Master's Thesis in Artificial Intelligence

Radboud University Nijmegen

Hierarchical Bayesian Inference Implementation in Neuromorphic Hardware

MS Artificial Intelligence: Cognitive computing

Author:
Borislav Sabev
s4726863
b.sabev@student.ru.nl

Supervisor:
dr Johan Kwisthout
Associate professor
Dept. of Artificial Intelligence
Radboud University

Second Reader:
dr. Leila Bagheriye
Postdoc
Dept of Artificial Intelligence
Radboud Univeristy

June 14, 2023

Abstract

Neuromorphic hardware is a novel computer architecture based on a computational model of biological neurons that uses sparse, event-driven, distributed and stochastic computation to provide superior energy efficiency compared to traditional von Neuman architecture. In neuroscience, Bayesian inference is used as a theoretical framework for modelling brain processing and there are numerous models that attempt to explain how Bayesian Inference can be computed in a biologically plausible neural network. These models can be used as a basis to investigate efficient neuromorphic algorithms for Bayesian Inference. After brief survey of spiking neural network models for Bayesian Inference, the Hierarchical Bayesian Inference model by [12] is selected for implementation on the Loihi neuromorphic chip. The Hierarchical Bayesian Inference model uses Winner-take-all circuits as a building block to represent categorical distribution, which are further stacked in a hierarchical fashion to extract more complex features from input data and learn to cluster input by Spike-Timing Dependend Plastisity (STDP). Findings suggest that the integer precision of Loihi, paired with hardware restrictions on the size of neuron parameters proves to be insufficient to accurately represent conditional probabilities. Additionally, in the WTA circuit implementation the simulation time required to resolve a winner grows with the number of excitatory neurons, therefore the largest WTA circuit bottlenecks the entire model even when smaller circuits have already converged to a stationary distribution. As a consequence of the lack of synaptic precision and the poor probability approximation of larger WTA circuits, the current neuromorphic implementation fails to reproduce the results of Guo et al [12] on the MNIST benchmark for unsupervised classification of handwritten digits.

1 Introduction

Bayesian Inference

Bayesian inference is a statistical method for computing the posterior probability distribution of the model parameters given observed data and prior knowledge about the parameters. The posterior distribution represents the updated knowledge about the parameters after taking into account the observed data. Bayesian Inference is at the foundation of statistical machine learning [5] and has wide applications in science, engineering, philosophy, and more [11]

Formally Bayesian Inference is the computation of the posterior probability $P(\theta|D)$ for a set of observed data D and model parameters θ. The posterior distribution is given by Bayes' Rule $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$, where:

- $P(\theta|D)$ - the *posterior probability*, the updated parameters θ given D is observed
- $P(D|\theta)$ - the *likelihood* of observing the data under the model parameters θ
- $P(\theta)$ - the *prior probability* of the parameters
- $P(D)$ - *marginal likelihood*, the probability to observe the data under any model parameters

The marginal likelihood is given by $P(D) = \int P(D|\theta)P(\theta)d\theta$. This integral can be computationally expensive to evaluate, especially for complex models with many parameters. In the simplest case, specifying a joint probability distribution $P(\mathbf{X})$ over a set of binary random variables $\mathbf{X} = X_1, ...X_n$ would require 2^{n-1} numbers. The exponential scaling as a function of the random variables means that for most practical applications the storage and computation demands of probability distributions required to compute Bayesian Inference are impossible to meet [16]. In fact, exact Bayesian Inference is an NP-hard problem[6] and a polynomial time algorithm is unlikely to exist. Approximate methods of Bayesian Inference are NP-hard as well if a specific bound on the approximation error is required [7]. Bayesian Inference algorithms are still an area of active research and creating

faster and more efficient algorithms for different problems is highly desirable given the number of applications[11].

Bayesian Networks

The storage requirements of representing a joint probability distribution can often be reduced in practice by exploiting the (conditional) independence between random variables. For a probability distribution $P(X_1, X_2, ...X_n)$ where all random variables are independent (for any pair i,j $X_i||X_j$), $P(X_1, X_2, ...X_n) = P(X_1)P(X_2)...P(X_n)$, hence the storage and computational demands scale only linearly with the number of variables. In case of conditional independence, for any $X_i||X_j|Y$, $P(X_1, X_2, ...X_n|Y) = P(X_1|Y)P(X_2|Y)...P(X_n|Y)$. Using this rule and the chain rule of conditional probabilities, any joint probability distribution can be factorised into a set of conditional probability distributions (CPD). A Bayesian network (BN) is a way to represent a set of CPDs together with the conditional dependence of random variables in the joint distribution. A Bayesian Network (BN) consists of a directed acyclic graph, where nodes are random variables, edges represent dependency relations between those variables and a set of CPDs of the factorised joint distribution, one CPD per node[16].

BNs are used for three categories of tasks: inference of unobserved variables, parameter learning and structure learning. A query to the BN consists of a set of observed variables and a target unobserved variable for which to infer the posterior probability distribution. An example query on Sprinkler BN would be "If it's cloudy and raining, what is the probability that the grass is wet?", formally expressed as computing $P(G|C = T, R = T)$. Parameter learning is the process of learning the CPD of the variables from observed data. In many problems, using expert knowledge one can construct an appropriate graph to model the problem, but obtaining exact CPTs would be impractical either due to time/cost considerations and/or expert subjectivity in assigning the probabilities. In that case, the CPTs can be estimated based on observed data with algorithms such as Maximum Likelihood Estimation (MLE) or Bayesian Parameter Estimation [16]. Structure learning is employed when the graph of the Bayesian Network is unknown from the start [16]. While this is a large area of active research, it is also beyond the scope of the current thesis, which would cover only parameter learning and inference of unobserved variables.

1.1 Inspiration from the brain

In recent years taking inspiration from the brain has been productive for AI research[13]. While the thesis of Hassabis et al. [13] is focused primarily on the advances of Deep Learning and rate-based Artificial Neural Networks (ANN), research on Bayesian Inference can benefit as well since it is widely used as a model for brain computation. It is a commonly believed notion in neuroscience that the brain encodes uncertain information at least implicitly in probabilities [24] and integrates evidence in accordance with Bayes Rule [15]. Over a wide range of experimental tasks, humans behave in a Bayesian optimal way [17]. If the brain has some way to compute near-optimal Bayesian inference in a wide variety of tasks, taking inspiration from the brain can be productive to advance the field of Bayesian inference. Over the last years, there are many computational models that attempt to explain how Bayesian Inference can be computed by a biological neural network [18, 3, 21, 2, 14, 22, 12, 26], which would be discussed in depth in the "Bayesian Inference with SNN" subsection.

A common formalisation for a biological neural network is the Spiking Neural Network (SNN). SNN is a class of ANN that utilises temporal spike transmission between neurons as opposed to the rate-based coding of deep learning models. For the purpose of this thesis, I will discuss a discrete-time SNN model. An

SNN consists of n neurons and a number of uni-directional synapses that can exist between any two neurons. An SNN has time-varying dynamics that plays out in time steps $t \in \mathbf{N}$ with a discrete step of $\delta t > 0$. The neuronal model of choice is the Leaky Integrate-and-Fire (LIF), where neurons have two state variables, the membrane potential $v_i(t) \in \mathbf{R}$ and the current $u_i(t) \in \mathbf{R}$. Synapses have weights w_{ij} and if $w_{ij} \neq 0$, neuron i can send spikes to neurons j. At every step, for all neurons $u_i(t)$ and $v_i(t)$ are computed according to neural dynamics equation:

$$u_i(t) =_{j \neq i} w_{ij}(\alpha_u \cdot \sigma_j)(t) + b_i \quad (1)$$

$$v_i(t) = -\frac{1}{\tau_v} v_i(t) + u_i(t) - \theta_i \sigma_i(t) \quad (2)$$

where α_u is the response function, b_i is a biasing current, τ_v is the leakage and θ_i is the firing threshold. If $v_i(t) \geq \theta_i$, neuron i emits a spike and $v_i(t + \delta t) = 0$ [8].

Neuromorphic Hardware

Investigating the capabilities of SNN has recently become more important as neuromorphic hardware is reaching the development stage in which it can be used for practical applications [8, 9]. Neuromorphic hardware is a novel, brain-inspired computer architecture that is organised according to SNN as a computational model of the brain. As the model of SNN is directly encoded in the hardware for neuromorphic computing, they can act as natural accelerators, offering superior computational efficiency compared to simulating SNN on CPU [8]. In addition, neuromorphic computing utilises sparse, event-driven, spike-based interaction for distributed communication between neurons, which naturally provides superior energy efficiency compared to traditional computing[9]. Deep learning was not recognised to have practical applications until the Graphical Processing Unit (GPU) was utilised to speed up training by several orders of magnitude and since then it has had countless applications. Therefore it is not unreasonable to assume that SNNs would also spike in capability when dedicated hardware, orders of magnitude more efficient than CPU for SNNs, is commercially available.

