Simulating Message Passing via Spiking Neural Networks Using Logical Gates

Anonymous Author(s)
Affiliation
Address
email

Abstract

It is hypothesized that the brain functions as a Bayesian inference engine, continuously updating its beliefs based on sensory input and prior knowledge. Message passing is an effective method for performing Bayesian inference within graphical models. In this paper, we propose that the XOR and the Equality factor nodes, which are important components in binary message passing, can be realized through a series of logical operations within a spiking neural network framework. Spiking neural networks simulate the behavior of neurons in a more biologically plausible manner. By constructing these factor nodes with a series of logical operations, we achieve the desired results using a minimal number of neurons and synaptic connections, potentially advancing the development of biological neuron-based computation. We validate our approach with two experiments, demonstrating the alignment between our proposed network and the sum-product message-passing algorithm.

1 Introduction

The Bayesian brain hypothesis proposes that the brain functions as a Bayesian inference machine, continuously updating its beliefs about the world by integrating sensory input with prior knowledge (1). This concept aligns with the Free Energy Principle, which asserts that biological systems must minimize uncertainty by using internal models to predict and adapt to environmental changes (3). Experimental evidence supporting these principles can be observed in studies involving in-vitro biological neurons cultured on Micro-Electrode Arrays (MEA). For instance, research by (8) highlights the computational capabilities of cultured neurons in a simulated game of Pong, while (7) investigates their application in blind source separation.

A widely used technique for probabilistic inference under uncertainty is the sum-product message passing on a factor graph, also known as belief propagation. This method offers a structured approach for efficiently performing Bayesian inference in graphical models by exchanging information between factor nodes to update beliefs (12).

Spiking Neural Networks (SNNs) are a class of artificial neural networks that emulate the behavior of biological neurons by using discrete spikes or action potentials for communication. SNNs encode information in the precise timing of these spikes (13). The biologically inspired nature of SNNs provides a more accurate simulation of neural processes, making them significant for advancements in computational neuroscience. In (10), the utility of SNNs is demonstrated by showing how Spike-Timing-Dependent Plasticity (STDP) can be leveraged to enable a self-learning spiking network to control a mobile robot. This study highlights the potential of SNNs in practical applications, where their ability to learn and adapt can enhance robotic control and autonomous decision-making

Submitted to Workshop on Bayesian Decision-making and Uncertainty, 38th Conference on Neural Information Processing Systems (BDU at NeurIPS 2024). Do not distribute.

In this paper, we propose SNN-based factor nodes for simulating Bernoulli message passing using logical nodes. We demonstrate that the proposed network, constructed with a minimal number of neurons and synaptic connections, yields results closely aligned with numerical sum-product rules. This approach represents a potential step toward exploring the use of biological neurons as computational functions.

2 Background

2.1 Sum-Product Message Passing on Forney-style Factor Graphs

The belief propagation or sum-product message passing algorithm, applied to Forney-style Factor Graphs (FFGs), is a powerful method for performing Bayesian inference in probabilistic models (12). In an FFG, edges represent random variables, and nodes represent factors in a probabilistic model. An edge connects to a node if the variable on that edge is an argument of the node's function. The sum-product algorithm works by iteratively passing "messages" along the edges of the graph. We denote the forward and backward messages using the notations $\overrightarrow{\mu}(\cdot)$ and $\overleftarrow{\mu}(\cdot)$, respectively. In general, for any node $f(y, x_1, \ldots, x_n)$, the sum-product rule for an outgoing message over edge y is given by

$$\underbrace{\overrightarrow{\mu}(y)}_{\text{outgoing messages}} = \sum_{x_1,\ldots,x_n} \underbrace{\overrightarrow{\mu}(x_1)\ldots\overrightarrow{\mu}(x_n)}_{\text{incoming messages}} \underbrace{f(y, x_1, \ldots, x_n)}_{\text{node function}} . \tag{1}$$

This algorithm is particularly scalable because it exploits the graph's structure to compute locally, significantly reducing the complexity compared to naive approaches that require summing over all possible configurations globally. The local and distributed nature of the sum-product algorithm allows it to handle large-scale problems efficiently, making it widely applicable. For example, in multi-agent trajectory planning (2) and active inference in complex environments (9).

