Simulation of Bayesian Learning and Inference on Distributed Stochastic Spiking Neural Networks

Khadeer Ahmed, Amar Shrestha, Qinru Qiu
Department of Electrical Engineering and Computer Science, Syracuse University, NY 13244, USA
Email {khahmed, amshrest, qiqiu} @syr.edu

Abstract— **The ability of neural networks to perform pattern recognition, classification and associative memory, is essential to applications such as image and speech recognition, natural language understanding, decision making etc. In spiking neural networks (SNNs), information is encoded as sparsely distributed train of spikes, which allows learning through the spike-timing dependent plasticity (STDP) property. SNNs can potentially achieve very large scale implementation and distributed learning due to the inherent asynchronous and sparse inter-neuron communications. In this work, we develop an efficient, scalable and flexible SNN simulator, which supports learning through STDP. The simulator is ideal for biologically inspired neuron models for computation but not for biologically realistic models. Bayesian neuron model for SNNs that is capable of online and fully-distributed STDP learning is introduced. The function of the simulator is validated using two networks representing two different applications from unsupervised feature extraction to inference based sentence construction.**

I. Introduction

The emerging field of neuromorphic computing is offering a possible pathway for approaching the brain's computing performance and energy efficiency for cognitive applications such as pattern recognition, speech understanding, natural language processing etc. Currently more and more complex problems are being attacked using machine intelligence and deep learning concepts. Fundamentally these are approaches where the application learns to perform a function instead of being programmed. The capability of learning is important because of the vast parameter space one has to explore, which is impossible to handle through the programming approach. Many strides in the field have been made from the inspiration of how the brain solves very complex problems in the most efficient manner.

The processing capability of brain comes from the collective processing abilities of simple processing components i.e., neurons. Interconnected neurons form the basis of neural networks. The ability of neural networks to perform pattern recognition, classification and associative memory, is essential to applications such as character recognition, speech recognition, sensor networks, decision making etc. [1] [2] [3] [4] [5]. Spiking neural networks (SNNs), which use spikes as the basis for communication, is the third generation of neural networks inspired by biological neuron models [6]. SNNs are capable of representing much richer information as it incorporates relative spike timing along with the neuron state and the synaptic weights for computation.

This work is partially supported by the National Science Foundation under Grants CCF-1337300.

There are many kinds of neural network architectures and learning algorithms proposed. For example Auto encoder, multilayer perceptron, deep learning, convolutional neural network etc., these are all computationally intensive when compared to SNN. The SNN has the potential to be very efficient as each neuron works asynchronously in an event-driven manner and with sparse spiking pattern. Moreover, fully distributed Spike Timing Dependent Plasticity (STDP) learning [7] can only be achieved on SNNs, which updates synaptic weight based only on local information of individual neuron. The emerging stochastic SNN that generates spikes as a stochastic process is not only more biologically plausible [8] but also enhances unsupervised learning and decision making [9] [10]. It further increases the fault tolerance and noise (delay) resilience of the SNN system since the results no longer depend on the information carried by individual spike but the statistics of a group of spikes.

Several studies have been performed to confirm that most of the perceptual and motor tasks performed by the central nervous system are stochastic in nature, which can be modeled in a Bayesian framework [11] [12] [13] [14]. The decision making process involves combining the priors with noisy information to compute predictions. SNNs built using these models have shown to perform inference based decision making. The emergent behavior of the model is not derived by mimicking the biological process in the neuron, but instead model the observed computational behavior. Hence these models are non-biologically realistic. There are several spiking neural network simulators available which support biologically realistic neuron models for large scale networks [15] [16] [17] [18]. However, there are no simulation tools available which can handle large scale SNNs with non-biologically realistic and mixed neuron models in an efficient manner. In this work we propose a simulation tool architecture to enable development of SNNs with non-standard network topologies and neuron models. The network topology can be developed in traditional fashion with stacks of neuron layers or with any arbitrary topology including simulation of complex networks consisting of sub networks with arbitrary recurrent connections. We address the limitation while modeling these networks by enabling functionality to support stochastic behavior, making synapse modeling uniform so the neural computation is similar for inhibitory and excitatory case and enabling non centralized control for neuron behavior.

The main contributions of this work are summarized as following.

1. We first introduce a general-purpose Bayesian neuron model for stochastic SNNs which is capable of in-system and fully-distributed STDP learning. The Bayesian neuron features stochastic firing using spikes,

which enables efficient Bayesian inference and maximum likelihood learning. The inclusion of both excitatory and inhibitory functions in these neurons naturally normalizes the firing rate across a set of neurons in a distributed manner. The uniform synapse connection and distributed learning capability of the proposed model enable the large-scale parallel implementation.

2. A highly scalable, flexible and extendable simulation platform was developed that explores above mentioned properties. The proposed simulator design is capable of performing online learning using STDP principles.

Comprehensive experimental results on two different applications: unsupervised feature extraction, and inference-based sentence construction, have demonstrated the effectiveness of the Bayesian neuron model and the simulator. In the rest of the paper we discuss the proposed neuron models and the network topology that accompanies these models which are used in building our SNNs followed by the parallel architecture for simulator implementation and finally we present the experiments.