The Loihi [8] developed by Intel is one of the first digital neuromorphic chips that allows for programming and executing flexible SNN models. The base unit of the chip is the Spiking Neural Unit (SNU), a hardware implementation of the LIF neuronal model. The dynamics of SNUs can be controlled with over 30 parameters such as response function, biasing current, leakage, firing threshold and reset voltage and more. SNUs are connected with synapses and each connection can have a learning rule that modifies synaptic strength on-chip. Loihi allows for any kind of SNN architecture, from feed-forward to models with recurrent connections.

1.2 Bayesian Inference with SNN

The early neuroscientific work on the Bayesian brain hypothesis focuses on ways to represent probability distributions in neural structures [27, 28, 15]. In the encoding-decoding framework [28] a population of neurons can encode a random variable through Probabilistic Population Coding (PPC). In PPC the neuronal dynamics of a population of neurons encodes a distribution over all possible states of a stimulus [28]. Every neuron is modelled as a Poisson spike generator, while collectively the firing patterns of the neurons can represent any continuous probability distributions from the exponential family [28]. In later work, populations of simulated Integrate-and-Fire neurons encoding stimuli in accordance with PPC are demonstrated to perform optimal Bayesian Inference in a cue integration task [18]. The PPC encoding enables Bayesian Inference in the modelling of decision tasks [3] and other complex inference tasks[2]. Beck et al. [3] demonstrate the first neural model for Bayesian Inference and decision-making that can optimally accumulate evidence and select from a set of possible decisions. The model can solve motion di-

rection tasks in biologically relevant timescales, continuously integrating sensory information per Bayes rule and selecting the maximum likelihood of the stimulus direction after a predefined time.

Previous examples of Bayesian Inference with PPC encoding demonstrate small models that are custom-made for the task and do not generalise. However, the PPC framework can be extended with learning through Variational Bayesian Inference [2]. The proposal distribution q needs to be from the exponential family to be encoded with PPC. The Variational Bayesian Expectation Maximisation algorithm optimises for a minimum of the Kullback-Leibler divergence between the proposed distribution q and the distribution of interest p. The algorithm is tested on a topic modelling task with large number of latent variables[2].

Another distinct line of research into Bayesian Inference with SNN is the neural sampling approach by Pecevski et al[21]. This approach is based on the mathematical theory that under specific conditions, the neural dynamics of an SNN can encode a model of a probability distribution [21]. Sampling in stochastic SNN can be modelled by a Markov chain, thus neural dynamics are guaranteed to converge to a given distribution p, effectively performing Markov Chain Monte Carlo (MCMC) sampling. For an SNN to sample from a distribution $p(z_1, ..., z_k)$, the neural computability condition (NCC) needs to be satisfied. If for each random variable z_i, the firing probability $p_i(t)$ is:

$$p(t) = \frac{1}{\tau} \cdot \frac{p(z_i = 1|z_{\sim i})}{p(z_i = 1|z_{\sim i})}$$

then NCC is satisfied [21].

Unlike PPC coding, with NCC random variables are discrete, as opposed to continuous, a random variable is represented by a single neuron, and spike timing rather than frequency determines the state of the random variable. While each random variable in the neural sampling model is encoded by a single neuron, auxiliary neurons that represent the interaction of states of random variables are required to satisfy the NCC. Assuming binary variables, a node with n parents has 2^n possible states to be conditioned upon, therefore requires at least $2^n - 1$ auxiliary neurons to satisfy NCC. Pecevski et al. [21] offer five different implementations of the model, each requiring an exponential number of auxiliary neurons relative to the maximum parent count in the Bayesian Network.

SNNs based on NCC satisfaction would suffer from the same convergence issues as MCMC methods. Since MCMC involve a Markov chain, each sample x^t is dependent on the previous state of the Markov chain x^{t-1}. Accurately estimating the posterior would require drawing effectively independent samples, which is not guaranteed for distributions with high co-variance convergence as results can heavily depend on the starting sampling step [1].

Pecevski et al [21] model is later extended in [4] to learn with Spike-Timing Dependent Plasticity (STDP). STDP is a common biologically plausible rule that adjusts the strength of the synapse based on the spike timing difference in the pre-synaptic and post-synaptic neurons. Generally, STDP is an unsupervised learning rule [20, 22, 19] and is computed using only the pre- and post-synaptic spike train. However, studies show that with small adjustments STDP can be applied in supervised learning by providing a teacher spike train as a label [25] or applied in reinforcement learning by augmenting the STDP computation with a global reward signal[10, 23].

Guo et al [12] propose a hierarchical model for Bayesian Inference and learning implemented in an SNN. The basic building block of the model is the WTA circuit as a neuronal population that can represent a discrete random variable, which can be stacked in a hierarchical organisation to extract higher-order statistics from the input. The model requires reducing an conditional dependencies to a single variable, thereby requiring a tree-structured Bayesian Network as input. Excitatory neurons in a WTA circuit G_k encode a random discrete variable $z_k = z_1, z_2, ..., z_k$

by having the spiking probability of neuron representing z_i to be equal to $p(z_i)$. Conditional probabilities between a child and a parent WTA circuit are encoded by one directional synaptic connections. The model allows for directly encoding the true conditional probabilities in the synapses, or learn from data the appropriate synaptic weights with STDP. Based on the mathematical framework of the study, WTA neurons firing based on input and STDP implement E-step and M-step of the variational expectation maximisation algorithm.

The hierarchical Bayesian Inference model is tested on the MNIST dataset [12]. The MNIST dataset consists images of digits 0-9 with the size of 28 x 28 pixels. In the input layer there are 28 x 28 = 784 WTA circuits, one for each pixel, which encode the probability of white or black pixel with 2 excitatory neurons per WTA (denoted as WTA K=2). The second layer consists of 16 WTA circuits with K=15, each taking input from a 7 x 7 patch of WTAs from the previous circuit. The final layer consists of a single WTA circuit with K=100, which serves as an output layer that assigns 1 out of 100 clusters to the input data. Even though classes in the dataset are only 10, with unsupervised learning it cannot be expected that 10 clusters are enough to cleanly separate classes. After the training the model, the 100 neurons in the output layer self-organise such that some of them are tuned to a specific digit and there is one consistent winner per class presented.

In the model by Guo et al. [12] can map only tree-structured Bayesian networks to a SNN. An arbitrary Bayesian Network can be converted to tree-structured one by combining the conditional probability tables of two or more parent nodes to the same child for all children nodes. This is not practical for graphs with high degree as the required memory to store fully merged conditional probability tables is exponential in the number of nodes. Additional limitation of the model is that because of the hierarchical structure and the clear input to output layer flow of information not all possible posteriors can be computed. As an example the model trained on MNIST computes the posterior of the digit class given input pixels, but it cannot compute the probability of pixel distribution give a class.

Another hierarchical model for Bayesian Inference is the Sampling-Tree Model (STM) [26]. As the name suggests, this model also works with tree-structured Bayesian Networks, but this time the model uses continuous variables encoded with the PPC framework. In STM a tree-structured Bayesian Network is divided into small network motifs of a single parent node and its children[26]. The authors demonstrate that separating a large network into small motifs allows for global inference on the network with only local, low dimensional importance sampling. Extending the model with more levels of hierarchy is also trivial due to the natural composability of the model. However, like in most other models, the neurons required scale exponentially with the degree of the input Bayesian Network.

Paper	Ma et al. 2006	Pecevski et al. 2011	Beck et al. 2012	Guo et al. 2019
Model Structure	3 node network	Bayesian network	Bayesian network	Tree
Probability Distributions (PD)	Gaussian	Binary	Exponential Family	Discrete
Neural Representation of PD	PPC	NCC	PPC	WTA circuit
Inference	Exact	Sampling	Variational	Variational
Learning	No	No	Supervised	Unsupervised

Table 1: Overview of the main discussed models for Bayesian Inference with SNN. Compares the requirements for input Bayesian network structure and probability distributions, the way probability is encoded in neurons, type of inference and learning paradigm.