2.2 Leaky Integrate-and-Fire Neurons

SNNs encompass several biophysical models that describe how neurons generate spikes to a varying degree of realism. Among these, the Leaky Integrate-and-Fire (LIF) model is one of the most widely used, and it is the one we employ here. In the LIF model, a neuron's membrane potential accumulates with incoming synaptic inputs over time, which can be either excitatory or inhibitory. The membrane potential also 'leaks' over time, gradually returning to the resting state in the absence of input, reflecting the neuron's inherent electrical properties. When the membrane potential reaches a certain threshold, the neuron 'fires' an action potential or spike, which propagates down the axon to stimulate other neurons.

3 Simulations

In this section, we start by implementing logical node diagrams using a small number of LIF neurons. We then utilize these diagrams to carry out message passing according to the sum-product rule with Bernoulli messages. Finally, we apply this method to a specific example and compare the results.

To simulate neural activities we utilized the Brian toolbox (4). We set the threshold for neuron firing at 1.0, the resting state at 0.0, and we assumed $dv/dt = -v/\tau$ for $\tau = 1.0$ ms for all neurons.

3.1 Logical Gates

We design the neurons' synaptic connections so as to achieve the output described in the truth table 3.1 for each logical gate. This process is more straightforward for the AND, OR, and NOT gates, as we can determine appropriate synaptic weights to achieve the desired output. The proposed SNN gates are illustrated in figure 1. We restricted the input spikes to every 10 ms. A sample set of inputs and their corresponding outputs are illustrated in the figure 1. x_1 spikes at times $\{10, 20, 40, 50, 70\}$, and x_2 spikes at times $\{0, 20, 40, 60\}$. The outputs of each gate match the expected results.

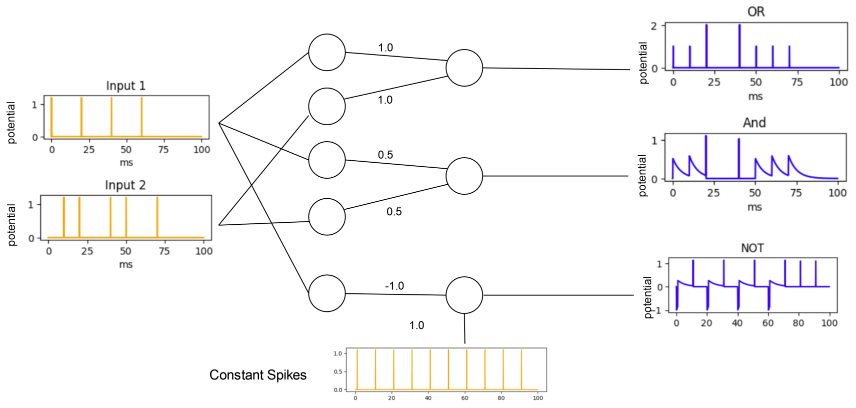

Figure 1: Synaptic weights for the AND, OR, and NOT gates. Each circle represents a single LIF neuron. To implement the NOT gate, a consistent spike train with intervals of 10 ms is applied throughout the simulation. For the AND gate, the output spike is generated only when both inputs are present; otherwise, the voltage increases but does not reach the firing threshold.

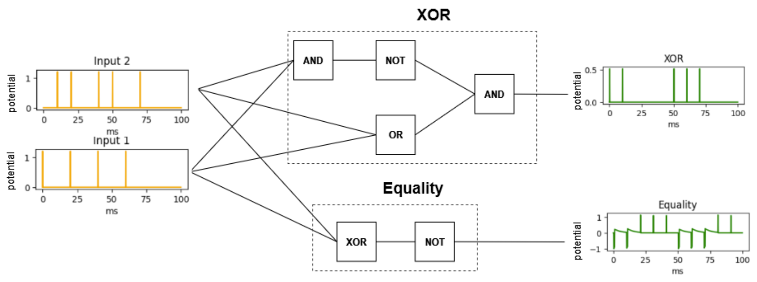

Figure 2: SNN Diagram for XOR gate.

