs that are fed back need to be passed through a decorrelator, as shown in **Figure 1**. The decorrelator (presented in Chen and Hayes, 2014) preserves the spiking rate of the input signal, but makes the occurrence of spikes independent of the randomly generated feedforward values. This is done by incrementing the membrane potential when it receives a spike, let the neuron firing threshold vary stochastically and once the neuron fires decrease the membrane potential by the same magnitude as it was increased. Parameters for modeling a decorrelator using TrueNorth neurons are shown in **Figure 5D**.

2.5. Precision

Computations on the neural network substrate generally have limited precision, producing accumulated error in the output. The two main sources of computation error are (1) quantization of the weights and the input, and (2) stochastic computations, when computations are performed using spiking weights. In this section, we first find the upper bound for the output error for the case where weights are hard-coded into the neural network substrate, considering only quantization errors in the weights and the input. We then update the upper bound for the case in which weights are represented using spikes, so that further stochastic errors arise in the computations.

2.5.1. Quantization Error

Given that the elements in the input and weight matrices contain quantization errors, we examine quantization errors in the output. We denote the errors in W_{hop}, W_{ff}, and B_n by ΔW_{hop}, ΔW_{ff}, and ΔB_n, respectively. If δ_{hop}, δ_{ff}, and δ_{bn} represent upper bounds on the individual elements of ΔW_{hop}, ΔW_{ff}, and ΔB_n, respectively, we have by the dimensions of A ($M \times N$) and B ($M \times P$) that

$$\|\Delta W_{\text{hop}}\|_2 \leq \|\Delta W_{\text{hop}}\|_F \leq N\delta_{\text{hop}}, \tag{28a}$$

$$\|\Delta W_{\text{ff}}\| \leq \sqrt{NM}\delta_{\text{ff}}, \tag{28b}$$

$$\|\Delta B_n\| \leq \sqrt{MP}\delta_{\text{bn}}. \tag{28c}$$

These errors produce an error ΔH in the final output matrix H resulting from iterative application of the formula Equation (22c) (or its equivalents). Assuming we know the exact values of the input matrices A and B_n and weight matrices W_{ff} and W_{hop} without quantization, we can find an upper bound on the norm of the output error ΔH^Q. This result requires a condition on the singular values of the modified iteration matrix $W_{\text{hop}} + \Delta W_{\text{hop}}$, without which the output errors ΔH_j at successive iterations j may diverge. We define

$$\bar{\sigma}_{\max} := \sigma_{\max}(W_{\text{hop}} + \Delta W_{\text{hop}}), \tag{29}$$

and require the following condition to hold:

$$\bar{\sigma}_{\max} < 1. \tag{30}$$

By using the definition Equation (9a), together with Equations (12), (13), (28), and the Wielandt-Hoffmann inequality, we have

$$\bar{\sigma}_{\max} \leq \sigma_{\max}(W_{\text{hop}}) + \|\Delta W_{\text{hop}}\|_F \leq \max\left(|1-\alpha\sigma_1^2|, |1-\alpha\sigma_N^2|\right) + N\delta_{\text{hop}}. \tag{31}$$

Thus a sufficient condition for Equation (30) is

$$\max\left(|1-\alpha\sigma_1^2|, |1-\alpha\sigma_N^2|\right) + N\delta_{\text{hop}} < 1. \tag{32}$$

Note that this condition can be satisfied *only if A has full rank*, that is, $\sigma_N > 0$. If we assume that in addition to Equation (15), α also satisfies the (not very restrictive) condition

$$0 < \alpha < \frac{1}{\sigma_N^2}, \tag{33}$$

then $|1-\alpha\sigma_N^2| = 1-\alpha\sigma_N^2 > 0$, and Equation (32) can hold only if $1-\alpha\sigma_N^2 + N\delta_{\text{hop}} < 1$, that is,

$$\delta_{\text{hop}} < \frac{1}{N}\alpha\sigma_N^2. \tag{34}$$

This condition bounds the allowable error in the elements of W_{hop} in terms of the steplength α and the spectrum of A.

Recall for the following result that the dimensions of matrices A and B_n are $M \times N$ and $M \times P$, respectively.

CLAIM 3. *Suppose that Equation (30) holds. Then the upper bound on 2-norm of accumulated error is as follows:*

$$\|\Delta H^Q\|_2 \leq \frac{1}{(1-\bar{\sigma}_{\max})} E_Q,$$

where E_Q is defined by

$$E_Q := \left(\delta_{\text{ff}}\sqrt{NM}\|B_n\| + \|W_{\text{ff}}\|\delta_{\text{bn}}\sqrt{MP} + \delta_{\text{ff}}\delta_{\text{bn}}\sqrt{NM}\sqrt{MP} + \delta_{\text{hop}}N\sqrt{NP}\right). \tag{35}$$

Proof: We rewrite Equation (22c) to include the errors in the constituent quantities:

$$H_k + \Delta H_k^Q = (W_{\text{ff}} + \Delta W_{\text{ff}})(B_n + \Delta B_n) + (W_{\text{hop}} + \Delta W_{\text{hop}})(H_{k-1} + \Delta H_{k-1}^Q). \tag{36}$$

By subtracting H_k from both sides of Equation (36), we obtain

$$\Delta H_k^Q = \Delta H_1^Q + \Delta W_{\text{hop}} H_{k-1} + (W_{\text{hop}} + \Delta W_{\text{hop}})\Delta H_{k-1}^Q, \quad k = 2, 3, \ldots \tag{37a}$$

$$\Delta H_1^Q = \Delta W_{\text{ff}} B_n + W_{\text{ff}}\Delta B_n + \Delta W_{\text{ff}}\Delta B_n. \tag{37b}$$

By taking norms in Equation (37a) and applying standard norm inequalities, we obtain

$$\begin{aligned}\|\Delta H_k^Q\|_2 &\leq \|\Delta H_1^Q\|_2 + \|\Delta W_{\text{hop}}\|_2\|H_{k-1}^Q\|_2 + \|W_{\text{hop}} \\ &+\Delta W_{\text{hop}}\|_2\|\Delta H_{k-1}^Q\|_2 \qquad (38)\\ &\leq \|\Delta H_1^Q\|_2 + N^{3/2}P^{1/2}\delta_{\text{hop}} + \bar{\sigma}_{\max}\|\Delta H_{k-1}^Q\|_2,\end{aligned}$$

where we used Equations (25) and (28) to bound the second term. By applying this formula recursively for $k-1, k-2, \ldots, 1$, we obtain

$$\begin{aligned}\|\Delta H_k^Q\|_2 &\leq \sum_{l=0}^{k-1} \bar{\sigma}_{\max}^l \left(\|\Delta H_1^Q\|_2 + N^{3/2}P^{1/2}\delta_{\text{hop}}\right) \\ &\leq \frac{1}{1-\bar{\sigma}_{\max}}\left(\|\Delta H_1^Q\|_2 + N^{3/2}P^{1/2}\delta_{\text{hop}}\right). \qquad (39)\end{aligned}$$