II. Related Work

Majority of the neuron models used in existing SNNs are not stochastic. Active dendrite and dynamic synapse with an integrate and fire neuron model is proposed for character recognition [1]. Spiking self-organizing maps using leaky integrate and fire neurons for phoneme classification is presented in [2]. They use this model to account for temporal information in the spike stream. Work presented in [4] uses Siegert approximation for integrate and fire neurons to map an offline trained deep belief network onto an event-driven spiking neural network suitable for hardware implementation. They demonstrate that the system is able to recognize digits in the presence of distractions and noise.

A large-scale model of a hierarchical SNN that integrates a low-level memory encoding mechanism with a higher-level decision process to perform visual classification task in real-time is implemented [5]. They model Izhikevich neurons with conductance-based synapses and use STDP for memory encoding. Stochastic nature in spike patterns has already been found in lateral geniculate nucleus (LGN) and primary visual cortex (V1) [8]. Ignoring the randomness in neuron model not only limits its effectiveness in sampling and probabilistic inference related applications [15] [16], but also reduces its resilience and robustness. This paper presents a STDP learning-enabled stochastic SNN for high noise tolerance.

Majority of available SNN simulators focus on biologically realistic neuron models, performing operational simulations and behavior characterizations. The NEURON simulation environment is primarily based on biologically realistic empirical models of neurons [15]. The GENESIS neural simulation system is another tool developed for computational neuroscience [16]. It also implements biologically realistic neuron models for understanding biological networks and artificial neural networks. Brian is a Python based simulator [17], which supports rapid development of single-compartment neuron models and other complex models by defining the underlying behavior using differential equations. This to an extent supports non-standard neuron models for simulation by providing functionality to model non biologically realistic operations. NEST is another simulator for spiking neural network which focusses on dynamics and scale of the network rather than the exact morphology of the individual neuron [18]. This tool is efficient for scalable simulation of biologically realistic neuron models. In general these simulation tools compute differential equations in the neuron models which is computationally expensive. An event-driven simulator which exploits the sparse nature of neuron spikes to pre-compute look-up tables for characterizing synaptic and neuronal dynamics is implemented which improves simulation speed [19]. SpikeNET, usually runs very fast simulations but does not allow the simulation of very complex or biologically-realistic neural models, however this project is no longer supported [20].

These popular tools model the biological mechanisms and behaviors in detail, hence are bulky and complicated. It is not feasible to use them to simulate large-scale SNNs in a reasonable time frame. More computationally-efficient simulation tools and neuron models have been developed for large-scale SNN operation simulations. SpiNNaker, which is a low-power and parallel neuromorphic computing platform, can be used to simulate various types of neural networks with different kinds of neurons and connectivity patterns [21].

In order to apply large scale SNN to machine learning applications, simple neuron models should be adopted, which are biologically inspired but not biologically realistic. The model should support efficient learning and recall while at the same time facilitate parallel and distributed implementation, and the simulator should explore the distributed computing and vector processing capability offered by today's multicore architecture for fast learning and evaluation.

III. Neuron models

A. Bayesian neuron model

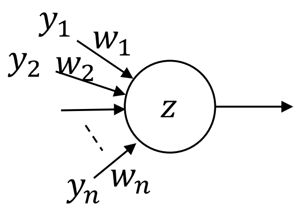


Fig. 1. Generic neuron model

We extend the generic Bayesian neuron model proposed in [14] for scalable and distributed computing purpose. This model supports recall and online learning using STDP. We use this Bayesian neuron model for building inference networks. This section discusses key background details of this model along with online STDP Learning. The details of a generic neuron model are shown in Fig. 1. In the neuron model, the membrane potential $u(t)$ of neuron Z is computed as.

$$u(t) = w_0 + \sum_{i=1}^{n} w_i \cdot y_i(t) \tag{1}$$

where w_i is the weight of the synapse connecting Z to its i^{th} presynaptic neuron y_i, $y_i(t)$ is 1 if y_i issues a spike at time t, and w_0 models the intrinsic excitability of the neuron Z. The stochastic firing model for Z, in which the firing probability depends exponentially on the membrane potential, is expressed as

$$prob(Z\ fires\ at\ time\ t) \propto \exp(u(t)) \tag{2}$$

In Eqn.(1), small variations of $u(t)$ resulting from the synaptic weight changes will have an exponential impact on the firing probability, which is not desirable. To mitigate this effect a *range mapping function* is adopted. This function is a parameterized sigmoid function for representing more flexible S-shaped curves:

$$u'(t) = A + B/(1 + \exp(-(u(t) - C) \cdot D)) \tag{3}$$

The above equation has four parameters for shape tuning. Parameter: A provides Y-axis offset, B performs scaling along Y-axis, C provides X-axis offset and finally D performs scaling along X-axis. It maps a range of $u(t)$ to a different range $u'(t)$ and the Out-of-range $u(t)$ to asymptotic values of the function. This makes sure that the membrane potential always lies within the dynamic range of the neuron. After mapping, $u(t)$ in Eqn.(1), should be replaced by $u'(t)$.