1.3 Research Question

The main goal of this thesis would be to implement the hierarchical Bayesian Inference model by Guo at al. [12] on neuromorphic hardware. The original paper does all SNN computations on a CPU and while the author suggest their model can implemened in neuromorphic hardware, limitations and constraints of current neuromorphic chips can prove that a challenge. Therefore it is valuable to try to reproduce the model on neuromorphic architecture and discuss any pitfalls that may arise. The main advantages of the hierarchical Bayesian Inference model that make its further study worthwhile are its composability, general-purpose, and transparent encoding of marginal and conditional probability distributions and practical applications in machine learning. Compared to other models, the purpose of Ma et al. 2006 [18] and Beck et al. 2012 [2] is neuroscience modelling and comparison with behavioural experiments in which the efficiency of neuromorphic hardware might not be as important. The model by Pecevski et al. 2011 [21] can be used for Bayesian Inference on arbitrary Bayesian Networks, which could also lead to practical applications in computer science and machine learning. However, the model deals only with binary distributions. As proposed by Pecevski et al [21], their model can scale to any discrete distribution by using a WTA circuit, therefore implementing a robust WTA circuit on neuromorphic hardware aid that research line as well.

The main research question is:

> Can the results of the hierarchical Bayesian Inference model by Guo at al. [12] be replicated on the Loihi neuromorphic chip?

Sub-questions:

1. What are the limitations of WTA circuits as implemented in Loihi in terms of precision of probability and number of categories?
2. Can the model by Guo et al. [12] be used to represent a Bayesian Network, do inference and learn the conditional probability distributions from data?

2 Methods

2.1 Hardware and Software

The Loihi neuromorphic chip [8] and the accompanying software package NxNet are used for implementing the computational model for the current project. This section will highlight the features and limitations of the Loihi chip and the NxNet software as relevant for the following implementation.

Loihi is a digital chip that allows for the modelling of flexible SNN models and custom control over a variety of neuron and synapse parameters as well as the creation of custom on-chip learning rules. The base units of the chip are Spiking Neural Units which are based on the Leaky Integrate-and-Fire neuronal model. Spiking Neural Units can be connected with synapses and each connection can have a learning rule that modifies synaptic strength on-chip. NxNet is a Python programming package that can be used to describe an SNN model at the level of neurons and synapses and run the model on Loihi. Following are NxNet classes that give an overview of the capabilities to define an SNN for the Loihi chip.

1. **CompartmentPrototype** - specifies a prototype to be used by one neuron or population of neurons, it allows the definition of firing threshold, current and voltage decay, current noise, and a bias current.
2. **Compartment** - creates a population of neurons with the size of the population between 1 and 1024 based on a specific CompartmentPrototype.
3. **ConnectionPrototype** - specifies a prototype for synaptic connections. It allows for the definition of custom learning rules

(implementations of STDP), can set Connections to be excitatory (synapse weight ≥ 0) or inhibitory (synapse weight ≤ 0) or mixed (no sign restrictions).

4. **Connection** - creates synaptic connections between two Compartment objects based on a ConnectionPrototype and a matrix of synaptic weights.

These classes offer the basic capabilities available when building SNN models for the Loihi. Once an SNN is built from Compartments and Connections it can be compiled and run on Loihi. After successful simulation on-chip, the voltages, currents, and spike times of neurons and the value of synapses can be retrieved for data processing. The chip has 3 CPU cores for recording simulation data.

A notable limitation of Loihi concerning the current project is that the hardware precision uses a fixed point system, therefore only integer values for synaptic weight, firing threshold and current are possible. Since the model by Guo et al.[12] is tested on a traditional CPU, it is assumed that it had access to floating-point operations, the precision of the Loihi could prove to be a hindrance in reproducing the model.

2.2 Building block - WTA circuit

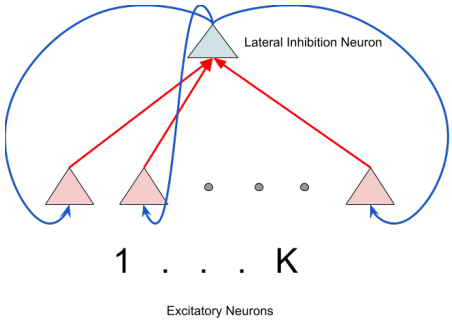

Figure 1: A WTA circuit with K excitatory neurons. The neurons stimulate the lateral inhibition neuron through excitatory synapses *(red)* and receive an inhibitory signal back *(blue)* from the lateral inhibition neuron.

The basic building block of the SNN model proposed by Guo et al.[12] is the Winner-Take-All (WTA) circuit. The role of the WTA circuit in this model is to encode the probability distribution of a given input and hierarchically stacking WTA circuits allows for more sophisticated probabilistic inference. A WTA circuit consists of K excitatory neurons and 1 lateral inhibition neuron, each excitatory neuron has an excitatory synapse towards the lateral inhibition neuron and receives an inhibition signal back (Figure 1). Excitatory neurons are connected to some form of input and the general pattern is that the neuron most sensitive to that input will spike first, which in turn suppressed other neurons from firing through the lateral inhibition neuron. Thus the neuron that spikes first is the winner. In [12], WTA circuits are used to represent a probability distribution over a categorical random variable by setting the membrane potential of the excitatory neurons such that the probability of firing represents the probability of the given category 1 to K.

A WTA circuit G_l encodes a probability distribution over a categorical random variable z_l where for every state z_l^k, $k = \{1...K\}$ there is exactly one excitatory neuron that encodes its

probability. WTA circuits can be further stacked and the parent WTA of G_l is referred to as $G_{parent(l)}$. In case a parent exists, the excitatory neurons of $G_{parent(l)}$ and G_l are fully connected with uni-directional synapses there the pre- and post- synaptic neurons are the ones of the parent and the child respectively. The synapses w_l^{ij}, $i = \{1...K_{G_l}\}, j = \{1...K_{G_{parent(l)}}\}$ encode the conditional probability $p(z_l = i|z_{parent(l)} = j)$. Since the firing probability of a neuron is an exponential function of the membrane potential, incoming synapses should encode the logarithm of the probability, hence $w_l^{ij} = \log p(z_l = i|z_{parent(l)} = j) + \hat{w}$, where $\hat{w}$ is a constant selected such that all $w_l^{ij} \in R^+$.

In the model by [12] uses WTA circuits with K=2, K=16 and K=100, so the WTA implementation needs to be robust for a wide range of K values. The NxNet implementation of the WTA uses two different compartment prototypes for the excitatory and the lateral inhibition neuron with two different settings for neuron parameters such as firing threshold and current decay. Excitatory neurons are connected to the lateral inhibition neuron with excitatory synapse, while the reverse connection is inhibitory, both synapses have a constant weight value.

Because the lateral inhibition neuron enforces competition between excitatory neurons, the overall spiking rate of the system is relatively constant [12]. That implies that for larger values of K, individual excitatory neurons on average will spike less frequently and too large of a current decay can completely shut down the neural dynamics until no excitatory neurons are firing despite external stimulation. This is confirmed in pilot tests with constant neuron parameter values, a WTA circuit would work as expected until $K < 30$ and for $K > 50$ excitatory neurons will stop firing in the presence of random input. As the instantaneous firing probability is proportional to the exponent of the current, $p_k(t) \sim e^{u_k(t)}$, the current decay is set to be $\sim 1/log(K)$ to scale appropriately for different values of K. Firing threshold is not found to change the dynamics of the WTA when varying K, so it is kept constant. NxNet allows for the addition of bias and noise currents, which add constant and random values respectively to the current at each timestep. By default they are non-zero values, but for this project they are set to 0 since excitatory neurons are expected to receive external stimulation from either data or child WTA circuits.

2.3 Toy problem with Bayesian Network

Before moving to the full model replication, the computational framework of Guo et al. [12] can be tested with a much simpler toy problem. To demonstrate that BN to SNN conversion is possible within this framework, the model needs to represent probabilities, infer probability distributions based on observations and learn from data. All these capabilities can be tested on a small Bayesian Network and for that purpose, the Sprinkler BN (Figure 2) is selected. The framework Guo et al. [12] requires tree-structured Bayesian Networks as to not have hidden conditional dependencies. The graph of Sprinkler is not a tree since the node *Wet Grass* has two parents, *Sprinkler* and *Rain*, therefore it needs to be converted by merging *Sprinkler* and *Rain* in a single node. Since *Sprinkler* and *Rain* are conditionally independent given *Cloudy*, the probability of the new combined node *Sprinkler-Rain* is the joint probability of the two conditional probabilities $P(Sprinkler|Cloudy)$ and $P(Rain|Cloudy)$.

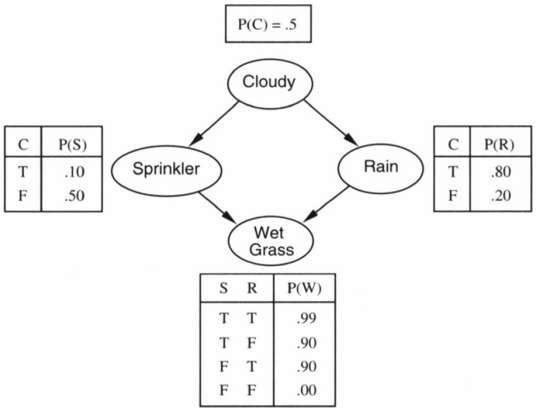

Figure 2: The Sprinkler Bayesian Network. It consists of only 4 binary variables, but it is sufficient to demonstrate Bayesian inference and learning from data.