We now obtain a bound on $\|\Delta H_1^Q\|_2$. By taking norms in Equation (37b) and using Equation (28), we obtain

$$\begin{aligned}\|\Delta H_1^Q\|_2 &\leq \|\Delta W_{\text{ff}}\|_2\|B_n\|_2 + \|W_{\text{ff}}\|_2\|\Delta B_n\|_2 + \|\Delta W_{\text{ff}}\|_2\|\Delta B_n\|_2 \\ &\leq \sqrt{MN}\delta_{\text{ff}}\|B_n\|_2 + \|W_{\text{ff}}\|_2\sqrt{MP}\delta_{\text{bn}} + \sqrt{MN}\delta_{\text{ff}}\sqrt{MP}\delta_{\text{bn}},\end{aligned}$$

giving the result. ■

This claim can be used to find the amount of resources needed to generate an output $H + \Delta H^Q$ in which the error satisfies a specified bound, for example, $\|\Delta H^Q\|_2 \leq \epsilon$. Note that the values of δ_{hop}, δ_{ff}, and δ_{bn} can be manipulated in various ways to meet these goals. For instance, in a neural substrate like TrueNorth, δ_{bn} can be reduced by increasing the number of time ticks at the cost of increased execution time. On the other hand, reductions in δ_{hop} can be achieved by using multiple neurosynaptic cores at the cost of increased area and power. The validation of the model is discussed in section 3. We leave exploration of such optimizations to future work.

2.5.2. Stochastic Error

As mentioned earlier, stochastic computation is the second key source of error in the Hopfield network. Computation in the stochastic domain is performed not on the exact values of inputs and outputs, but on their expected values. However, the random errors present in the inputs to the computations performed in the network lead to random errors in the output, which accumulate during execution of Hopfield network. In this section, we seeks bounds on stochastic error. Claim 4 shows a bound on the output error for the entire computation in terms of error bounds for a single stochastic matrix multiplication. We complement this claim by estimating the bound on stochastic error in a single stochastic matrix multiplication. It is important to note that there are no useful error bounds that hold with absolute certainty! It is possible—though highly unlikely under reasonable assumptions—for stochastic errors to overwhelm the computation. However, we can use information about the distribution of the errors to give some insight into how these errors propagate through the computation, showing conditions under which we can reasonably expect the results to be acceptably accurate.

Claim 3 defines the error bound due to quantization error, in terms of a quantity E_Q defined in Equation (35). We can use this definition as part of the upper bound for stochastic error. For this analysis we assume that the only stochastic computation performed in the Hopfield network is stochastic multiplication—we assume that additions are exact. Each matrix multiplication yields some stochastic error that propagates through subsequent iterations.

CLAIM 4. *Suppose that Equation (30) holds. We denote by E_M a bound on the stochastic multiplication error caused by multiplication of W_{ff} and B_n, and denote by E_N a bound on stochastic multiplication error caused by multiplication of W_{hop} and H_k. Let E_Q be defined as in Claim 3. Then the error bound for ΔH can be estimated as follows:*

$$\|\Delta H\| \leq \frac{1}{1-\bar{\sigma}_{\max}}(E_Q + E_M + E_N),$$

Proof: We rewrite the Hopfield equation, introducing stochastic error terms into Equation (36), as follows:

$$\begin{aligned}H_k + \Delta H_k = &(W_{\text{ff}} + \Delta W_{\text{ff}})(B_n + \Delta B_n) + \Delta_{M_k} + (W_{\text{hop}} \\ &+\Delta W_{\text{hop}})(H_{k-1} + \Delta H_{k-1}) + \Delta_{N_k},\end{aligned}$$

where Δ_{M_k} and Δ_{N_k} represent the stochastic multiplication errors for $W_{\text{ff}}B_n$ and $W_{\text{hop}}H_{k-1}$. Our assumptions yield the following bounds on these error quantities:

$$\|\Delta_{M_k}\| \leq E_M, \quad \|\Delta_{N_k}\| \leq E_N. \qquad (40)$$

By comparing the formula above, and denoting by ΔH_k^Q the quantization error Equation (37), we obtain the following recursive formula for total error ΔH_k:

$$\begin{aligned}\Delta H_k = &\Delta H_k^Q + \sum_{j=0}^{k}(W_{\text{hop}} + \Delta W_{\text{hop}})^j\Delta_{M_{k-j}} + \sum_{j=0}^{k-1}(W_{\text{hop}} \\ &+\Delta W_{\text{hop}})^j\Delta_{N_{k-j}}. \qquad (41)\end{aligned}$$

This formula represents a closed-form expression for the stochastic error, similar to the closed form equation that was used to derive quantization error in Equations (37a) and (37b). By taking norms, and using Equations (29) and (30) together with Equation (40), we have

$$\begin{aligned}\|\Delta H_k\| &\leq \|\Delta H_k^Q\| + \sum_{j=0}^{k}\|(W_{\text{hop}} + \Delta W_{\text{hop}})^j\Delta_{M_{k-j}}\| \\ &+ \sum_{j=0}^{k-1}\|(W_{\text{hop}} + \Delta W_{\text{hop}})^j\Delta_{N_{k-j}}\| \\ &\leq \|\Delta H_k^Q\| + \sum_{j=0}^{k}\bar{\sigma}_{\max}^j\|\Delta_{M_{k-j}}\| + \sum_{j=0}^{k-1}\bar{\sigma}_{\max}^j\|\Delta_{N_{k-j}}\| \\ &\leq \|\Delta H_k^Q\| + \frac{1}{1-\bar{\sigma}_{\max}}(E_M + E_N).\end{aligned}$$

The result now follows from the bound on $\|\Delta H_k^Q\|$ in Claim 3. ■

We complete the error analysis by obtaining bounds E_M and E_N on the stochastic matrix multiplication error norms. Instead of finding a *certain* value for these upper bounds, we find estimates based on the distribution of the elements of the stochastic error matrices that arise in the computations. (It is possible to find a certain bound, but it is too loose to be useful, and the error that actually appears during computation rarely approaches this certain bound.)