Learning involves updating the weight w_i of ith synapse and the intrinsic excitation w_0 of the neuron. Their changes are calculated as below.

$$\Delta w_i = \begin{cases} ce^{-w_i} - 1, & \textit{if spike occured in STDP window} \\ -1, & \textit{if there is no spike in STDP window} \end{cases} \tag{4}$$

$$\Delta w_0 = e^{-w_0} \cdot z - 1 \tag{5}$$

The above delta changes are scaled with a constant learning rate to perform the final update. It can be proved [14] that based on this learning rule the w_i converges to the log of the probability that the presynaptic neuron y_i fired within the STDP window before neuron Z fires, and the firing probability of Z calculated by Eqn. (2) is the Bayesian probability of Z given the condition of its input neurons $y_1, y_2, \dots y_n$ (i.e $P(Z|y_1, y_2, \dots y_n)$.

To obtain Poisson spiking behavior, the method presented in [22] is adopted. The spike rate $\lambda(t)$ is an exponential function of the inputs, which is represented by Eqn.(4). To generate a Poisson process with time-varying rate $\lambda(t)$, the *Time-Rescaling Theorem* is used. According to this theorem, when spike arrival times v_k follow a Poisson process of instantaneous rate $\lambda(t)$, the time-scaled random variable $\Lambda_k = \int_0^{v_k} \lambda(v)dv$ follows a homogeneous Poisson process with unit rate. Then the inter-arrival time τ_k satisfies exponential distribution with unit rate.

$$\tau_k = \Lambda_k - \Lambda_{k-1} = \int_{v_{k-1}}^{v_k} \lambda(v)dv \tag{6}$$

To find the next spiking time v_k, a random variable is generated satisfying exponential distribution with unit rate, which represents τ_k. The integral in Eqn.(6) cumulates the instantaneous rates from Eqn. (2) over time until the integral value is greater than or equal to τ_k. Once this happens it implies that the inter-spike interval has passed and a spike is generated accordingly. In this way Poisson spiking behavior is generated based on the state of the neuron.

B. Rectified Linear Unit neuron model (ReLU)

From theory behind the Bayesian neuron model it is clear that the neuron is memory less and computation happens based on instantaneous rates. So, when this neuron is used in a network any weighted spike received by this neuron will have small effect on the overall firing rate. Hence the net effect of the weighted spike must be spread over time. This conversion mechanism is achieved by using a spiking Rectified Linear Unit (ReLU) neuron.

The ReLU function is defined as $Z = max(U_{th}, u(t))$ where Z is the number of output spikes, U_{th} is a constant threshold, and $u(t)$ is the membrane potential of this neuron calculated as $u(t) = u(t-1) + \sum_{i=1}^{n} w_i \cdot y_i(t) - U_{th}$. In other words, the membrane potential of a ReLU neuron accumulates every weighted input spike and discharges it over time resembling a burst firing pattern. In our implementation, the spiking threshold U_{th} is set to 1, and after each spike generation, the membrane potential is reduced by the threshold value. This makes sure that accumulated membrane potential is discharged faithfully over time.

C. Winner takes all

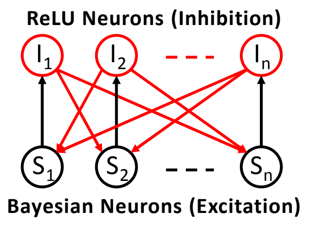


Fig. 2. Winner take all circuit

Fig. 2 shows a neural circuit to laterally inhibit a group of Bayesian neurons in a *winner take all* (*WTA*) manner. WTA circuit is a recurrent network where a set of neurons compete with each other for activation. Hard or soft WTA behavior can be achieved based on the degree of inhibition delivered. *Hard WTA* happens when the inhibition is strong such that it brings down the firing rate of the non-preferred Bayesian neurons to zero, resulting in only one neuron with highest excitation being active. On the other hand, if plural voting action is required within the set, the degree of inhibition is tuned to be moderate. This makes Bayesian neurons fire with different stable rates which is, *soft WTA* behavior where firing rate is proportional to their relative excitation levels.

IV. Parallel implementation of SNN Simulator

The bottleneck in wide spread adoption of SNNs is the lack of simulation tools to handle large-scale networks. Our proposed neuron model has features, such as uniform synapse connection and distributed learning, which facilitate large scale parallel implementation. In this work, we develop SpNSim, a flexible, multithreaded and vectorized spiking neural network simulator using C++. SpNSim has the ability to simultaneously simulate and train heterogeneous neural networks, i.e., networks consisting of different spiking neuron models with different behaviors including activation functions and STDP rules. This is a key feature in implementing complex neural networks with distinct subnetworks.