Since for Sprinkler the conditional probabilities are known I can select to tackle only two of the required capabilities at a time, representation and inference, while leaving learning for a second experiment. The nodes *Cloudy*, *SprinklerRain* and *WetGrass* can be represented as WTA circuits with K=2, 4 and 2 respectively. Conditional probabilities are encoded in the synapses between excitatory neurons of a parent WTA and its child. As the *Cloudy* node has a uniform probability, the biasing current from its two nodes is a sufficient input to get the dynamics of the neural system started. The neural dynamics of the SNN would then act as a sampling mechanism and the expected result is that the average firing rate of a population representing a given node is proportional to the marginal probability of that node.

The learning capabilities of this computational model can also be tested with the Sprinkler Bayesian Network. Instead of directly encoding the correct conditional probabilities in the synapses of the SNN, the goal is to start from random weights and learn from the data.

The structure of the learnable SNN is identical as before, 3 WTA circuits one for each variable Cloudy, SprinklerRain and WetGrass and synaptic connections corresponding to the graph edges. The difference is that synaptic weights between different WTA-s are now randomly initialised from a normal distribution with $\mu = 3$ and $\sigma = 1$ and all negative values are set to 0. The mean of the Gaussian is selected such that 95% of the samples fall in the range of [0,5], which is the range of the synapses with encoded correct probabilities. Additionally, every inter-WTA synapse has STDP learning enabled. Synapses within the WTA-s themselves are not modified from the original setup and are still constant.

A dataset is necessary for the training phase of the SNN. The data is gathered by random sampling of the tree-structured Sprinkler network with the following procedure for N number of steps.

$$
\begin{aligned}
c_i &\sim P(Cloudy) \\
sr_i &\sim P(SprinklerRain|Cloudy = c_i) \\
wg_i &\sim P(WetGrass|SprinklerRain = sr_i)
\end{aligned}
$$

To prepare the data for input the SNN it needs to be converted to spike trains. Given a discrete random variable z with categories $\{z^1, z^2, ..., z^K\}$, sampled for N steps producing the dataset $(z_1, z_2, ...z_N)$, K spike trains are initialised where a neuron An example is illustrated in Figure 3.

The verification of successful learning can be done with two methods, first to test if neurons spike according to the true marginal probability, second to decode conditional probabilities from synapses and compare them with the original network. After training is successfully completed, the model should have comparable dynamics to the model in the first part with the encoded true weights. For the decoding part, we can apply the inverse of the encoding function to the synaptic weight to retrieve the encoded probability.

$$f(x) = \log x + \hat{w} \tag{3}$$
$$f^{-1}(x) = exp(x - \hat{w}) \tag{4}$$
$$p(z_l = i | z_{pa(l)} = j) = exp(w_l^{ij} - \hat{w}) \tag{5}$$

Here (3) is the encoding function from probability to synapse and from there (4) can be derived as the decoding function from synaptic weight to probability. Equation (5) is the decoding function substituted with the probability and the synaptic weight. Since the synapses are learned $\hat{w}$ is unknown and needs to be discovered.

$$p(z_l = i | z_{pa(l)} = j) = exp(w_l^{ij} - \hat{w})$$
$$\sum_i^K p(z_l = i | z_{pa(l)} = j) = \sum_i^K exp(w_l^{ij} - \hat{w})$$
$$\sum_i^K exp(w_l^{ij} - \hat{w}) = 1$$
$$\sum_i^K \frac{exp(w_l^{ij})}{exp(\hat{w})} = 1$$
$$\sum_i^K exp(w_l^{ij}) = exp(\hat{w})$$
$$\hat{w} = \log \sum_i^K exp(w_l^{ij})$$

Filling the result for $\hat{w}$ in (5) gives us decoding from synaptic weight to the conditional probability. Once the SNN is trained the encoded probability can be decoded and compared with the true probabilities from the Bayesian Network to establish the quality of the learning process.

2.4 Guo et al. 2019 Replication

The MNIST dataset is used in Guo et al. [12] to validate their computational model. MNIST consists of grey-scale images of handwritten digits 0-9, 60 000 train samples and 10 000 test samples. The dimension of each image is 28 x 28 pixels. To verify that the replication of the model is correct, the currently presented implementation needs to learn in an unsupervised manner to distinguish between classes of the MNIST set.

As the original dataset comes in pixel values ranging from 0-255 some preprocessing is necessary to make the input suitable for an SNN. In the original paper [12] values are binarised 0 to 1 and for each pixel a 200 Hz Poisson spike train lasing 150ms is created. The input layer of the SNN consists of 784 WTA circuits, one for each pixel, and each circuit has 2 excitatory neurons representing the values of 0 and 1. For each pixel, the Poisson spike train is fed to the excitatory neuron of the corresponding value, while the other excitatory neuron does not receive any external input. In the current implementation the binarisation is assumed to be based on the mean of the training dataset, as in a single value is taken as the mean of the whole training dataset and any pixel less or equal to the mean is set to 0, the rest to 1. 100 images were visually inspected before and after binarisation to ensure vital patterns for recognising digits are intact.

Binary pixels are then converted to Poisson spike trains. Poisson spike generation can be implemented by considering that given the firing rate r and the discrete time step Δt, the probability of firing at any given step is $r\Delta t$. As original paper uses $r = 200Hz$ and $\Delta t = 1ms$, therefore the probability of instantaneous firing is $P(fire) = 0.2$. At any timestep a random number $rand$ between 0 and 1 is sampled uniformly, if $rand <= P(fire)$ add a spike to the train, otherwise do not add anything. A spike train is sampled for every image and every pixel. Same as in the paper, Loihi and NxNet use by default the 1ms as a discrete time step Δt. After pre-processing the input has a shape of 60 000 x 784 x 2 x 150, 60k images, 784 pixels per image, 2 neurons per pixel encoding 0 or 1 for 150 ms of simulation time with 1ms time step. The input is encoded in spike generators, excluding the "empty" spike trains for excitatory neurons that code for the opposite of the current input pixel. Note that simulation time differs from clock time, as in 150 ms of simulation time

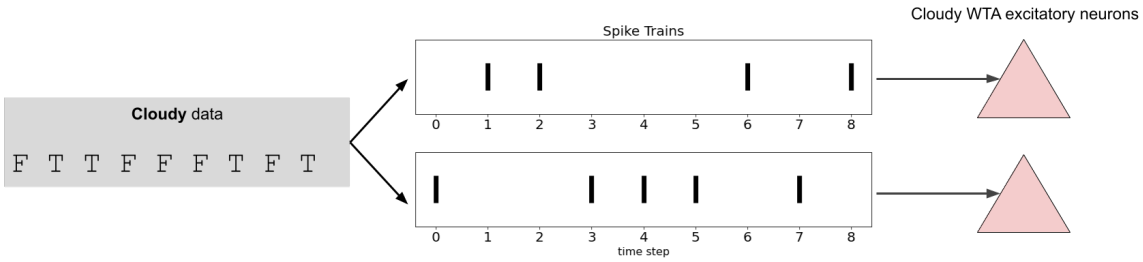

Figure 3: The process of going from sampled data to spike train and at the end neuronal input. Cloudy is a binary variable with True, False values. The data is then divided into two spike trains, one for each category, and a train spikes only if the value it represents is currently the sampled value. The spike trains are finally injected into the excitatory neurons of the WTA circuit that encodes the Cloudy node with sufficient synaptic strength to elicit a spike from reset threshold.

to present one image could take anywhere from nano seconds to seconds based on the complexity of the model.

The spike generators of the input are connected with static (no learning enabled) synapses to the input layer of the SNN. Input layer consists of 784 WTA circuits with K=2. The final model by Guo et al. includes in the hidden layer 16 WTA circuits with K=15. The input layer can be spatially divided into 4x4 cells, each cell containing 7x7 input WTA circuits, then each of the circuits in the hidden layer connects with a single cell. Within cell excitatory neurons are fully connected, for one cell there are 7x7x2 excitatory neurons pre-synaptic and 15 excitatory neurons of the hidden WTA post-synaptic. The synapses are uni-directional and have STDP learning enabled.