Before describing the stochastic errors that arise from a matrix multiplication, we examine the error in multiplying two *scalars* in the range $[0, 1]$. Let y_1 and y_2 be two such scalars, and let $z = y_1 y_2$ be their product. On the spiking substrate, y_1 and y_2 are represented by spike trains of length L, and we denote by Y_1 and Y_2 (respectively) their *represented* values, which are random variables. Since each spike can be thought of as a binary random variable, the expected value and variance of Y_1 and Y_2 are as follows:

$$E(Y_1) = y_1, \quad Var(Y_1) = \frac{y_1(1-y_1)}{L} \tag{42a}$$

$$E(Y_2) = y_2, \quad Var(Y_2) = \frac{y_2(1-y_2)}{L}. \tag{42b}$$

Denoting by Z the random variable resulting from the multiplication of Y_1 and Y_2, we have that $E(Z) = z = y_1 y_2$ and

$$\sigma_Z^2 = \text{Var}(Z) = \text{Var}(Y_1)E(Y_2) + \text{Var}(Y_2)E(Y_1) + \text{Var}(Y_1)\text{Var}(Y_2). \tag{43}$$

The value of σ_Z^2 approaches its minimum value of 0 when Y_1 and Y_2 are close to 0 or 1. A closed form solution can be calculated for the *maximum* value of σ_Z over all possible $y_1, y_2 \in [0, 1]$.

CLAIM 5. *For large L, σ_Z^2 reaches its maximum when $y_1 = y_2 \approx \frac{2}{3}$, and this maximum value is approximately* $0.296/L$.

Proof: From Equations (43) and (42), we have that:

$$\sigma_Z^2 = \frac{y_1(1-y_1)y_2}{L} + \frac{y_1 y_2(1-y_2)}{L} + \frac{y_1(1-y_1)y_2(1-y_2)}{L^2}.$$

As L increases, the third term is dominated increasingly by the first two terms, so we can omit it from consideration. By taking the gradient and Hessian of the resulting approximation of σ_Z^2 with respect to (y_1, y_2), we obtain

$$\text{gradient} = \frac{1}{L}\begin{bmatrix} y_2(1-2y_1) + y_2(1-y_2) \\ y_1(1-y_1) + y + 1(1-2y_2) \end{bmatrix},$$

$$\text{Hessian} = \frac{1}{L}\begin{bmatrix} -2y_2 & 2-2y_1-2y_2 \\ 2-2y_1-2y_2 & -2y_1 \end{bmatrix}.$$

It is easy to check that when $y_1 = y_2 = 2/3$, the gradient is zero and the Hessian is negative definite. Thus $y_1 = y_2 = 2/3$ is an approximate maximizer, and the value of σ_Z^2 at this point is approximately $8/(27L) \approx 0.296/L$. ■

Note that each element of the matrix Δ_{M_k} in the proof of Claim 4 is the stochastic error that arises from taking the inner product of two vectors: one row of $W_{\text{ff}} + \Delta W_{\text{ff}}$ and one column of $B_n + \Delta B_n$. These two vectors have length M, and the product matrix has dimension $N \times P$. Since we assume that no stochastic error occurs in matrix additions, the accumulated stochastic error in each element of Δ_{M_k} is the sum of M random variables, each with expectation 0 and variance bounded by $0.296/L$ (for large L). Since the variance of a sum of uncorrelated random variables is the sum of the variances, we can reasonably say that each element of Δ_{M_k} is a random variable with expectation 0 and variance bounded by $0.296M/L$. An appropriate value of E_M in Equation (40) can be obtained by taking the Frobenius norm of an $N \times P$ matrix whose elements are all equal to the standard deviation (the square root of this variance), and multiplying by a "safety factor" greater than 1. If we choose the value 4 for this safety factor, we obtain the following value for E_M:

$$E_M := 4\sqrt{NP(0.296M/L)} \approx 2.176\sqrt{MNP/L}.$$

A similar calculation involving Δ_{N_k} and E_N, using the fact that W_{hop} is $N \times N$ and H is $N \times P$ results in the following estimate for E_N:

$$E_N := 4\sqrt{NP(0.296N/L)} \approx 2.176N\sqrt{P/L}.$$

This derivation of suitable values for the bounds E_M and E_N is informal, but it suffices to give insight into how these bounds vary with the dimensions of the matrices involved, and with the length of the spike train representation L.

3. RESULTS

This section presents the validation results and observations for the mathematical models for scaling factor and the precision analysis. For all experiments, we set $\alpha = 1.9/\text{trace}(A^T A)$; Equation (7) guarantees convergence of the iterative process for this value of the parameter.

3.1. Experiments

A TrueNorth-based Hopfield linear solver was applied in the context of real-time robotics applications in Shukla et al. (2017). This article looked at three different applications—target tracking (**Figure 2A**), optical flow (**Figure 2B**), and inverse kinematics—and reported relative error and absolute error of these experiments. Each of these experiments required a different input matrix dimension, and results were reported for over 500 different input matrix values. However, Shukla et al. (2017) did not include a mathematically complete architecture, nor did it study the effects of computational limitations, both of which have been examined in this paper.

Table 1 shows the results for 15 different types of matrices, with each matrix repeated 20 times. These experiments were conducted using spike based weight representation scheme on the TrueNorth system. In total, 712 TrueNorth neurons were required to implement the proposed algorithm (just 0.067% of available hardware neurons) and we needed just 11 cores of the 4,096 available cores. In the notation of section 2.1, the dimensions are $M = 25$, $N = 2$, and $P = 1$. Thus, A is 25×2, B 25×1 and H_j in Equation (22a) is 2×1.

TABLE 1 | Results for Hopfield linear solver with spike based weight representation.