A. Architecture

The SpNSim is designed to be modular and extendable. Its overall architecture is shown in Fig. 3. There are three main layers in the design. First layer is Network layer where network definitions are read from user-provided XML files and neural networks are created. It also writes the trained networks back to XML files. Internally the network representation is maintained as a 3D graph in a Cartesian coordinate system, with vertices representing neurons and edges representing connections. The second layer is the simulation engine, which takes care of simulating the network in a multi-threaded environment. Finally, the visualizer layer helps in debugging and rendering the complex

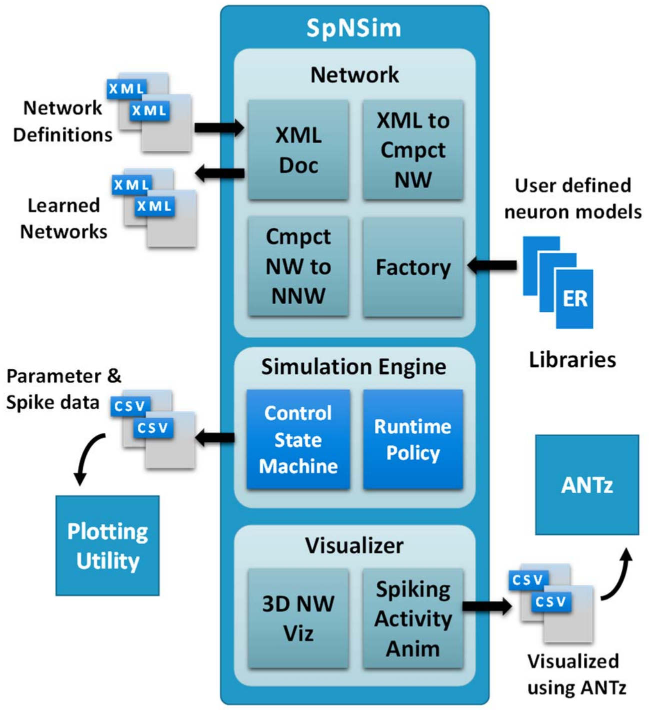


Fig. 3. SpNSim architecture

SNNs in 3D. All the subcomponents of SpNSim are described in detail in the following sections. The simulation treats time in discrete time steps called ticks.

B. Evaluation Routines for Neuron Behavior Simulation

We use interface class to make developing neuron models flexible so that any kind of behavior can be integrated. Neuron models are represented as evaluation routines (ER). ER provides platform for multi-threaded execution and thread synchronization. By default only one thread per ER is assigned, but based on the model requirement it is scalable. Each instance of ER is capable of holding data for any number of neurons of its type. The key advantage of this approach is that all data including weights and neuron parameters for numerous neurons of the same type are stored in arrays. These arrays are dynamically created and memory aligned to the processors vectorization boundary during initialization. The functions for the compute logic is developed such that there are no data dependencies across iterations of loops (i.e. no inter neuron data dependency), which is a prerequisite for enabling vectorization of code. The spike propagation is handled using pointers for quick communication. To avoid data dependency while computing current spike status we use two arrays, one is called current spike and the other is called pending spike. For any given tick the status of the neuron is computed using current spike and the results are stored in pending spike to avoid data corruption as all the neurons are being evaluated in parallel. Later the pending spike will be copied to current spike. This constitutes to spike communication in a hazardless manner. Since all the compute logic and data is available in the ER, it is straightforward to optimize them for use with accelerators like GPGPU or Intel Xeon Phi. Though this kind of acceleration is possible, it not tested for current implementation.

C. Runtime policy

Runtime policy (RP) is a feature defined in the network definition file. This specifies dynamic behavior of the network, such as the starting and ending time of training and recall phases. Also at run time certain neuron parameters can be overridden for debug purposes or for modeling certain biological behaviors where the presence of neurotransmitters modulates the behavior of neurons for example providing reward and penalty behavior to neurons. The run time policies are defined as operations to be performed on the specified set of neurons. These operations are associated with triggers, which can be activated by certain conditions. Triggers can be of different types. Currently we have only implemented time triggers, with room for expansion to other kinds of triggers in the future. Time triggers are defined with activation expression, which always resolves to absolute simulation time. The expression can also be constructed based on other triggers plus relative time. Time triggers can also be defined as sequences or patterns, which resolve to a list of time steps. This rich way of defining triggers allows complex dynamic behavior of the network.

D. Simulation engine

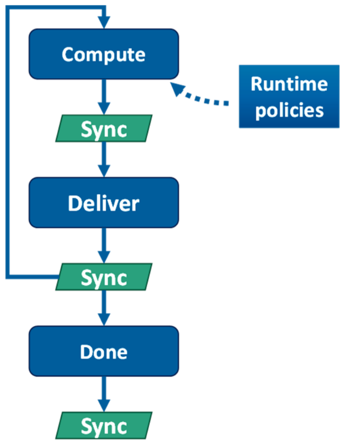


Fig. 4. Simulation engine control flow

This module is core of the simulator. Once the network is created, a list of ER instances is registered with the simulation engine (SE). This process links up every thread from all the ERs with thread-safe blocking queues for two-way communication. Using blocking queue allows a thread to go to sleep while the queue is empty, thus freeing up resources for other operations. SE communicates with ER threads using command and response messages. Simulation engine implements a state machine with three states; Compute, Deliver and Done. Transition to next state is done only after a Sync operation where the responses of all the threads is received. The Sync operation enables the discretization of simulation time and also enables the computation of all neurons asynchronously with in each state.

The basic control flow is shown in Fig. 4. Simulation time is advanced in compute state and compute command is broadcast to all threads. After receiving their responses, the state transitions to deliver state. Here the output spike status generated during this tick is delivered as inputs to downstream neurons, which will be used for computation during next tick. This task primarily boils down to copying data from pending spike array to current spike array. Once the simulation time reaches the user-defined limit, it sends termination commands to all the threads. After receiving all responses, the simulation terminates. This ensures that all threads have safely terminated and released all the resources back to the system.

