Probabilistic Inference Using Stochastic Spiking Neural Networks on A Neurosynaptic Processor

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

Qing Wu
Air Force Research Laboratory, Information Directorate
525 Brooks Road, Rome, NY, 13441, USA
Email qing.wu.2@us.af.mil

***Abstract*— Spiking neural networks are rapidly gaining popularity for their ability to perform efficient computation akin to the way a brain processes information. It has the potential to achieve low cost and high energy efficiency due to the distributed nature of neural computation and the use of low energy spikes for information exchange. A stochastic spiking neural network naturally can be used to realize Bayesian inference. IBM's TrueNorth is a neurosynaptic processor that has more than 1 million digital spiking neurons and 268 million digital synapses with less than 200 mW peak power. In this paper we propose the first work that converts an inference network to a spiking neural network that runs on the TrueNorth processor. Using inference-based sentence construction as a case study, we discuss algorithms that transform an inference network to a spiking neural network, and a spiking neural network to TrueNorth corelet designs. In our experiments, the TrueNorth spiking neural network constructed sentences have a matching accuracy of 88% while consuming an average power of 0.205 mW.**

I. Introduction

The brain is capable of performing a wide variety of tasks in an efficient and distributed manner, while consuming very low power [1]. This is due to the large number of simple computing elements i.e. neurons, and the rich connectivity among them. Neurons communicate using characteristic electrical pulses called action potentials. Inspired by the biological nerve system, Spiking Neural Networks (SNNs), which utilize spikes as the basis for operations, is the third generation of neural networks. The SNN has the potential to achieve very low energy dissipation since each neuron works asynchronously in an event-driven manner. Moreover, fully distributed Spike Timing Dependent Plasticity (STDP) learning [2] can be achieved on SNNs, which relies only on local information of individual neurons. The emerging stochastic SNN that generates spikes as a stochastic process not only is more biologically plausible [3] but also enhances unsupervised learning and decision making [4] [5]. It further increases the fault tolerance and noise (delay) resiliency of the SNN system because the results are no longer dependent on the information carried by individual spikes but the statistics of a group of spikes.

Bayesian inference and belief networks are powerful tools for many applications, such as error correction, speech recognition, and image recognition. Recently deep belief networks have demonstrated amazing results in unsupervised feature extraction [6] and image recognition [7]. A stochastic SNN naturally implements Bayesian learning and belief propagation. In [8], the authors presents a Bayesian neuron model and the STDP learning rule. It can be proven that based on given STDP learning rules the synaptic weight of a neuron converges to the log of the probability that the presynaptic neuron fired within the STDP window before post synaptic neuron fires, and the firing probability of the post synaptic neuron is its Bayesian probability given the condition of its input neurons.

Despite the simplicity of the SNN, it is not efficient when implemented on traditional processors with the Von Neumann architecture, due to the performance gap between memory and processor. The IBM Neurosynaptic System provides a highly flexible, scalable and low-power digital platform [9] that supports large scale SNN implementation. IBM's neurosynaptic processor called TrueNorth has 4096 cores and each core features 256 neurons and axons. The synaptic connections and their weights between axons and neurons are captured by a crossbar matrix at an abstract level. This abstraction is in the form of the programming paradigm for TrueNorth called Corelet [10]. Corelets represent a network on the TrueNorth cores by encapsulating all details except external inputs and outputs. The creating, composing and decomposing of corelets is done in an object-oriented Corelet Language in Matlab.

While the TrueNorth chip is a flexible platform, it does pose several constraints. To maintain extremely low cost and high energy efficiency, each column in the crossbar only supports 4 different synaptic weights [11] and all the synaptic weights are associated to axon types which are shared by all other neurons of the core. Hence all neurons using a row are required to use the same weight rank. Also because of the 256x256 crossbar, the fan-in and fan-out per neuron is limited to only 256. These constraints limit the direct mapping from a given SNN to its TrueNorth implementation. To the best of our knowledge, there has not been any public domain tool that converts an arbitrary SNN to the TrueNorth implementation. Though, several applications have been developed on TrueNorth by following design approaches, "train-then-constrain" [12] [13] or "constrain-then-train" [11], which include the methods of constructing and training the network on libraries such as Pylearn2/Theano or Caffe and mapping them onto TrueNorth as per their network.

In this work, we aim at implementing a trained probabilistic inference network on TrueNorth. It involves two steps: at first the inference network is transformed into a stochastic SNN; and secondly the stochastic SNN is converted into a TrueNorth implementation. The main contributions of this work are summarized as follows.