Experiment number	Properties of matrices *A* and *B*	Additional comments	Time ticks (in millions)	MSE for $\|\Delta H\|$ (%)	SDSE for $\|\Delta H\|$ (%)
1	Each element chosen uniformly in $[-1, 1]$	Most basic test for Hopfield linear solver	1.05	0.0004	0.0013
2	Each element is an integer chosen uniformly in $[-100, 100]$	Observe the behavior of linear solver as the input range is increased	3.5	0.0025	0.008
3	Each element chosen uniformly in $[-100, 100]$	Each element can have fractional values	3.5	0.0014	0.0028
4	Each element chosen uniformly in $[1, 100]$	All elements of *A* have the same sign	4	0.0353	0.19
5	Each element chosen uniformly in $[0.001, 1]$	Analyzing the convergence when the values are small	4	0.0038	0.008
6	Each element chosen uniformly in $[0.0001, 1]$	Analyzing the convergence when the possible values are smaller than previous experiments	4.25	0.0068	0.0234
7	Each element chosen uniformly in $[-1000, 1000]$	Higher precision is required for calculation	4	0.0186	0.0413
8	Each element chosen uniformly in $[-10,000, 10,000]$	Testing for the cases when even more precision is required for calculation	4	0.32	0.83
9	Each element chosen uniformly in $[1, 10,000]$	Matrix *A* has elements with same sign; requires higher precision for convergence	4	1.16	2.97
10	Each element chosen uniformly in $[-1000, 1000]$ except that 50% of elements in *A* and *B* are 0	Effect of sparsity on final result and convergence	4	0.024	0.0488
11	Each element chosen uniformly in $[1, 10,000]$ except for 50% zeros	Effect of sparsity on final result and convergence	4	0.24	0.94
12	Each element chosen uniformly in $[0.0001, 1]$ except for 45% zeros in *A* and *B*	Effect of sparsity on final result and convergence when elements of *A* are small	4.25	0.0038	0.0114
13	Each element chosen uniformly in $[0, 50]$. For matrix *A*, ratio of smallest to largest singular values is about 0.25	Both the eigenvalues of W_{hop} will have magnitude close to 1, but will have opposite signs	4.25	0.37	1.01
14	Each element chosen uniformly in $[-5 \times 10^5, 5 \times 10^5]$	Testing for the cases when up-to 10^{-6} precision would be required for calculation	4.25	5.11	9.54
15	Each element chosen uniformly in $[1, 5 \times 10^5]$	Precision of better than 10^{-6} would be required for calculation and all matrix input values have the same sign	4.25	96.48	316.54

Column 2 describes how the values of matrices A and B were generated, while Column 3 explains why these matrices were chosen. Column 4 presents the number of clock ticks (or the spike duration) for each experiment. Columns 5 and 6 show the percentage mean (MSE) and percentage standard deviation (SDSE) of the squared error of the Hopfield linear solver output relative to double-precision MATLAB quantity ($\|\Delta H\|$).

In **Table 1**, Columns 5 and 6 show the percentage mean (MSE) and percentage standard deviation (SDSE) of the squared error of the Hopfield linear solver output relative to double-precision MATLAB quantity ($\|\Delta H\|$). The results of Hopfield linear solver (matrix H) were compared against the results that were obtained using MATLAB's double precision pseudoinverse function. Per the error analysis in section 2.5.2 and the principles of stochastic computing (Alaghi and Hayes, 2013), progressive precision holds true for the proposed Hopfield solver. That is, as the number of clock ticks increases, the stochastic error asymptotically approaches zero. We can say from the results of **Table 1** that the output matrix has a value which is quite close to its double-precision pseudoinverse counterpart in many cases.

Inputs that require low precision for computations (Experiments 1, 2, and 3 in **Table 1**) converge faster and show lower MSE in comparison to inputs that require higher precision (Experiments 9, 14, and 15 in **Table 1**). Note that the scenarios in which the Hopfield linear solver algorithm shows high MSE occur because the algorithm requires considerably more iterations to converge and precision greater than or equal to 10^{-6} to reach a solution. Since the proposed work is using stochastic computing, it would require at least 1 million ticks in the best-case scenario to represent a precision of 10^{-6} for a single value, as well as requiring more iterations to converge. While implementing the Hopfield solver on spiking neural substrate such as TrueNorth, the developer would have to consider this speed-accuracy tradeoff. For low-precision values, the Hopfield solver would converge faster, but many more ticks may be required for for high precision values.

3.2. Implementation Analysis

When the firing rates of TrueNorth neurons saturate, the actual outputs of the Hopfield linear solver algorithm may no longer match the expected output; in fact, the difference may be quite large. However, for a large enough input scaling factor, the firing rates of neurons will be low enough so that they will never saturate.

We refer to cases 1, 2, and 3 in **Table 2** for range analysis. **Figures 6A–C** show the plots for number of neurons that are saturating at maximum frequency vs. the scaling factor that was assigned to normalize the values of input. The neuron firing rates were collected using the corelet filter API which is a part of IBM TrueNorth's corelet programming environment (Amir et al., 2013). The firing rate of neurons was gathered for matrices A and B with different set of values, as shown in **Table 2**. The factor η (the scale factor bound calculated in section 2.2) proves that the computed values never saturate, irrespective of whether the computations are happening in the positive or negative domain. The bounds shown are high because the maximum value

TABLE 2 | Sample matrices for worked out examples.

Case 1	Case 2	Case 3	Case 4	Case 5
$A = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$	$A = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$	$A = \begin{bmatrix} 0.1 & -0.1 & 0.2 \\ -0.2 & 0.1 & 0.1 \\ 0.1 & 0.4 & -0.1 \end{bmatrix}$	$A = \begin{bmatrix} 0.08 & 8 \\ -1 & 0.01 \end{bmatrix}$	$A = \begin{bmatrix} 0.8 & 1.25 \\ 1 & 0.00008 \end{bmatrix}$
$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$B = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$	$B = \begin{bmatrix} -4 \\ 0.2 \end{bmatrix}$	$B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
$\eta = 60$	$\eta = 20$	$\eta = 30$	$\delta_{\text{hardcoded}} = 0.025\%$ $\delta_{\text{spiking}} = 3.39\%$	$\delta_{\text{hardcoded}} = \text{N/A}$ $\delta_{\text{spiking}} = 0.8\%$

that every element in the matrix can have after the geometric series summation would be a multiple of $\sigma_{\min}^{-1}$. If the matrix A contains elements with very small magnitude then the term $\sigma_{\min}$ will be small as well; as a result we get a larger scale factor. The scenarios where η is close to the desired bound is when all of the elements in a matrix are the same and each element has values of high magnitude, similar to Case 2 in **Table 2** (**Figure 6B**).