Table 1: Logical Gates Truth Table

x_1	x_2	OR	AND	NOT-x_1	XOR	Equality
0	0	0	0	1	0	0
0	1	1	0	1	1	Not Defined
1	0	1	0	0	1	Not Defined
1	1	1	1	0	0	1

The XOR gate is more complex to generate, meaning it cannot be implemented solely by adjusting synaptic weights, as was done for the gates in Fig 1. However, it can be constructed by combining the implemented gates, as outlined in Fig 2.

3.2 Message Passing

We consider $\overrightarrow{\mu}(x_1) = \mathcal{B}er(x_1|p_1)$ and $\overrightarrow{\mu}(x_2) = \mathcal{B}er(x_2|p_2)$ as the input messages, we sample from these distributions every 10 ms and use them as input spikes. For the XOR gate, we define the output message $\overrightarrow{\mu}(y) = \mathcal{B}er(y|\frac{spike-count}{spike-time})$, where $spike - count$ is the total number of the output spikes according to the 2, and the $spike - time$ is the total times we sample from the input messages. For example, if we run the simulation for 10000 ms and sample each 10 ms from the inputs, the $spike - time$ is 1000 times.

For the Equality node, it is more complicated. As we can see in the truth table, the output spikes of this node are similar to the AND gate, but the combinations of unequal inputs are not defined for this operation. So we can use the AND output for the $spike - count$ and consider just the number of equal inputs for $spike - time$. This can be achieved by connecting the inputs to an XOR gate (which gives us the number of unequal inputs) followed by a NOT gate.

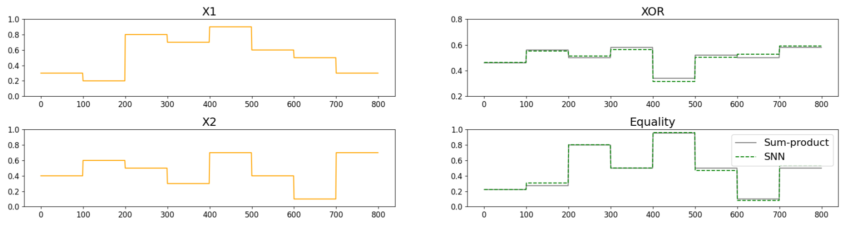

Figure 3: Comparison of results from the proposed SNN nodes for passing Bernoulli messages. X_1 and X_2 represent randomly selected parameters for the input Bernoulli messages. Each pair of inputs was sampled every 10 ms over a period of 100 seconds. The firing rate of the SNN gates was calculated as described in the text and is compared with those obtained using the sum-product formula.

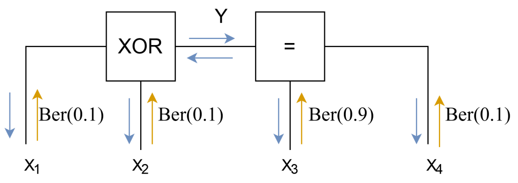

Figure 4: The FFG corresponding to the discussed example. Input messages are highlighted with orange arrows.

Message	Ground truth (11)	Our result
$\overrightarrow{\mu}(Y)$	$\mathcal{B}er(0.18)$	$\mathcal{B}er(0.174)$
$\overleftarrow{\mu}(Y)$	$\mathcal{B}er(0.5)$	$\mathcal{B}er(0.488)$
$\overleftarrow{\mu}(X_1)$	$\mathcal{B}er(0.5)$	$\mathcal{B}er(0.526)$
$\overleftarrow{\mu}(X_2)$	$\mathcal{B}er(0.5)$	$\mathcal{B}er(0.526)$
$\overleftarrow{\mu}(X_3)$	$\mathcal{B}er(0.024)$	$\mathcal{B}er(0.022)$
$\overleftarrow{\mu}(X_4)$	$\mathcal{B}er(0.664)$	$\mathcal{B}er(0.657)$

Table 1: Comparison of the ground truth messages in the example with the results obtained from our SNN-based factor nodes.