1. This work introduces a general architecture of a stochastic SNN that has close correspondence to the probabilistic inference model. The network features excitatory and inhibitory links. The belief propagation is carried out by Bayesian neurons and their excitatory links. The normalization among neurons in the same category is realized by special neural circuitry that performs soft winner-take-all (WTA) functions.
2. We have developed a set of algorithms that automatically convert the stochastic SNN into the core and crossbar configurations for efficient TrueNorth implementation.
3. The effectiveness of the proposed method is demonstrated by ap-

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

plying the TrueNorth implemented network to the sentence construction problem, which searches for the correct sentence from a set of given words. The results show that the system always chooses the words that form grammatically correct and meaningful sentences with only 0.205 mW average power consumption.

In the following sections we will discuss the related work, then introduce the proposed neuron models and discuss in detail about the WTA networks. We elaborate on the design environment for creating and implementing the stochastic SNN, which is followed by the details on creation of an inference-based sentence construction SNN along with the programming of a spiking neural network processor, TrueNorth. Finally, the experimental results are presented and discussed.

II. Related Works

The majority of neuron models used in existing SNNs are not stochastic. [14] presents a neuron model which uses active dendrite and dynamic synapse approach with an integrate and fire neuron for character recognition. [15] implements spiking self-organizing maps using leaky integrate and fire neurons for phoneme classification. They use this model to account for temporal information in the spike stream. [16] implements a large-scale model of a hierarchical SNN that integrates a low-level memory encoding mechanism with a higher-level decision process to perform a visual classification task in real-time. They model Izhikevich neurons with conductance-based synapses and use STDP for memory encoding. However, the stochastic nature in spike patterns has already been found in lateral geniculate nucleus (LGN) and primary visual cortex (V1) [3]. Ignoring the randomness in a neuron model not only limits its effectiveness in sampling and probabilistic inference related applications [17] [18], but also reduces its resilience and robustness. In this paper we propose modifications to a Bayesian spiking neuron model presented in [8] to make it scalable and efficient for implementing SNNs with distributed computation. We have developed a highly scalable and distributed SNN simulator, SpNSim [19], which we utilize to simulate networks consisting of, among other neuron models, the proposed Bayesian neuron model.

A very low-power dedicated hardware implementation of a SNN is an attractive option for a large variety of applications, in order to avoid the high power consumption when running state-of-the-art neural networks in server clusters. This has led to the development of a number of neuromorphic hardware systems. Neurogrid, developed at Stanford University, is used for biological real-time simulations [20]. It uses analog circuits to emulate the ion channel activity and uses digital logic for spike communication. BrainScaleS is another hardware implementation which utilizes analog neuron models to emulate biological behavior [21]. These implementations have focus limited to biologically realistic neuron models and are not optimized for large-scale computing. On the other hand, IBM's TrueNorth processor is very low-power, highly scalable, and optimized for large-scale computing [11]. However, harnessing the strengths of TrueNorth demands algorithms which are adept to its constraints. Recent developments suggests an emergence of neuromorphic adaptations of machine learning algorithms. It has been shown that a "train-and-constrain" approach can be taken to map a Recurrent Neural Network (RNN) based natural language processing task (question classification) to a TrueNorth chip [12] by matching artificial neuron's responses with those of spiking neurons with promising results (74% question classification accuracy, less than 0.025% of cores used and an estimated power consumption of ≈17μW). The same "train-and-constrain" approach is used to map a Deep Neural Network (DNN) on to a TrueNorth chip [13] for a sentiment analysis task. Here, the mapping is possible through substitution of the ReLU neurons in the DNN with integrate-and-fire neurons and adjusting their neuron thresholds and discretizing the weights using a quantization strategy. Few recognition tasks have also been implemented in other promising neuromorphic hardwares [22] [23]. In this work we also take a "train-and-constrain" approach to implement inference-based Bayesian spiking neural networks on the TrueNorth chip.

III. Probabilistic Inference Using Stochastic SNN

Various experiments have shown the evidence of brain applying Bayesian inference principles for reasoning, analyzing sensory inputs and producing motor signals [24] [25]. Bayesian Inference is a statistical model which estimates the posterior probability with the knowledge of priors. It can produce robust inference even with the presence of noise. This section presents the first step of the design flow, which converts a probabilistic inference network to a stochastic SNN.

A. Confabulation model

We adopt a cogent confabulation model as our input probabilistic inference network [26]. Cogent confabulation is a connection-based cognitive computing model with information processing flow imitating the function of the neocortex system. It captures correlations between features at the symbolic level and stores this information as a knowledge base (KB). The model divides a collection of symbols into categories known as *lexicons*. A lexicon may represent a feature, e.g. color, or any abstract level concept. The symbols represent the elements of the feature, e.g. blue, green etc. are symbols of color lexicon. The symbols within a lexicon inhibit each other and at the same time excite symbols of different lexicons. The connection between symbols of different lexicons is a knowledge link (KL). This link represents the log conditional probability that source symbol (*s*) and target symbol (*t*) are co-excited. It is defined as $\ln[P(s|t)/p_0]$, where $P(s|t)$ is the probability that *s* fires given the condition that *t* fires, and p_0 is a small constant to make the result positive. This definition agrees with the Hebbian theory, which specifies that the synaptic strength increases when two neurons are constantly firing together.