Cases 4 and 5 of **Table 2** show the comparison of absolute errors when the same matrices are given as inputs, where W_{ff} and W_{hop} are either hard coded on TrueNorth or are supplied as spike train inputs. As per case 4, absolute error for hardcoded weights ($\delta_{\text{hardcoded}}$) is less than spiking weights (δ_{spiking}) for same number of spike ticks. This is because hardcoding the weights gives us more control over precision when compared with spiking weights. In Case 5, $\delta_{\text{hardcoded}}$ cannot be computed because TrueNorth neuron's threshold parameter has a limited number of bits, so W_{ff} cannot be mapped onto the board using the technique of Algorithm 1. This problem does not occur with the spike train representation, as higher precision can be represented with longer duration.

3.3. Precision Analysis

The goal of this section is to analyze the quantization error bound and stochastic error bound with the worst-case erroneous output of the Hopfield linear solver. We do so by injecting errors into the weight and input matrices and measuring the resulting error. We are evaluating whether the bounds are tight enough so that they are close to the worst-case erroneous Hopfield linear solver output. Initially, quantization error is introduced in the weight and input matrices and its effect is analyzed for the linear solver. Next, stochastic error is added to weight and input matrices, and the linear solver simulation results are compared with the stochastic bound that was derived in section 2.5.2. Despite being hard to predict, our simulations show that the stochastic error can reach an average of 70% of the bound demonstrating sufficient agreement between the analysis and simulated data.

In order to evaluate the effect of quantization error, the output of the Hopfield network is compared for two cases. In the first case, we provide the exact input and weights to the Hopfield network and calculate the output. In the second case, we introduce some error to the input and the weights equal to the maximum quantization error, and again evaluate the output of the network. Finally, we compare these two outputs and calculate the output error.

The error evaluation results are shown in **Table 3** for the cases presented in **Table 2**. These results show that for all cases, the error remains below the estimated bound. In addition, the error in Case 1 is close to the estimated (91%) bound, which indicates that the bound is tight enough to be useful. It is also important to note that in Case 2, where the matrix A has singular values equal to 0, the error remains below the bound. Thus, even though there are scenarios where our analytical approach does not provide a guarantee, in practice, the estimated error bound holds.

Later, we evaluate the stochastic error bound using a similar method as was used for quantization error. However, due to the randomness of the stochastic error, we repeat each test 100 times and report both the average over all repetitions as well as their maximum. As mentioned in section 2.5.2, the calculated bound does not define an absolute upper limit for each repetition, but rather a bound for their average. In other words, we expect the average error over multiple runs to be smaller than the estimated bound, but the maximum value of the error could exceed the bound.

The results for three representative cases (Cases 1, 2, and 3, from **Table 2**) are shown in **Figure 7** for different spike train lengths. The average error always remains below the bound, but the maximum error for Case 2 exceeds the bound at some points. Furthermore, the gap between the bound and the average error is larger than when we only consider the quantization error. This is due to the unpredictable nature of the stochastic error resulting in a looser stochastic error bound.

Simulations of the quantization and stochastic errors show that the proposed bounds can provide reasonable upper limits for the error of the Hopfield network implemented on a neural network hardware. Therefore, these bounds can be used to gain insight into the precision results of this network for any set of input and weights, before running the algorithm. In addition, they can be used to allocate appropriate resources in order to achieve a specific output precision.

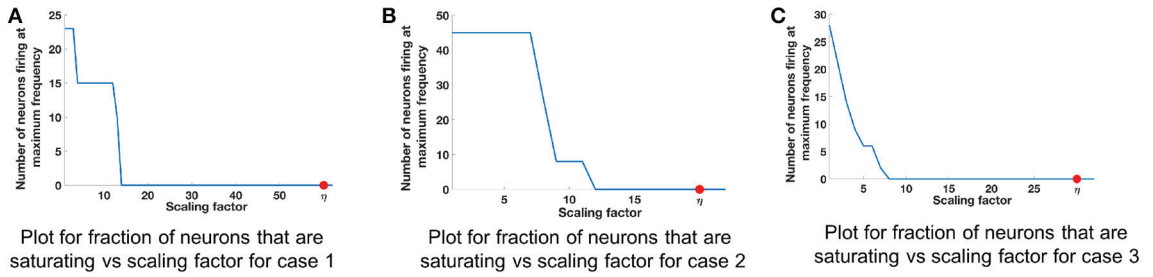

FIGURE 6 | Comparison of scaling factor for different matrix structures. **Table 2** lists the three different input matrices A and B. **(A)** Plot for fraction of neurons that are saturating vs. scaling factor for case 1. **(B)** Plot for fraction of neurons that are saturating vs. scaling factor for case 2. **(C)** Plot for fraction of neurons that are saturating vs. scaling factor for case 3.

TABLE 3 | Quantization error simulation results.

	Error relative to the bound (%)
Case 1	90.75
Case 2	5.13
Case 3	38.63
Case 4	13.89
Case 5	24.86

3.4. Architecture-Application Analysis

Prior work Shukla et al. (2017) showed how these linear solvers can be used to compute transformation matrices for applications such as inverse kinematics, object tracking and optical flow. To analyze the proposed Hopfield linear solver in a practical implementation scenario, we tested the proposed work for Lucas-Kanade based optical flow application that has a setup similar to the one shown in **Figure 2B** and described in the prior works (Esser et al., 2013; Shukla et al., 2017). The image in **Figure 2B** is a grayscale image in which high intensity pixels are represented with a value of 1 and low-intensity pixels are represented with 0. The two black bars in the figure have a pixel width of 5 pixels. the resolution of the image was set at 240-by-360 pixels which is same as QVGA format videos. The horizontal and vertical bars were initially positioned at the center along height and width of the image, respectively. The two lines intersected at the center of the image. The sequences of images are streaming in to the hardware at 30 frames per second. For the first set of frames, the horizontal bar is moving upwards, and the vertical bar is moving toward left. In the implementation, the frame size of QVGA video was first reduced by a factor of 4 to 120-by-180 pixels, then a 5-by-5 pixels convolutional operation was applied to it.