The runtime policies including their triggers, operations and the set of associated neurons are resolved before the start of simulation. This information is registered with the SE. During the compute state SE checks for any triggered events. If any one of them are activated then the actions associated with the corresponding operations are included in the command messages to ERs, which encapsulates the list of specific neurons affected by that operation.

E. Network specification and creation

In biological nervous system, the neurons form well defined circuits performing specific tasks. To accommodate such complex neural circuits in SpNSim we define templates, which is a subnetwork comprising several neurons of any type with their associated connectivity. Instances of those templates can be placed at any given location in 3D space. Each neuron in that instance is referred hierarchically, using a concatenation of the instance name and its relative 3D location within the template. The intention of using template is to have a library of frequently used sub-networks and also to save trained networks, which can be reused as sub-circuits in more complex designs.

Two types of templates can be defined to realize neurons; group template and column template. Group template has a 2D structure, it defines placement of neurons in the X-Y plane. A column template has a 3D structure which is built by instantiating group templates along X, Y and Z directions. A column template also defines the connectivity among the instantiated group templates, hence creating a template of connected sub network. Since group instances in a column template results in building the column template, no physical neuron is realized until this column template is instantiated in the network. Like column templates the group templated can be instantiated directly in the network to realize physical neurons.

The input of the simulator consists of one network definition file and any number of template definition files. All of them are specified in XML format. The template definitions can be included in the network definition file however, the use of template definition files provide the power of modularity and re-usability by lending support for developing a library of trained/re-usable sub-networks. The network definition file is responsible for instantiating all neurons within templates to build a network. The runtime policy is also specified in the network definition file for controlling the dynamic behavior of the network.

The connectivity among neurons can be specified as explicit connection or as a group. Explicit connectivity specifies connections from multiple source neurons to only one target neuron, whereas group specification makes multiple connections from specified list of source neurons to a list of target neurons. Depending on the requirement different patterns of connectivity can be assigned for example, full connectivity where all sources are connected to all targets or one-is-to-one connectivity where one source neuron only connects to corresponding target neuron in the list. Apart from this, probability values can be associated to the connectivity specification, so that links are established randomly. Weight patterns, including specific and random weight assignments, are defined for these connections as well. For example, a given weight is applied to all the connections or random weights with in the specified range are applied. Other connection parameters can be defined for example, an incoming connection to a neuron can have its learning mode enabled or disabled.

After running the simulation the learned network can be saved back to XML format. The network can be saved as network definition file or template definition file. The network definition file can be loaded back and run at a later time from the previous state. This allows snapshots of simulation to be saved. If the network was saved as template definition file then it can be imported by any other network definition file and be used as trained sub-network.

SpNSim creates the network in a three step process. In the first step we read all the definition files. These XML files are parsed and a XML tree structure is created. In the next step, using this tree a compact network representation is created. The reason behind creating this is to determine the total amount of memory required to build the SNN. The total memory requirement is not directly evident from the XML tree as templates can overlap in certain situations resulting in fewer neurons for such cases. Knowing the exact amount of memory is critical as this memory must be dynamically allocated such that it is memory aligned to the processors vectorization requirement. Finally, the actual spiking neural network is created. During the network creation process, first all the neurons are created then the connections are made, hence avoiding complex network graph traversals. To resolve name conflicts across network and template definition files, all file names in the project are required to be unique. In the internal representation of data elements all items are renamed with respect to their scope including file scope, hence making all names across files unique and a name resolution lookup is maintained to identify the right element.

Neuron types defined in the XML file result in ERs being created. These ERs behave as containers for neurons as described earlier. A neuron factory is used to create neurons in the appropriate ER based on the neuron type. If there are a large number of neurons of the same type, then extra instances of ERs can be created to increase the number of threads to evaluate them without modifying the underlying ER code.

F. 3D Visualizer

Developing and debugging complex SNNs is a non-trivial task, especially when one has to creatively imagine the network in 3D and provide the specification through XML. To facilitate this process we use an open source 3D data visualizer called ANTz [23] to visualize the network and its spiking activities. SpNSim creates CSV files in the required format to represent the inferred SNN from the XML files. These CSV files are loaded on to ANTz for rendering the inferred network in 3D. All neurons are displayed as spheres and directed connections as cones with the base at the source neuron and the tip at the target neuron. Each neuron type is displayed with a different color for better understanding. SpNSim also records details of spiking activities in another CSV file, which can be used by the ANTz to animate spike generation and delivery.

G. Plotting utility

SpNSim outputs spikes and neuron parameter data in CSV file format for the specified neurons. These can be analyzed using Microsoft Excel or using a rich plotting utility we have developed in MATLAB. This utility has the capability of showing raster plots for spikes and perform post processing like window analysis and signal to noise ratio analysis on these plots to determine the statistical behavior among the spikes. The neuron parameter variations can also be plotted for debugging and analysis purpose. The plotting utility is capable of displaying a set of weights over consecutive time steps resulting in an animated view of weight evolution during learning phase. This feature is particularly helpful in understanding the learning behavior of the network with respect to change in parameters.