The output layer consists of a single WTA circuit with K=100. Excitatory neurons from the hidden layer fully connect to the output layer with both feedforward and feedback connections. In the original paper [12] it is not entirely clear on how and why feedback connections from output to hidden layers are implemented so some assumptions are necessary. First, it is assumed that feedforward and feedback connections are uni-directional and do not share weights as opposed to having one bi-directional synapse for every connection. The second assumption is that the feedback connections are meant to stabilise the representation in the hidden layer of the winner class in the output layer, so no learning is done on the feedback synapses. Additional reason to not real feedback synapses is that if STDP is enabled in both the feedforward and feedback synapses, the weights would naturally converge to inverses of each other and in the original paper the weights are constricted to positive space (is this warranted assumption?).

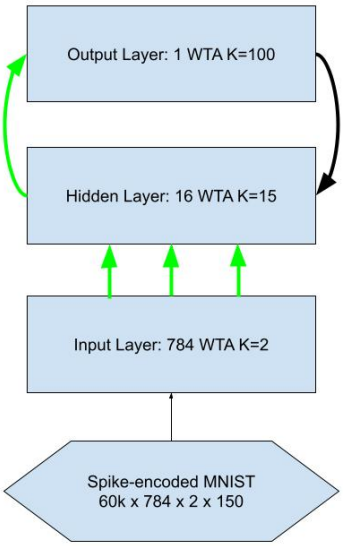

Figure 4: Diagram of the current implementation of the model by Guo et al. It consists of 3 layers and a data input spike generator. Arrows represent synapses and their direction, green synapses learn, while black do not. Input to hidden layer is not fully connected.

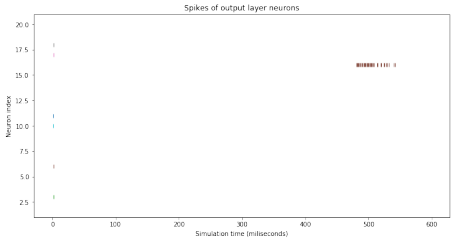

Figure 5: Simulated Data for Demonstrations. Expected spike data of the output layer. A sample is presented to the network starting at 450ms to 550ms and a single output neuron wins for that specific digit.

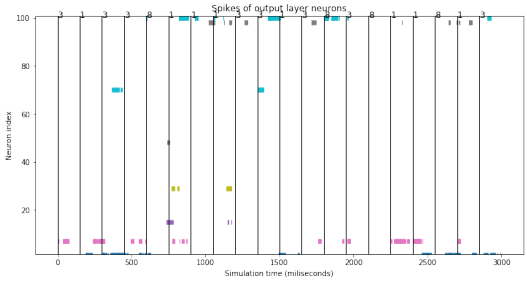

Figure 6: SNN output for 20 images each presented for 150ms. The circuit behaves like a WTA circuit as only 1 neuron wins at a given time, but it does not learn useful relations. Every segment on the graph, with a digit on top signifies which digit was presented during that interval. Neurons do not properly encode for specific classes

After the MNIST data is loaded into the spike generators and the three layers of the SNN are initialised as in Figure 4, the model can be compiled and run on the Loihi neuromorphic chip. The expected result of successful replication is that while initially the output layer fires more or less random, and eventually as training is done and synapses converge to a stable point for every class there is a specific dedicated neuron that fires whenever a sample from that class is presented to the SNN.

After training the result of 20 images sampled at random are plotted in 6. The output circuit works as a WTA circuit, but it does not select the winner at the proper times. For example the purple spike trains can be seen to activate for win for both digits 1 and 3. In an attempt to reduce the necessary representational power and ease the learning process, a second training session is done on only a subset of the data for digits 1,3 and 8. This does not improve the results, which would suggest that something fundamental is wrong with the implementation

of the model that prevents it from learning entirely. Since results are unexpected and the next section would be dedicated to possibly unjustified assumptions about the model and additional reasoning on how Loihi / NxNet implementation might differ from the implementation in the paper.

3 Results

The following section covers the results from running the three models described in the methods on Loihi. First, the Sprinkler Bayesian network is modelled by a SNN with the correct conditional probability tables encoded directly in the synapses. The aim is to investigate if WTA circuits implemented in Loihi can represent and compute with probabilities. Next, the same SNN starts with randomly initialised synapses and it is trained on data sampled from the Sprinkler to learn the proper synaptic weights for representing conditional probabilities in the network. The last subsection covers the results from the replication of Guo et al. [12].

3.1 Sprinkler network, no learning

The goal of the first model is to represent and compute with probabilities of the Sprinkler Bayesian network. The SNN model works only with tree-structured networks, so Sprinkler is converted to a tree-structured Sprinkler with 3 nodes, *Cloudy*, *SprinklerRain* and *WetGrass*. In the SNN model each node is represented by a WTA with two, four and two excitatory neurons respectively. The relative firing rate of excitatory neurons in the WTA represents the sampled probability of the variable corresponding to the WTA. Since the root node, Cloudy, has uniform probability, no external input is required to get the dynamics of the system started. The two excitatory neurons of the Cloudy WTA are identical and have a bias current that allows them to spike in absence of input.

The SNN is run for 10 000 simulation steps, each step presenting a different sample of Cloudy, SprinklerRain, WetGrass. On Figure 7 are presented raster plots for each WTA. In the leftmost plot for the Cloudy variable, neuron 0 accounts for 58% of the total spikes, so the probability sampled from the WTA is $P(Cloudy = 0) = 0.58$. Other distributions can be inferred in the same manner. On the rasterplots we can also observe the competition between neurons, in a period where one neuron is highly active, others are shut down. Out of all times a neuron spikes, two or more neurons spike at the same time less than 0.5% the time, which indicates that the dynamics of the WTA are correct.

To verify the accuracy of the SNN sampling, it can be compared with the exact solution for marginal probabilities (Figure 8). From the side-by-side comparison it is clear that the SNN fails to represent the correct probabilities. $P(C = 0) = 0.57$ is far from uniform distribution, and representational errors from the root of the Bayesian Network propagate to the other nodes. Analytically computed $P(WetGrass = 0) = 0.41$, while the SNN represents the value as $P(WetGrass = 0) = 0.59$. Kullback–Leibler divergence can be used to quantify how well the sampled spiking probability P_{snn} represents the analytically computed true probability P_{true}. The KL divergence, $D_{KL}(P_{true}||P_{snn})$, for the nodes Cloudy, SprinklerRain and WetGrass are 0.024, 0.26 and 0.08 bits (using $\log_2$ in the D_{KL} formula). $D_{KL}(P_{true}||P_{snn})$ represents the information loss when using P_{snn} to approximate P_{true} and an information loss of maximum 0.26 bits of the SprinklerRain node is relatively low.

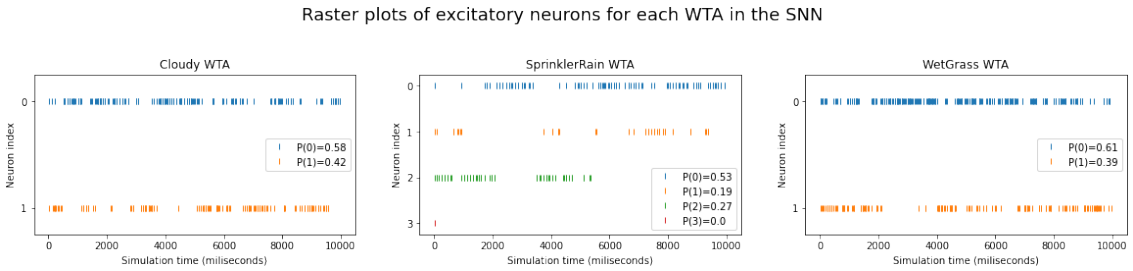

Figure 7: Spiking behaviour of the SNN (no learning) on the Sprinkler network problem. Each rasterplot represents a single node in the network represented by a WTA circuit. Marginal probabilities of the nodes are represented in by the relative firing rates of the neuron assigned to the state