As calculated in the Appendix, the sum-product message according to Equation (1) follows

$$\overrightarrow{\mu}(y) = \mathcal{B}er(y \mid p_1 - 2p_1p_2 + p_2) \tag{2}$$

$$\overrightarrow{\mu}(y) = \mathcal{B}er(y \mid \frac{p_1p_2}{1 - p_1 + 2p_1p_2 - p_2}) \tag{3}$$

for the XOR and Equality nodes respectively. As shown in Fig. 3, the proposed SNNs yield results that closely match the numerical solution according to (2), and (3).

3.3 An example

We can now apply our proposed SNN-based message passing to solve an example. We used the problem introduced in (11). Consider the FFG depicted in Figure 4, which represents the binary code $C = \{(0,0,0,0),(0,1,1,1),(1,0,1,1),(1,1,0,0)\}$. In this example, the messages $(\overrightarrow{\mu}(X_1), \overrightarrow{\mu}(X_2), \overrightarrow{\mu}(X_3), \overrightarrow{\mu}(X_4))$, indicated by orange arrows on the graph, serve as the inputs. Table 1 compares the other messages passed along the edges with the results obtained from our SNN-based approach, demonstrating that our method closely matches the expected outcomes.

3.4 Conclusion

In this paper we implemented SNN-based message passing with Bernoulli messages on XOR and Equality factor nodes using logical gates. This result also aligns with the model proposed in (15), where continuous sum-product message passing was implemented using a Liquid State Machine (14). While their approach involved hundreds of synaptic weights, our goal in this paper is to implement the process as simply as possible. This simplification is crucial for our next step, which focuses on using self-learning STDP algorithms like those described in (5) and (6), to set synaptic weights in a way that is more biologically plausible. The ultimate aim is to implement the entire system on an MEA chip, paving the way for the development of a hybrid bio-computational chip.

Appendices

We can derive the forward sum-product message $\overrightarrow{\mu}(y)$ according to the XOR factor node, given Bernoulli input messages x_1 and x_2, as

$$\begin{aligned}
\overrightarrow{\mu}(y) &= \sum_{x_1}\sum_{x_2} \overrightarrow{\mu}(x_1)\, \overrightarrow{\mu}(x_2)\, f(y, x_1, x_2) \\
&= \sum_{x_1}\sum_{x_2} \mathcal{B}er(x_1|p_1)\, \mathcal{B}er(x_2|p_2)\, f(y, x_1, x_2) \\
&= \mathcal{B}er(0|p_1)\, \mathcal{B}er(0|p_2) f(y, 0, 0) + \mathcal{B}er(1|p_1)\, \mathcal{B}er(0|p_2) f(y, 1, 0) \\
&\quad + \mathcal{B}er(0|p_1)\, \mathcal{B}er(1|p_2) f(y, 0, 1) + \mathcal{B}er(1|p_1)\, \mathcal{B}er(1|p_2) f(y, 1, 1) \\
&= (1 - p_1)(1 - p_2) f(y, 0, 0) + p_1(1 - p_2) f(y, 1, 0) \\
&\quad + (1 - p_1) p_2 f(y, 0, 1) + p_1 p_2 f(y, 1, 1)\,.
\end{aligned}$$

Using the truth table, we can substitute y and evaluate the terms, such that

$$\begin{aligned}
\overrightarrow{\mu}(y) &= \begin{cases} p_1(1 - p_2) + (1 - p_1)p_2 & \text{if } y = 1 \\ (1 - p_1)(1 - p_2) + p_1 p_2 & \text{if } y = 0 \end{cases} \quad = \quad \begin{cases} p_1 - 2p_1p_2 + p_2 & \text{if } y = 1 \\ 1 - p_1 + 2p_1p_2 - p_2 & \text{if } y = 0 \end{cases} \\
&= \mathcal{B}er(y|p_1 - 2p_1p_2 + p_2)\,.
\end{aligned}$$