V. Experiments

To validate its functionality, we applied the neuron model on two different applications. In the first experiment, the neuron model is used to perform unsupervised feature learning and extraction of hand written digits. With this experiment, we demonstrate the in-system learning capability of the neuron model. The second experiment demonstrates the model's capability of performing Bayesian inference, where it is applied for inference based sentence construction.

A. Unsupervised Feature Learning and Extraction

The stochastic firing and STDP learning enables unsupervised feature learning and extraction, which is the function of the base layer in every convolutional network for image recognition. The MNIST dataset is used for this experiment to learn features of handwritten digits ranging from '0' to '9'. We use 2000 randomly selected samples from the training set to learn the features and tested against 2000 images randomly picked from the testing set. For all the experiments we use binary MNIST images.

A convolutional neural network is constructed using the Bayesian neurons. Two different kernel sizes are used for the experiments, 5x5 and 7x7. For both kernels we set a stride of 2 pixels along X and Y directions. Each kernel is mapped to 9 features, implemented by 9 Bayesian neurons in the output layer. Each neuron of the Bayesian output layer is connected to all input neurons in its kernel. The input neurons perform population coding of input pixels, with two neurons representing black and white value of each input pixel. The neurons in the input layer fire, facilitating the Bayesian neurons to fire. Based on their relative spike-timing, the weight of the synapse is updated. A ReLU neuron based inhibition layer is attached to the output layer which realizes hard WTA function to ensure that only one feature will be activated for each kernel so that each Bayesian neuron learns a unique feature. The network setup of a 5x5 kernel size is shown in Fig. 5. The input layer consist of 50 neurons, the output and inhibition layers both have 9 neurons each. When an input neuron is active, it fires at 10% probability. The learning rate is fixed at 0.01, and the STDP period is 30 ticks for the experiments. The duration of STDP window is in the range of 10ms in a biological system.

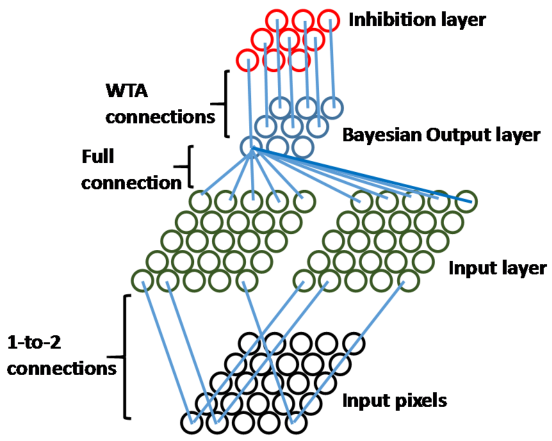


Fig. 5. Network structure (50x9x9)

The stochastic SNN performs learning and feature extraction similar to Convolutional Restricted Boltzmann Machine (CRBM) [24]. The same set of training and testing images is applied to an open source software implementation of CRBM. We found that they give comparable feature maps and filtered images as shown in Fig. 6. The support vector machine (SVM) classifier is used to check the effectiveness of the learnt features. Two different SVMs are trained and tested using features extracted from stochastic SNN and CRBM. The results show that the features extracted by stochastic SNN and CRBM can be used to classify with an accuracy of 94.4% and 94.45% respectively using 5x5 kernel, and 92.6% and 91.4% accuracy using 7x7 kernel, demonstrating the effectiveness of stochastic Bayesian neuron model with the results as shown in TABLE I. This accuracy is lower than the state-of-the-art results, which is 94% for SNN [4] and 99.18% for CRBM [24] as we are using the black-and-white images instead of greyscale images. We also observed that losing 50% of the connection will not cause notable performance degradation for large kernel. However, accuracy loss starts at 50% connections for small kernel. Finally, we expedite training by reducing the time that the training image is exposed to the system, and observe marginal impact.

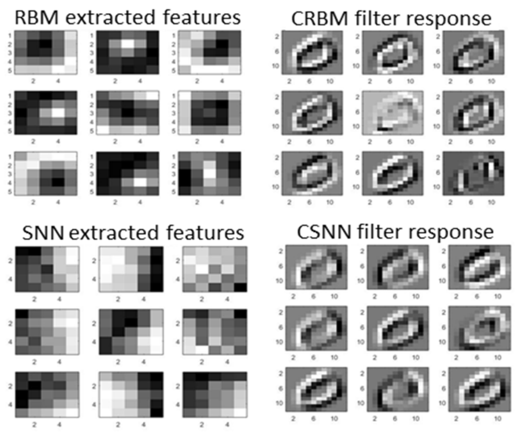


Fig. 6. Nine 5x5 extracted features and corresponding filter responses from our SNN and CRBM

TABLE I. Classification results

5x5 kernel (NW Size: 50x9x9)			**7x7 kernel (NW Size: 98x9x9)**		
Learning Time (ticks)			**Learning Time (ticks)**		
100	**300**	**500**	**100**	**300**	**500**
93.25	94.05	94.4	91.2	92.6	92.5
Connectivity %			**Connectivity %**		
50	**70**	**100**	**50**	**70**	**100**
91.7	92.35	93.25	91.2	90.25	91.2

The 3D visualization of the network learning 9 features from a 5x5 kernel is shown in Fig. 7. The bottom layer of green neurons are input pixel neurons, they fire only if a bright pixel of the image is exposed to the neuron. The next two layers of blue neurons correspond to input layers. One layer consists of neurons preferring black pixels and the other white pixels. The next layer with turquoise neurons are the Bayesian neurons and finally the orange neurons provide inhibition.