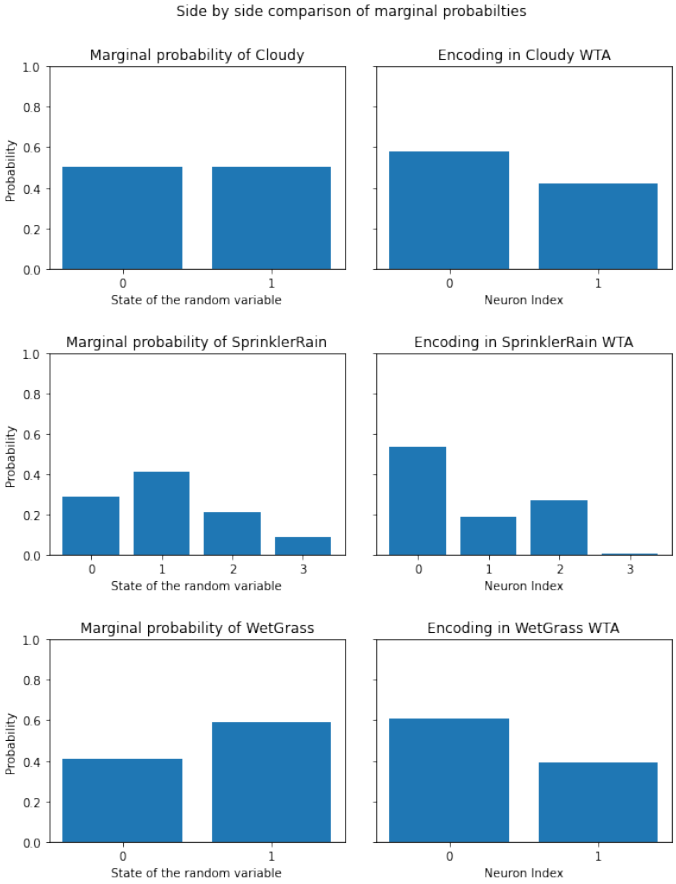

Figure 8: Marginal probabilities of each node in the tree-structured Sprinkler network, analytically computed (*left*) versus as encoded by the SNN (*right*)

3.2 Sprinkler network with learning

The previous section covered the results from an SNN where synapses representing conditional probabilities are directly set to the correct weight. In this subsection, an SNN is trained on sampled data from the Bayesian network to learn the synaptic connections between WTA circuits. Similar to the previous section the probability representation of the SNN can be compared with the accurately sampled data. In addition to that, intra-WTA synapses can be decoded to retrieve the represented conditional probabilities and compared with the conditional probability tables (CPT) of the Bayesian Network.

The SNN is trained from 10 000 steps on the randomly sampled data from the Bayesian Network. Each sample of the data contains a specific state for *Cloudy*, *SprinklerRain* and *WetGrass* and at each timestep corresponding to a sample the neurons in the SNN corresponding to that state are stimulated with external current. After training is completed, the model weights are frozen and the simulation is run for another 10 000 steps.

Similarly to the previous model, raster plot of the excitatory neurons (Figure 9) and a comparison of the probability distributions between random sampling and the SNN (Figure **??**) are presented. The SNN with learning also fails to represent the marginal probabilities at every node of the Bayesian Network. As expected, the results from Cloudy WTA are similar in both the learning and no-learning condition, as the excitatory neurons in Cloudy WTA are not affected by plastic synapses. The KL divergence, $D_{KL}(P_{true}||P_{snn})$, for the nodes Cloudy, SprinklerRain and WetGrass are 0.023, 0.25 and 0.20 bits respectively respectively. The information loss from approximating P_{true} with P_{snn} is again low, with the highest value of 0.25 bits for the SprinklerRain node.

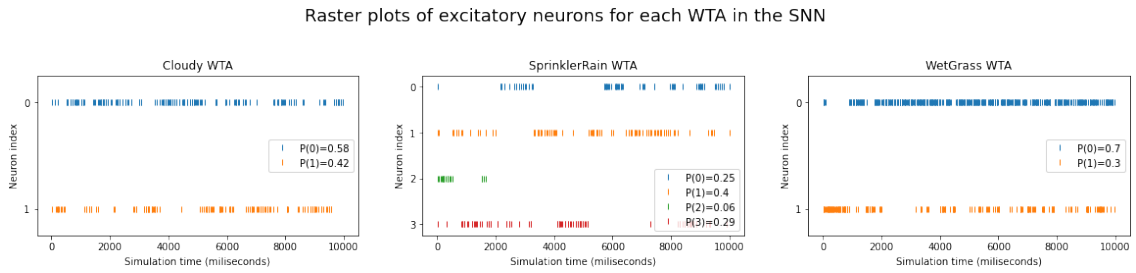

Figure 9: Raster plots of the WTA circuits in the SNN on the Sprinkler network. Similar to Figure 7, except here synaptic weights are learned from data with STPD.

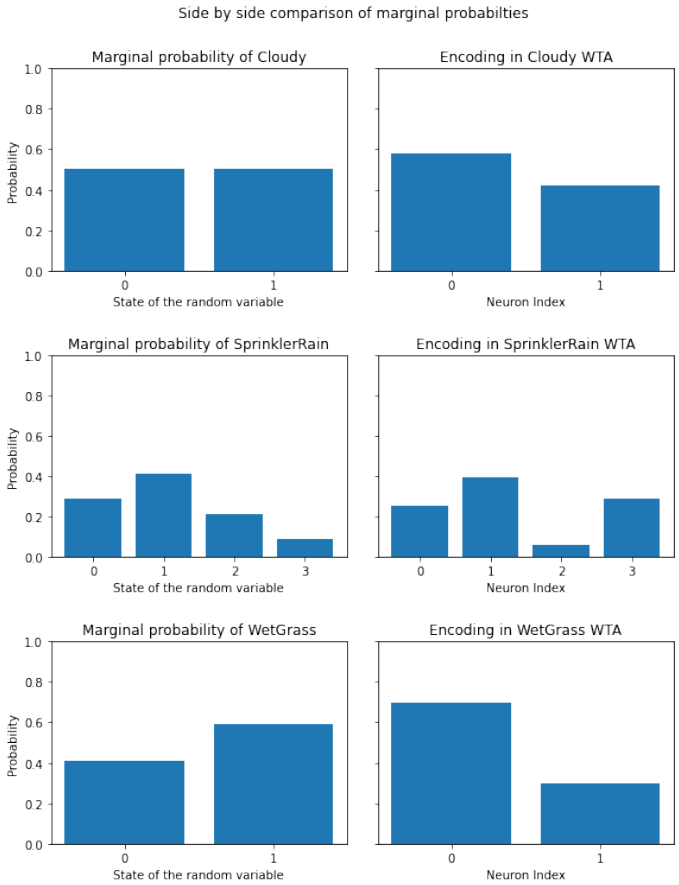

Figure 10: Marginal Probabilities of each node in the Sprinkler network, comparison between standard sampling and sampling with WTA circuits in a SNN.

Synapses between WTA circuits encode conditional probability of the post-synaptic WTA given the values of the pre-synaptic WTA. The weights of the synapses are decoded and compared with the true conditional probabilities from the Sprinkler network in Table 1. Likely due to the low precision of the synapses, they tend to encode extreme values as most probabilities are either close to 0 or 1. From the comparison of the marginal probabilities in Figure 10 it can be seen that a small bias towards *not Cloudy* leads to a very large increase in probability for option 0 that represents *not Cloudy* and *not Rainy*.

To illustrate the problem with integer synaptic weights we can review some example weights and see how they will be decoded into probabilities (Table 2). As it can be seen on Table 3, if only using integer weights values tend to be extreme and there is no middle ground between uniform distribution and a $0.73 : 0.27$ distribution.

	True Probability	Synaptic Encoding of P
P(S=0, R=0 | C=1)	0.1800	0.0003
P(S=0, R=1 | C=1)	0.7200	0.9997
P(S=1, R=0 | C=1)	0.0200	0.0180
P(S=1, R=1 | C=1)	0.0800	0.0003
P(S=0, R=0 | C=0)	0.4000	0.9820
P(S=0, R=1 | C=0)	0.1000	0.0000
P(S=1, R=0 | C=0)	0.4000	0.0000
P(S=1, R=1 | C=0)	0.1000	0.0180
P(W=1 | S=0, R=0)	0.0500	0.5000
P(W=1 | S=0, R=1)	0.8000	0.9970
P(W=1 | S=1, R=0)	0.8000	0.5000
P(W=1 | S=1, R=1)	0.9000	0.9820

Table 2: Comparison of the true conditional probabilities of the Bayesian Network and the conditional probabilities decoded from the learned synaptic connections. Some of the encoded conditional probabilities are off by nearly 50%. The probabilities derived from the synaptic encoding tend to assume values close to 0, 1 or 0.5

w	P(A=0 | B=b)	P(A=1 | B=b)
$[w_1, w_2 = w_1]$	0.5	0.5
$[w_1, w_2 = w_1 + 1]$	0.269	0.731
$[w_1, w_2 = w_1 + 2]$	0.119	0.881
$[w_1, w_2 = w_1 + 3]$	0.048	0.952

Table 3: Example values for synaptic weights and how they will be decoded with the inverse of the encoding by Guo et al. [12], Starting with $w_1 = w_2$ on the first row, each following row increments w_2 by 1, the smallest increment since Loihi uses 8-bit integers to encode synaptic values. Loihi cannot represent any conditional probability in between those values with the encoding scheme by [12].