Since the XOR factor is symmetric according to their truth table forward and backward messages are equal. For instance, if we want the backward message $\overleftarrow{\mu}(x_1)$ given $\overrightarrow{\mu}(x_2)$ and $\overleftarrow{\mu}(y)$, it is:

$$\overleftarrow{\mu}(x_1) = \mathcal{B}er(x_1|p_2 - 2p_yp_2 + p_y)\,.$$

So we don't need any other gate for computing the backward messages and we can use the proposed gate but with the backward messages as the inputs.

Similarly, we have the following computations for the equality factor node:

$$\begin{aligned}
\overrightarrow{\mu}(y) &= \begin{cases} p_1 p_2 & \text{if } y = 1 \\ (1 - p_1)(1 - p_2) & \text{if } y = 0 \end{cases} \\
&= \mathcal{B}er\Big(y|\frac{p_1p_2}{1 - p_1 + 2p_1p_2 - p_2}\Big)\,.
\end{aligned}$$

References

[1] Doya, K.: Bayesian brain: Probabilistic approaches to neural coding. MIT press (2007)

[2] van Erp, B., Bagaev, D., Podusenko, A., İsmail, Ş., de Vries Bert: Multi-agent trajectory planning with NUV priors. American Control Conference (2024, in press)

[3] Friston, K., Kilner, J., Harrison, L.: A free energy principle for the brain. Journal of physiology-Paris **100**(1-3), 70–87 (2006)

[4] Goodman, D.F., Brette, R.: Brian: a simulator for spiking neural networks in python. Frontiers in neuroinformatics **2**, 350 (2008)

[5] Hem, I.G., Ledergerber, D., Battistin, C., Dunn, B.: Bayesian inference of spike-time dependent learning rules from single neuron recordings in humans. bioRxiv pp. 2023–04 (2023)

[6] Hu, Z., Wang, T., Hu, X.: An STDP-based supervised learning algorithm for spiking neural networks. In: International Conference on Neural Information Processing. pp. 92–100. Springer (2017)

[7] Isomura, T., Kotani, K., Jimbo, Y.: Cultured cortical neurons can perform blind source separation according to the free-energy principle. PLoS Computational Biology **11**(12), e1004643 (2015)

[8] Kagan, B.J., Kitchen, A.C., Tran, N.T., Habibollahi, F., Khajehnejad, M., Parker, B.J., Bhat, A., Rollo, B., Razi, A., Friston, K.J.: In vitro neurons learn and exhibit sentience when embodied in a simulated game-world. Neuron **110**(23), 3952–3969 (2022)

[9] Van de Laar, T.W., De Vries, B.: Simulating active inference processes by message passing. Frontiers in Robotics and AI **6**, 20 (2019)

[10] Lobov, S.A., Mikhaylov, A.N., Shamshin, M., Makarov, V.A., Kazantsev, V.B.: Spatial properties of stdp in a self-learning spiking neural network enable controlling a mobile robot. Frontiers in Neuroscience **14**, 88 (2020)

[11] Loeliger, H.A.: An introduction to factor graphs. IEEE Signal Processing Magazine **21**(1), 28–41 (2004)

[12] Loeliger, H.A., Dauwels, J., Hu, J., Korl, S., Ping, L., Kschischang, F.R.: The factor graph approach to model-based signal processing. Proceedings of the IEEE **95**(6), 1295–1322 (2007)

[13] Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural Networks **10**(9), 1659–1671 (1997)

[14] Maass, W.: Liquid state machines: motivation, theory, and applications. Computability in Context pp. 275–296 (2011)

[15] Steimer, A., Maass, W., Douglas, R.: Belief propagation in networks of spiking neurons. Neural Computation **21**(9), 2502–2523 (2009)