The implementation of a Hopfield linear solver in such a setup is challenging since the Hopfield neural network (W_{ff} and W_{hop}) weights change continuously. Also, in this setup there is no training or testing data involved. The goal here is to compute the results online by just looking at the streaming input values without any prior knowledge of the experiment or scenario. We observe additional benefits by deploying multiple linear solvers in parallel since we have to calculate pseudoinverse for multiple different locations on the image at the same time. These experiments give us better insights with respect to selecting TrueNorth as a potential substrate for deployment of such algorithms, and provides a vehicle for energy analysis when compared with more traditional approaches. In this experiment we measure the motion vector error against the baseline, but have also utilized an approximately correct metric: as long as the solver correctly detects flow in one of eight possible ordinal and cardinal directions, we count it as correct. The velocity of the movement of two bars is calculated by solving for X in the equation $AX = B$. Matrix A contains partial derivatives of initial image frame with respect to directions x and y around pixel qi. This is represented by terms $I_x(qi)$ and $I_y(qi)$, in Equation (44). Matrix B contains partial derivatives of pixel positions between initial image frame and image frame at time t around pixel qi. This is represented by terms $I_t(qi)$, in Equation (45). After implementing matrix division, output matrix X will report the speed and direction of the image pixels, by computing the pseudoinverse of matrix A.

$$A = \begin{bmatrix} I_x(q1) & I_y(q1) \\ I_x(q2) & I_y(q2) \\ \vdots & \vdots \\ I_x(qn) & I_y(qn) \end{bmatrix} \tag{44}$$

$$B = \begin{bmatrix} -I_t(q1) \\ -I_t(q2) \\ \vdots \\ -I_t(qn) \end{bmatrix} \tag{45}$$

In the proposed setup, we can have multiple input matrices A and B (see Equation 3), that are independent of each other, since the convolution operation can operate on separate and independent patches of image at the same time. The results of these independent convolutions can be streamed as different input matrices A and B. As a result, we can have multiple independent linear solvers running in parallel to compute different pseudo-inverses for these different input matrices. For a frame of size 120-by-180 pixels, linear solver implementation processed 9,800 pixels of a single frame to predict the motion vectors.

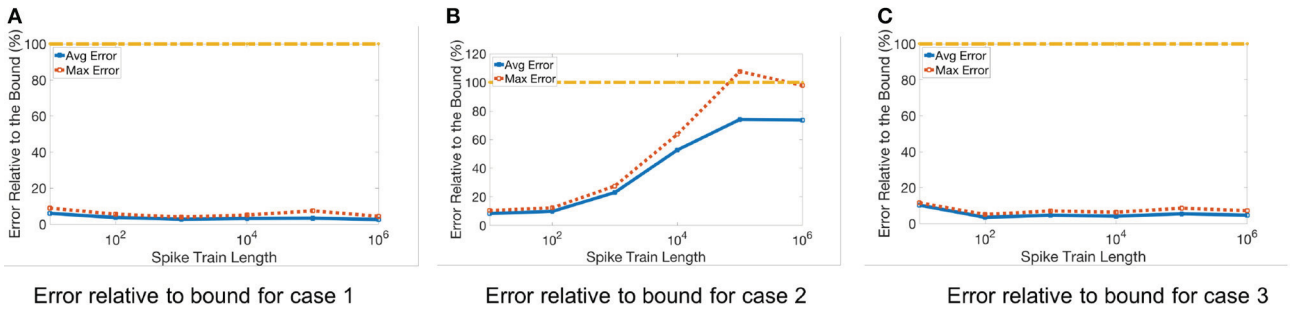

FIGURE 7 | Stochastic error for different spike train lengths. **Table 2** lists the three different input matrices A and B. **(A)** Error relative to bound for case 1. **(B)** Error relative to bound for case 2. **(C)** Error relative to bound for case 3.

Using the optical flow implementation described above, we compare the power and energy consumption of TrueNorth based linear solver implementation with more traditional approaches like QR inverse algorithm on Virtex-7 FPGA (xc7vx980t) and on an ARM cortex A15 mobile processor.

On TrueNorth we can implement 392 instances of the Hopfield linear solver that operate in parallel independent from one another. These 392 instances required 4,092 cores of the available 4,096 cores and can process roughly 9,800 pixels for predicting the motion vectors. Therefore, we would need to compute optical flow motion vectors in the specified scenario in batches of two streaming input pixels for a single 120-by-180 pixels frame.

To maintain the throughput of 30 FPS for 9,800 pixels we needed an 8-core ARM chip operating at 2.5 GHz. For the same FPS and pixel count instantiate 32 instances of QR inverse algorithm on Virtex-7. A detailed discussion about each of the implementation technique is presented as follows:

TrueNorth: We have implemented 392 instances of the Hopfield linear solver which operate in parallel, independent from one another. These 392 instances required 4,092 cores of the available 4,096 cores. The power consumption values were reported from IBM's test and development board. For a supply voltage of 0.8 V and 1KHz operating frequency, the scaled leakage power of our implementation is 46.31 mW and the scaled active power 18.67 mW. Since the goal is to implement optical flow at 30 FPS, we increase the operating frequency of TrueNorth NS1e hardware to 9KHz and report a linearly scaled active power of 168.03 mW for these experiments.

Virtex-7 FPGA: The QR inverse algorithm was implemented using the matrix algebra libraries present in Xilinx Vivado HLS (Xilinx, 2014) and the frequency of the platform was set at 20 MHz. Power analysis of the following implementations were done using Xilinx Power Estimator tool (Xilinx, 2017). For 32 parallel instances of QR inverse solver the total power consumption is 1.881W with a static power consumption of 383 mW.

8-core ARM A15 processor: The QR inverse algorithm was implemented using the C++-based eigen library (Guennebaud et al., 2010). Simulations for matrix inversion were done using gem5 (Binkert et al., 2011) in the system emulation mode and the power consumption details were collected using *McPat* (Li et al., 2009). We used the ARM A15 configurations for Gem5 and McPat simulations that have been presented in (Endo et al., 2015). For processing the QR inverse algorithm, an octa-core ARM Cortex A-15 chip consumed 2.55 W of power which includes 93.5 mW of static power.

Figure 8 shows the comparison of energy consumption and time elapsed for computation on three different hardware platforms. Since TrueNorth can perform computations on 9,800 pixels at a time, it would have to time-multiplex 120-by-180 pixels into two batches to perform the linear solver operation. At 9 KHz frequency each time multiplexed batch would need a maximum of 150 time ticks for computing the inverse on a portion of the image. After 150 ticks, the accuracy of predicting the direction in optical flow is 99.33% and the speed of motion can be estimated with an accuracy of 80.9%. As per the plots in **Figure 8A,B**, the TrueNorth-based linear solver is more energy efficient than the ARM or FPGA implementations. For both the FPGA- and ARM-based QR inverse solvers, the accuracy is 100% as they are using floating point units for computation. The TrueNorth-based linear solver consumes 0.0575 J of energy per frame, the FPGA consumes 0.4074 J of energy per frame and, the ARM processor consumes 4.986 J of energy per frame. On the other hand, the accuracy of TrueNorth depends on how many ticks it requires. As a result, if TrueNorth is operated for more ticks, the solution achieves higher accuracy but consumes more energy.