3.3 Hierarhical Bayesian Inference replication

For the replication of Guo et al. the model is trained on the MNIST training set of 60 000 images, 150 timesteps per image, then tested on the 10 000 image test set. If the implementation work the expected output is that given an input of a specific digit, there will be one specific neuron that activates and wins the activation consistently for that digit. Rasterplot of the activation for 20 random images of the test set shows that this is not the case. Overall, some neurons tend to be more active than others, but there seems to be no clear correspondence between a digit and a winner neuron.

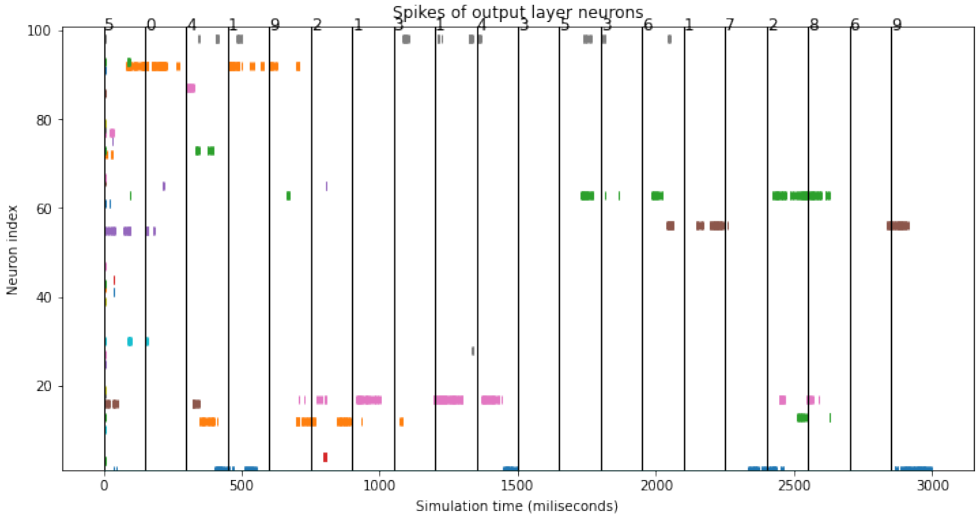

Figure 11: Raster plot of the output layer for 20 images. Vertical lines separate the different samples, with the label presented at the top of the plot.

The spiking data can also be analysed by finding the relative spiking frequency of neurons for specific digits. Data is grouped by class label and for each label, for all instances of that class, the relative time that a specific neuron is the "winner" of the activation can be recorded. For a working model, we would expect for every class label there is a specific neuron that wins 90-100% of the time. Here neuron 0 has the highest firing rate for any digit present and wins overall 30% of the time any digit is present (Figure 12). For every digit, the top 5 spiking neurons come out of a subset of only 8 neurons, indexed 0,15, 69, 78, 84, 97, 62, 9.

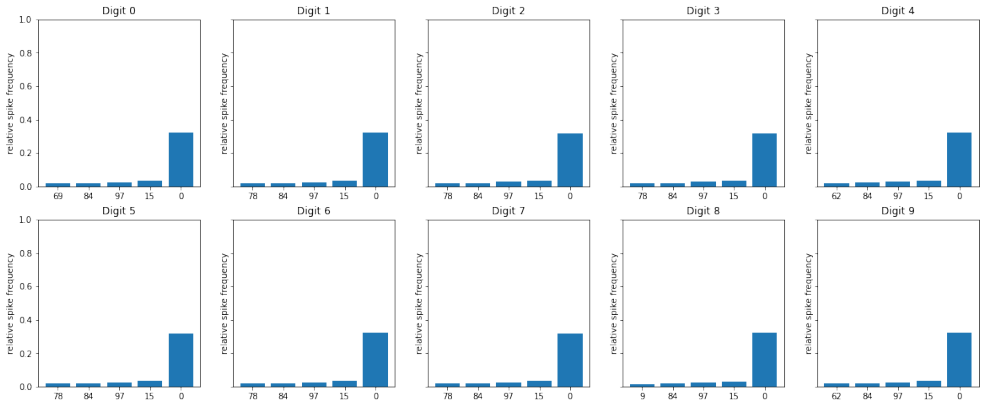

Figure 12: For every digit in the dataset, present the top 5 neurons that spike the most when that digit is presented.

The fact that neuron 0 is the most often spiking neuron might not be a random coincidence. In the previous 2 models, 5 out of 6 WTA circuits show a bias towards neuron 0 spiking. This may be indicative of how Loihi resolves membrane potential changes that happen in the same time step.

4 Discussion

4.1 Assumptions of the implementation

To distinguish between the specification of the model as described in Guo et al. [12] and the current implementation here is provided with a list of assumptions to fill in the gap in the computational model described in the paper:

- Neuron hyperparameters such as bias, current, firing threshold, leakage, and noise differ between excitatory and lateral inhibition neurons. Justification: initial values are taken from NxNet tutorial published by Intel on WTA circuits, the hyperparameters are then adjusted to work with a higher number of excitatory neurons.

- The excitatory neurons in the input WTA circuits are implemented as LIF neurons and external current (NxNet spikeGenerator) is applied to the neurons to simulate input. One sentence in the paper suggests that it could be that the excitatory neurons in the input WTA circuits are the spike generators themselves: "The binarised value is converted to Poisson-spike with firing rates 200 Hz of the corresponding excitatory neuron, whereas the other neuron is inhibited and keeps silent." a direct quote from [12]. However, if that is the case then it would not make sense to have a WTA circuit on the input layer if the spiking of the excitatory neurons is strictly predetermined by the input.

- Feedforward and feedback connections between the hidden and the output layer do not share synapses and weights. Additionally, feedback connections do not learn. Justification: feedback connections would learn the inverse of feedforward connections because of how STDP works and the authors clip all weights in the positive range. From that, it follows that either the feedback or the feedforward connection would go to 0.

- Hyperparameters of the STDP learning. There are several variations of STDP learning based on the exact parameters used.

Loihi supports e-STDP and provides the possibility to specify the rate of synaptic change relative to voltage, while in the original paper, the authors describe a more complex STDP with a double exponential window.

- There is no need for a mechanism that resets the voltage and current of the network after every example. The authors do not mention if they use such a mechanism, but given my results, it is possible that the network confuses when one sample end and the next begin.

4.2 Hardware Precision and Representing Conditional Probabilities

One of the main challenges of implementing the computational framework by Guo et al. [12] in the Loihi neuromorphic chip has proven to be the fixed point precision of the hardware. Having access to only integer synaptic weights severely limits the representation of conditional probabilities in the synapses. Given a random discrete variable $z_l^k k = 1, ..K$ with probability that is conditioned on $z_{parent(l)}^{k'}$, the synaptic weight between z_l^k and $z_{parent(l)}^{k'}$ is encoded as: $w_l^{ij} = \log p(z_l = i|z_{parent(l)} = j) + \hat{w}$, where $\hat{w}$ is a constant. When this is translated to Loihi, since w_l^{ij} is a non-integer value, it is floored by removing anything after the decimal point to fit in the fixed-point hardware system. This leads to large ranges of the continuous value for $p(z_l = i|z_{parent(l)} = j)$ to be represented by identical weight. In Figure 13 it can be seen that all probabilities in the range of $p \in [0.35, 1)$ are represented in synapses by a single value, $w = 4$, severely limiting the representational power of the system.

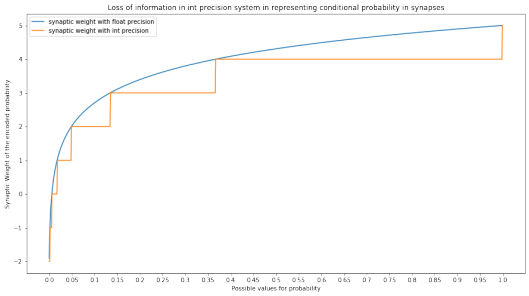

Figure 13: Comparison between theoretical synaptic encoding and the encoding in integers for all ranges of $p \in [0, 1]$. With integer precision, a wide range of probabilities is treated as identical by the encoding.

If the neuromorphic system allows for higher precision, the issues of representation quickly become alleviated. Figures 14 and 15 presented similar comparisons, except that now the encoding system can compute with decimals until the 1st and the 2nd significant digit respectively. A system using decimals with 2 significant digit can encode probabilities with minimal loss in information as the two lines for float precision encoded synapses and 2 significant digit encoding overlap. Empirical tests with higher precision decimals show that to encode probabilities up to n-significant digit, it is necessary for synapses also to be represented in the hardware up to the n-significant digit.