B. Confabulation theory based inference

An inference network for sentence construction is created using Bayesian neurons. It consists of lexicons representing words and phrases. As shown in Fig. 2, a lexicon is a WTA sub-

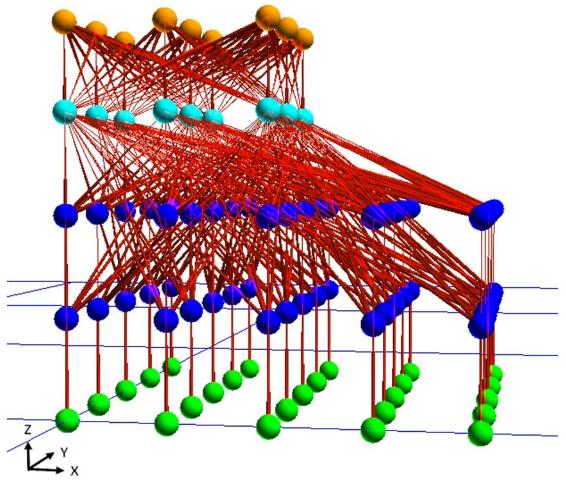
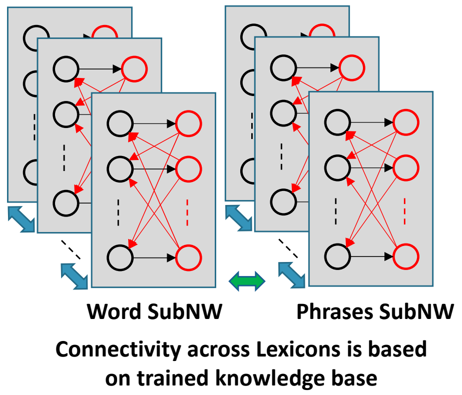


Fig. 7. 3D NW visualization of 9 feature learning of 5x5 kernel

network with Bayesian neurons for excitation and ReLU neurons for inhibition. Each Bayesian neuron represents a symbol, which in this case is a potential word or phrase at certain location of sentence. The synapses between neurons across lexicons are created based on the log conditional probability of the two connected words (phrases). All neurons are initialized with the same intrinsic potential (i.e. the same initial firing rate). The most strongly connected neurons resonate and enhance each other and at the same time inhibit the other neurons in the same lexicon. The network settles on contextually correct behavior and neurons with the highest firing rate in each lexicon marking the sentence that is grammatically correct and semantically meaningful. In this application, the inhibition layer performs soft WTA. It has an advantage over hard WTA because symbols with lower excitation are not discarded, thus more information is retained during inference. Fig. 8 shows the network topology.



Word SubNW
Phrases SubNW
Connectivity across Lexicons is based on trained knowledge base

Fig. 8. Sentence confabulation network

We randomly picked 45 sentences from document images. Fuzzy character recognition is performed on these images which results in multiple possible words for each word position as described in [25]. For example, given input candidates [{we, wo, fe, fo, ne, no, ns, us} {must, musk, oust, onst, ahab, bust, chat} {now, noa, non, new, how, hew, hen, heu} {find, rind, tina} {the, fac, fro, kho} {other, ether}], the SNN settles at a grammatically correct sentence as [we must now find the other]. The raster plot for this example is shown in Fig. 9. The labels along Y-axis are grouped by lexicon in the following format; the lexicon number, symbol represented by the neuron and the 3D (x y

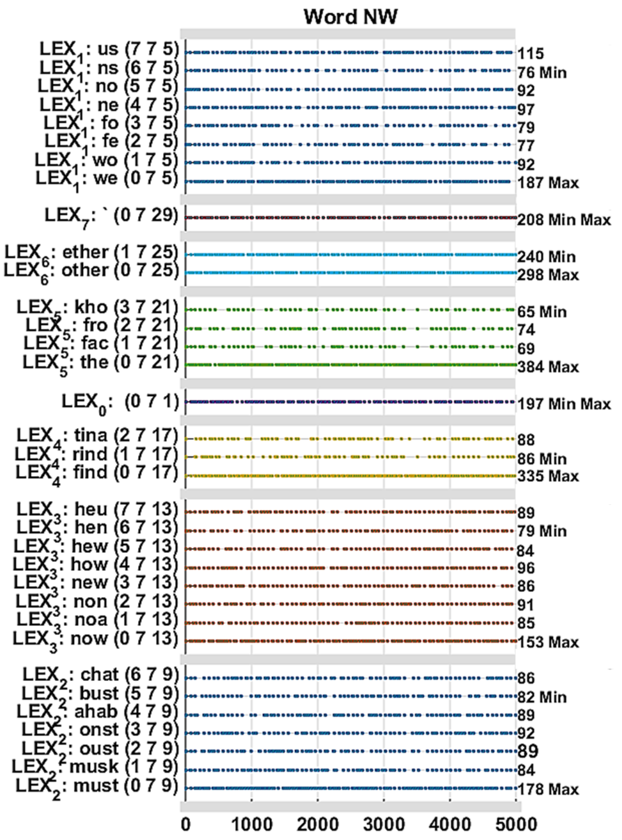


Fig. 9. Confabulation results raster plot

z) co-ordinate of the neuron in the network. The spike count for winning symbols is highest in the Lexicon, which is shown along the secondary Y-axis. The X-axis represents the simulation ticks. Lexicon-0 and lexicon-7 in the figure mark the beginning and end of the sentence. The average SNR across all lexicons is 2.57. Overall, the stochastic SNNs are able to construct correct sentences for 83.8% of the test cases.