4. DISCUSSION

To the best of our knowledge, this paper is the first attempt to formalize a mathematical framework for determine scaling factors and error bounds when deploying a recurrent numerical solver on limited-precision neural hardware. The proposed research developed a mathematical and algorithmic framework for calculating generalized matrix inverses on this hardware platform. Apart from using the proposed algorithm for real-time robotics applications, it could also be used for on-chip training of

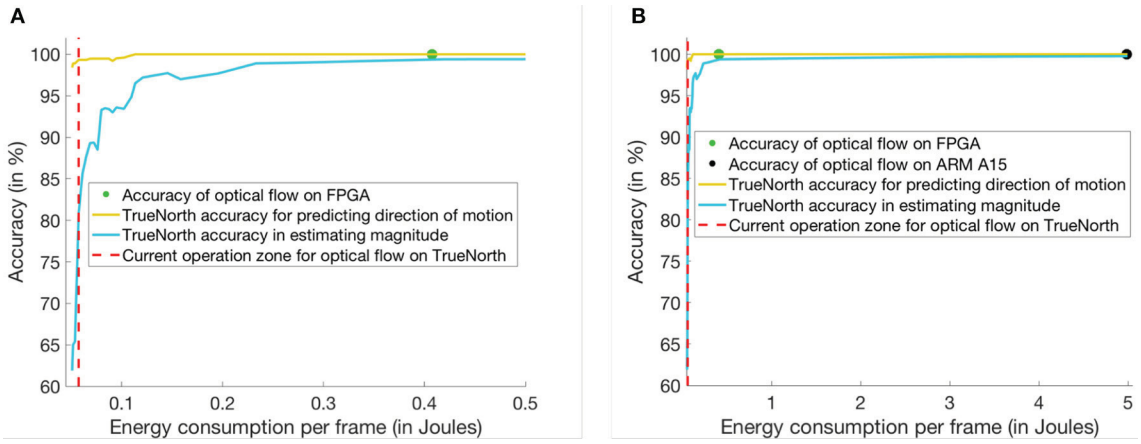

FIGURE 8 | This figure shows a comparison between three different implementation techniques for matrix inversion. Y-axis of the plot shows the percentage accuracy in predicting the motion of bars for optical flow. And, X-axis of the plot shows the energy consumed per frame (in Joules) for optical flow. **(A)** Comparison of power consumed between FPGA and TrueNorth hardware. **(B)** Comparison of power consumed between ARM, FPGA and TrueNorth hardware.

multi-layered perceptrons and extreme-learning machines (Tang et al., 2016) for a variety of classification and regression based tasks. We validate the mathematical model using a Hopfield network-based linear solver that has been implemented on the IBM TrueNorth spiking neural substrate. Our empirical results show that the analytic bounds are never violated for the scenarios evaluated.

4.1. Summary of Experiments and Results

First, section 3.1 compares the results of proposed linear solver against MATLAB's double precision pseudo-inverse function. Results presented in **Table 1** suggests that a stochastic-computing implementation can produce an output matrix that are quite close to their double-precision pseudoinverse counterparts in many cases. However, the developer would have to keep in mind speed-accuracy tradeoff. For low precision values, the Hopfield solver would converge faster, while many more ticks would be required for high precision values.

Second, section 3.2 presents the range analysis of Hopfield linear solver. We can guarantee that the proposed scaling factor will keep the firing rates of neurons low enough that they never saturate. Similarly, in section 3.3, we validate the bounds that were proposed in precision analysis. For quantization error, the experimental errors can get very close to estimated (91%) bound, indicating that the bound is tight enough to be useful. For stochastic errors, the average of experimental error always remains below the bound.

Finally, section 3.4 compares the TrueNorth-based Hopfield linear solver against standard QR inverse algorithms that were implemented on the ARM processor and in FPGA. Experiments with the optical-flow application showed the energy benefits of deploying a reduced-precision and energy-efficient generalized matrix inverse engine on the IBM TrueNorth platform. Since TrueNorth architecture was designed to be low power, deployment of multiple linear solvers running in parallel could give a 10× to 100× improvement in terms of energy consumed per frame over FPGA and ARM core baselines.

4.2. Extending Hopfield Neural Network Based Linear Solver to Other Hardware Substrates

Sections 2.3.2 and 2.4 present algorithms that can compute matrix inverses using concepts from stochastic computing (Gaines, 1967). The proposed algorithms can be extended to other spiking and non-spiking hardware substrates that have the ability to perform stochastic computing and provides the capability to have recurrent neural network connections. Prior work such as Smithson et al. (2016), Thakur et al. (2016), and Cassidy et al. (2013) show that digital spiking neural substrates can perform stochastic computing. We can also perform stochastic computing on non-spiking hardware substrates such as FPGAs (Li et al., 2016), FinFETs (Zhang et al., 2017), and magnetic-tunnel-junction (Lv and Wang, 2017). These technologies provide us with a promising opportunity to implement linear solvers based on Hopfield neural networks while being energy-efficient and operate at a higher frequency. Developers would have to keep in mind that the proposed linear solver is performing lossless addition (**Figure 5C**). When a neuron receives spikes from multiple inputs at the same, its membrane potential increases by the same amount as the number of input spikes it has received at that time tick. The membrane potential decreases by one after the neuron fires. Scaling factor η that was derived in Equation (19) and Claim 1 guarantees that even with a lossless addition present in the equations, the intermediate computation will never saturate.

4.3. Future Work

In future work, we will look into speeding up the computation by using a population coding scheme for encoding values to spikes.

Our current implementation uses a rate coding technique for encoding values with a single neuron. Considering the resources that we have available on TrueNorth board, a population coding scheme could perform computations in parallel, hence reducing the time to solution.

Large scaling factor values may end up resulting in longer computation time, since we end up requiring more ticks to represent the scaled values accurately. Alternatively, tight scaling factors that still avoid saturation require computation of the matrix pseudoinverse, using external hardware. This complicates deployment of the Hopfield solver in scenarios where the matrix changes over time. Developing tighter bounds, especially ones that are easier to compute online, would avoid these problems.

Finally, in the proposed architecture in this paper the term α was precomputed. As part of our future work, we would like to create a TrueNorth based framework where α could be computed dynamically via spikes.