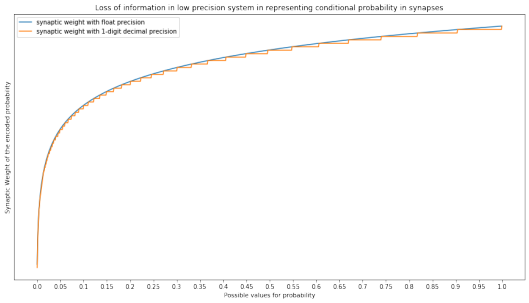

Figure 14: 1st significant digit decimal synaptic encoding versus 32bit float synaptic encoding for all ranges of $p \in [0, 1]$.

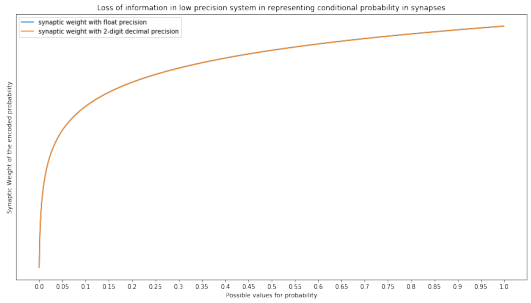

Figure 15: 2st significant digit decimal synaptic encoding versus 32bit float synaptic encoding for all ranges of $p \in [0, 1]$.

Another noteworthy property of the precision of such encoding is that it is more precise for values in the lower range that tend towards 0, while having the poorest precision for numbers that approach 1. This property stems from the shape of the logarithm function. This can be relevant if a specific problem domain works mostly with low-likelihood events.

It is important to note that the $\hat{w}$ constant does not change the precision of the system, but non-integer $\hat{w}$ can be used to shift the range of probabilities p that are represented by a single synaptic weight.

One way to implement an n-significant digit precision system (where n is a small number) in a fixed-point system is to simply scale relevant numbers in the system by 10^n. In the current project, higher synaptic precision can be achieved by multiplying all synapses by 10^n, which in turn would require scaling of the firing threshold of neurons, their current decay, bias, noise and the step change in STDP learning. Here the Loihi chip imposes another restriction which makes the scaling non-trivial, as current decay is limited to only values between 0 and 4096 and the lateral inhibition neuron before scaling already uses 4095 current decay to ensure a small "memory" of inputs that are insufficient to cause a spike.

Precision limitations can also be resolved on a hardware level with floating-point arithmetic (FP). FP allows to trade-off range for precision. For example synapses in Loihi are represented by an 8-bit integer, which allows for encoding whole numbers from 0 to 255 (signed integer) or -128 to 127 (unsigned integer). The same 8 bits can be divided between two numbers x and e which represent the number $x2^e$. The more bits are allocated to e, the higher precision will be as a trade off for the range.

4.3 Representing categorical probability distributions with WTA

WTA circuits offer a simple framework for representing a categorical probability distribution. Every excitatory neuron is assigned a category to represent and the probability of a category p_i is taken as the number of spikes of neuron i divided by the number of spikes of all excitatory neurons. The competition between neurons enforces near constant energy requirements per timestep of simulation. Trainable synapses between the input and the excitatory neurons allow for unsupervised clustering as demonstrated by Guo et al.

However, as WTA circuits are implemented on neuromorphic hardware, numerous challenges arise.

5 Conclusion

In the introduction, it is established a link between neuromorphic hardware and the neuroscience modelling of cortical processing with Bayesian Inference, arguing that neuromorphic chips can potentially compute fast and energy-efficient Bayesian Inference. To explore this idea, the aim of this thesis is to replicate a recent spiking neural network model computing Bayesian inference [12] and investigate pitfalls that might arise in implementation on the Loihi neuromorphic chip. The results from the original paper on unsupervised clustering on MNIST are not reproduced, and the Loihi implementation does not produce meaningful results on cluster-

ing. To investigate possible problems with the implementation I zoomed in on a single WTA circuit, the building block of Guo et al model, and the connection between two WTA circuits. The main roadblock seems to be the limited synaptic precision, as Loihi uses 8 bit integers to represent synaptic weights. Fixed-point or floating-point athletics on hardware level can dramatically improve the precision of representing conditional probabilities in synaptic weights.

References

[1] Alex M. Andrew. *Information Theory, Inference, and Learning Algorithms*, volume 33. 2004.

[2] Jeff Beck, Katherine Heller, and Alexandre Pouget. Complex inference in neural circuits with probabilistic population codes and topic models. *Advances in Neural Information Processing Systems*, 4:3059–3067, 2012.

[3] Jeffrey M. Beck, Wei Ji Ma, Roozbeh Kiani, Tim Hanks, Anne K. Churchland, Jamie Roitman, Michael N. Shadlen, Peter E. Latham, and Alexandre Pouget. Probabilistic Population Codes for Bayesian Decision Making. *Neuron*, 60(6):1142–1152, 2008.

[4] Johannes Bill, Lars Buesing, Stefan Habenschuss, Bernhard Nessler, Wolfgang Maass, and Robert Legenstein. Distributed Bayesian computation and self-organized learning in sheets of spiking neurons with local lateral inhibition. *PLoS ONE*, 10(8):1–51, 2015.

[5] Christopher Bishop. *Pattern Recognition and Machine Learning*. 2006.

[6] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[7] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[8] Mike Davies, Narayan Srinivasa, Tsung Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1):82–99, 2018.

[9] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A.Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R. Risbud. Advancing Neuromorphic Computing with Loihi: A Survey of Results and Outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

[10] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9(JAN2016), 2015.

[11] Haipeng Guo and William Hsu. A Survey of Algorithms for Real-Time Bayesian Network Inference. *Papers from the AAAI Workshop on Real-Time Decision Support and Diagnosis Systems*, (1):1–12, 2002.

[12] Shangqi Guo, Zhaofei Yu, Fei Deng, Xiaolin Hu, and Feng Chen. Hierarchical Bayesian inference and learning in spiking neural networks. *IEEE Transactions on Cybernetics*, 49(1):133–145, 2019.

[13] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-Inspired Artificial Intelligence. *Neuron*, 95(2):245–258, 2017.

[14] David Kappel, Stefan Habenschuss, Robert Legenstein, and Wolfgang Maass. Network Plasticity as Bayesian Inference. *PLoS Computational Biology*, 11(11):1–31, 2015.

[15] David C. Knill and Alexandre Pouget. The Bayesian brain: The role of uncertainty in neural coding and computation. *Trends in Neurosciences*, 27(12):712–719, 2004.

[16] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*, volume 1. 2009.

[17] Konrad Paul Kording. Bayesian statistics: Relevant for the brain? *Current Opinion in Neurobiology*, 25:130–133, 2014.

[18] Wei Ji Ma, Jeffrey M. Beck, Peter E. Latham, and Alexandre Pouget. Bayesian inference with probabilistic population codes. *Nature Neuroscience*, 9(11):1432–1438, 2006.

[19] Timothée Masquelier and Simon J. Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Computational Biology*, 3(2):0247–0257, 2007.

[20] Bernhard Nessler, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. Bayesian Computation Emerges in Generic Cortical Microcircuits through Spike-Timing-Dependent Plasticity. *PLoS Computational Biology*, 9(4), 2013.

[21] Dejan Pecevski, Lars Buesing, and Wolfgang Maass. Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons. *PLoS Computational Biology*, 7(12), 2011.

[22] Dejan Pecevski and Wolfgang Maass. Learning probabilistic inference through spike-timing-dependent plasticity. *eNeuro*, 3(2):8616–8620, 2016.

[23] Rajesh P.N. Rao and Terrence J. Sejnowski. Spike-Timing-Dependent Hebbian Plasticity as Temporal Difference Learning. *Neural Computation*, 13(10):2221–2237, 2001.

[24] Adam N. Sanborn and Nick Chater. Bayesian Brains without Probabilities. *Trends in Cognitive Sciences*, 20(12):883–893, 2016.

[25] Peter Suma. Biologically Plausible Cortical Hierarchical-Classifier Circuit Extensions in Spiking Neurons. *ArXiv*, 2017.

[26] Zhaofei Yu, Shangqi Guo, Fei Deng, Qi Yan, Keke Huang, Jian K. Liu, and Feng Chen. Emergent Inference of Hidden Markov Models in Spiking Neural Networks through Winner-Take-All. *IEEE Transactions on Cybernetics*, 50(3):1347–1354, 2020.

[27] Richard S. Zemel and Peter Dayan. Combining probabilistic population codes. *IJCAI International Joint Conference on Artificial Intelligence*, 2:1114–1119, 1997.

[28] Richard S. Zemel, Peter Dayan, and Alexandre Pouget. Probabilistic Interpretation of Population Codes. *Neural Computation*, 10(2):403–430, 1998.