VI. Conclusion

In this paper we have developed and presented the architecture for a flexible, scalable and high performance simulation platform for spiking neural networks. The simulator has the ability to model non-biologically realistic neuron models which enable efficient computation. We also demonstrate the functionality of the simulator for learning through STDP and evaluation of inference networks. These network results were validated based on other existing platforms.

References

[1] Gupta, A.; Long, L.N., "Character Recognition using Spiking Neural Networks," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, vol., no., pp.53-58, 12-17 Aug. 2007

[2] Behi, T.; Arous, N.; Ellouze, N., "Self-organization map of spiking neurons evaluation in phoneme classification," in *Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), 2012 6th International Conference on*, vol., no., pp.701-705, 21-24 March 2012

[3] Wang, Y.; Tang, T.; Xia, L.; Li, B.; Gu, P.; Yang, H.; Li, H.; Xie, Y.; "Energy Efficient RRAM Spiking Neural Network for Real Time Classification." In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 189-194. ACM, 2015.

[4] O'Connor, P.; Neil, D.; Liu, S.C.; Delbruck, T.; Pfeiffer, M.; "Real-time classification and sensor fusion with a spiking deep belief network."*Frontiers in neuroscience* 7 (2013).

[5] Beyeler, M.; Dutt, N.D.; Krichmar, J.L.; "Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule." *Neural Networks* 48 (2013): 109-124.

[6] Maass, W., "Networks of spiking neurons: the third generation of neural network models." *Neural networks* 10, no. 9 (1997): 1659-1671.

[7] Sjöström, J.; Gerstner, W., "Spike-timing dependent plasticity." *Spike-timing dependent plasticity* (2010): 35.

[8] M. W. Oram, M. C. Wiener, R. Lestienne, and B. J. Richmond, “Stochastic nature of Precisely Time Spike Patterns in Visual System Neuronal Responses,” *Journal of Neurophysiology*, vol. 81, pp. 3021—3033, 1999.

[9] H. S. Seung, “Learning in spiking Neural networks by Reinforcement of Stochastic synaptic Transmission,” *Neuron,* Vol. 40, pp. 1063-1073, Dec., 2003.

[10] T. A. Engel, W. Chaisangmongkon, D. J. Freedman, and X. J. Wang, “Choice-correlated Activity Fluctua-tions Underlie Learning of Neuronal Category Representation,” *Nature Communications* 6, Mar., 2015.

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

[12] Rao, R.P.N.; Olshausen, B.A.; Lewicki, M.S., Probabilistic models of the brain: Perception and neural function. MIT press, 2002.

[13] Deneve, Sophie. "Bayesian inference in spiking neurons." *Advances in neural information processing systems* 17 (2005): 353-360.

[14] Nessler, B.; Pfeiffer, M.; Buesing, L.; Maass, W., "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity." (2013): e1003037.

[15] Hines, M.; Carnevale, N., "The NEURON Simulation Environment," in *Neural Computation*, vol.9, no.6, pp.1179-1209, Aug. 15 1997

[16] Bower, J.M.; Beeman, D. The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System. Springer Science & Business Media, 2012.

[17] Goodman, Dan, and Romain Brette. "Brian: a simulator for spiking neural networks in Python." *Frontiers in neuroinformatics* 2 (2008).

[18] Gewaltig, Marc-Oliver, and Markus Diesmann. "NEST (neural simulation tool)." *Scholarpedia* 2, no. 4 (2007): 1430.

[19] Ros, Eduardo, Richard Carrillo, Eva M. Ortigosa, Boris Barbour, and Rodrigo Agís. "Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics." *Neural computation* 18, no. 12 (2006): 2959-2993.

[20] http://sccn.ucsd.edu/~arno/spikenet/

[21] Furber, S.B.; Lester, D.R.; Plana, L.A.; Garside, J.D.; Painkras, E.; Temple, S.; Brown, A.D., "Overview of the SpiNNaker System Architecture," in *Computers, IEEE Transactions on*, vol.62, no.12, pp.2454-2467, Dec. 2013

[22] http://white.stanford.edu/pdcwiki/index.php/Spike_generation_using_a_Poisson_process

[23] http://openantz.com

[24] Lee, H.; Grosse,R.; Ranganath, R.; Ng, A.Y.; "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 609-616. ACM, 2009.

[25] Qinru Qiu; Qing Wu; Bishop, M.; Pino, R.E.; Linderman, R.W., "A Parallel Neuromorphic Text Recognition System and Its Implementation on a Heterogeneous High-Performance Computing Cluster," in *Computers, IEEE Transactions on*, vol.62, no.5, pp.886-899, May 2013