AUTHOR CONTRIBUTIONS

RS was responsible for preparing the manuscript, scaling analysis of linear solver, coming up with the algorithm for hardware implementation of linear solver and all of the experiments related to hardware (TrueNorth, FPGA and ARM) analysis and hardware implementation. RS also did the scaling factor based experiments. SK was responsible for mathematical theory of error analysis and precision based error analysis. EJ did the implementation of algorithm on TrueNorth hardware. JL, and ML were the ones that guided the project, came up with initial ideas. SW suggested corrections for mathematical theorem and proofs, as well as did rewritings in the manuscript.

FUNDING

This work was funded in part by the National Science Foundation through grant CCF-1628384.

REFERENCES

Alaghi, A., and Hayes, J. (2013). Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.* 12:92. doi: 10.1145/2465787.2465794

Amir, A., Datta, P., Risk, W. P., and Cassidy, A. S. (2013). "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX), 1–10. doi: 10.1109/IJCNN.2013.6707078

Ben-Israel, A., and Charnes, A. (1963). Contributions to the theory of generalized inverses. *J. Soc. Indust. Appl. Math.* 11, 55–60. doi: 10.1137/0111051

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. E., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., et al. (2011). The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 1–7. doi: 10.1145/2024716.2024718

Cassidy, A. S., Merolla, P., Arthur, J. V., and Esser, S. K. (2013). "Cognitive computing building block: a versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX), 1–10. doi: 10.1109/IJCNN.2013.6707077

Chen, T. H., and Hayes, J. P. (2014). "Analyzing and controlling accuracy in stochastic circuits," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)* (Seoul), 367–373. doi: 10.1109/ICCD.2014.6974707

Cheung, K., Schultz, S. R., and Luk, W. (2016). Neuroflow: a general purpose spiking neural network simulation platform using customizable processors. *Front. Neurosci.* 9:516. doi: 10.3389/fnins.2015.00516

Endo, F. A., Couroussé, D., and Charles, H.-P. (2015). "Micro-architectural simulation of embedded core heterogeneity with gem5 and mcpat," in *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO '15* (New York, NY: ACM), 7:1–7:6. doi: 10.1145/2693433.2693440

Esser, S. K., Andreopoulos, A., Appuswamy, R., Datta, P., Barch, D., Amir, A., et al. (2013). "Cognitive computing systems: algorithms and applications for networks of neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX), 1–10. doi: 10.1109/IJCNN.2013.6706746

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gaines, B. R. (1967). "Stochastic computing," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67* (New York, NY: ACM), 149–156. doi: 10.1145/1465482.1465505

Goux, L., Fantini, A., Degraeve, R., Raghavan, N., Nigon, R., Strangio, S., et al. (2013). "Understanding of the intrinsic characteristics and memory trade-offs of sub-micron filamentary RRAM operation," in *Symposium on VLSI Circuit Digest of Technical Paper*, Vol. 88 (Kyoto), 2012–2013.

Guennebaud, G., Jacob, B., et al. (2010). *Eigen v3*. Available online at: http://eigen.tuxfamily.org

Hopkins, M., and Furber, S. (2015). Accuracy and efficiency in fixed-point neural ode solvers. *Neural Comput.* 27, 2148–2182. doi: 10.1162/NECO_a_00772

Jin, X., Furber, S. B., and Woods, J. V. (2008). "Efficient modelling of spiking neural networks on a scalable chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (Hong Kong), 2812–2819.

Lendaris, G., Mathia, K., and Saeks, R. (1999). Linear hopfield networks and constrained optimization. *IEEE Trans. Syst. Man Cybern. B* 91, 114–118. doi: 10.1109/3477.740171

Li, B., Najafi, M. H., and Lilja, D. J. (2016). "Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '16* (New York, NY: ACM), 36–41. doi:10.1145/2847263.2847340

Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. (2009). "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (New York, NY), 469–480.

Lv, Y., and Wang, J. (2017). "A single magnetic-tunnel-junction stochastic computing unit," in *2017 International Electron Devices Meeting, IEDM '17*. San Francisco, CA.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Narayanan, S., Shafiee, A., and Balasubramonian, R. (2017). "Inxs: bridging the throughput and energy gap for spiking neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK), 2451–2459. doi: 10.1109/IJCNN.2017.7966154

Schemmel, J., Fieres, J., and Meier, K. (2008). "Wafer-scale integration of analog neural networks," in *Proceedings of the International Joint Conference on Neural Networks* (Hong Kong), 431–438. doi: 10.1109/IJCNN.2008.4633828

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., et al. (2017). A survey of neuromorphic computing and neural networks in hardware. *arXiv:1705.0696*

Shukla, R., Jorgensen, E., and Lipasti, M. (2017). "Evaluating hopfield-network-based linear solvers for hardware constrained neural substrates," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK), 1–8.

Shukla, R., and Lipasti, M. (2015). "A self-learning map-seeking circuit for visual object recognition," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney), 1–8.

Smithson, S. C., Boga, K., Ardakani, A., Meyer, B. H., and Gross, W. J. (2016). "Stochastic computing can improve upon digital spiking neural networks," in *2016 IEEE International Workshop on Signal Processing Systems (SiPS)* (Dallas, TX), 309–314.

Tang, J., Deng, C., and Huang, G. B. (2016). Extreme learning machine for multilayer perceptron. *IEEE Trans. Neural Netw. Learn. Syst.* 27, 809–821. doi: 10.1109/TNNLS.2015.2424995

Thakur, C. S., Afshar, S., Wang, R. M., Hamilton, T. J., Tapson, J., and van Schaik, A. (2016). Bayesian estimation and inference using stochastic electronics. *Front. Neurosci.* 10:104. doi: 10.3389/fnins.2016.00104

Xilinx (2014). *Vivado Design Suite User Guide-High Level Synthesis*. Available online at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf

Xilinx (2017). *Xilinx Power Estimator*. Available online at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug440-xilinx-power-estimator.pdf

Zhang, Y., Wang, R., Jiang, X., Lin, Z., Guo, S., Zhang, Z., et al. (2017). "Design guidelines of stochastic computing based on finfet: a technology-circuit perspective," in *2017 International Electron Devices Meeting, I EDM '17*. San Francisco, CA.

Conflict of Interest Statement: ML has financial interest in Thalchemy corp. and is co-founder of the said corporation. Thalchemy corp. was not at all involved in this research project in any form.

The other authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.