The confabulation model works in three steps. First the excitation level of a symbol is calculated as $el(t) = \sum I(s) \ln(p(s|t)/p_0)$, where *s* are symbols in other lexicons that have excitatory links to t, and *I*(*s*) is the *belief* of *s*. We refer to this as *belief propagation*. Secondly, in each iteration, the weakest symbol in a lexicon is suppressed and deactivated. We refer to this step as *suppression*. A winning symbol (*t*) of a lexicon is computing the log likelihood of the status of other symbols given status of *t* itself, i.e. $e(t) \propto \ln\left[P\left(s_1 s_2 \dots s_{|E_t|} | t\right)\right]$. Thus the confabulation model resolves ambiguity using maximum likelihood inference. Finally, the belief of each active symbol is calculated as $I(s) = el(s)/\sum_{i \in same\ lexicon} el(i)$, so that the total belief of symbols in a lexicon always add up to 1. We refer to this step as *normalization*.

To implement such model using a stochastic SNN, we propose to map symbols to a set of Bayesian neurons and perform belief propagation through their excitatory connections; a winner-take-all circuit is introduced to implement the suppression function; and two special neurons, an *Upper Limiter* (*UL*) and a *Lower Limiter* (*LL*), are used to approximate the normalization function. Details of the design will be introduced in the next sections.

B. Bayesian neuron model

Neurons with stochastic firing are used to implement the symbols in the confabulation model. We extend the generic Bayesian neuron model proposed in [8] to enhance scalable and distributed computing. This section discusses key background details and our extensions of this model. Although the original model supports STDP learning, in

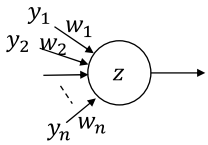


Fig. 1. Generic neuron model

this work we consider only trained network and STDP Learning will not be touched. 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.

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

where w_i is the weight of the synapse connecting Z to its i^{th} pre-synaptic neuron y_i, $y_i(t)$ is 1 if y_i issues a spike at time t and 0 otherwise, and w_0 models the intrinsic excitability of the neuron Z. In our application, w_i is set to $\ln[P(s|t)/p_0]$ that is trained in the confabulation model. The stochastic firing model for Z, in which the firing probability depends exponentially on the membrane potential is expressed as

$$prob(Z\ fires\ at\ time\ t) \propto \exp(u(t)) \quad (2)$$

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:

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

The above equation has four parameters for shape tuning. Parameters *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 $u'(t)$.

To obtain Poisson spiking behavior, the method presented in [27] 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.

$$\tau_k = \Lambda_k - \Lambda_{k-1} = \int_{v_{k-1}}^{v_k} \lambda(v)dv \quad (4)$$

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. (4) 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. Because the Bayesian neurons are used to implement symbols in the confabulation model, we also refer to them as *symbol neurons*.

C. Winner-take-all circuit

We introduce a *winner-take-all* circuit within each lexicon, where neurons inhibit each other so that the more active neurons suppress the activity of weaker neurons. Before presenting the structure of the WTA circuit, we need to introduce a class of non-stochastic neurons, which is used to generate inhibition spikes. The Bayesian neurons are memoryless and their firing rate depends on instantaneous membrane potential, which is limited to a relatively narrow range in order to maintain the stability of the system. Any input outside the range will be truncated. This affects the inhibition process significantly as all the neurons inhibit each other, and the accumulated amplitude of the inhibition spikes will have a large range. Our solution is to spread the large inhibition signal over time. Instead of using amplitude to indicate the strength of inhibition, we spread the amplitude of inhibition over time and use the duration of spikes to represent the strength of inhibition. 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.

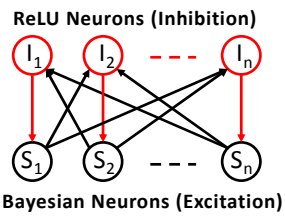


Fig. 2. Winner-take-all NW

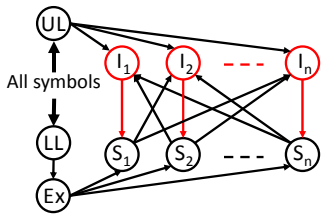


Fig. 3. Normalized winner-take-all NW

Fig. 2 shows a neural circuit to laterally inhibit a group of Bayesian neurons in a winner-take-all manner. WTA network is recurrent where a set of symbol neurons compete with each other for activation. Every symbol neuron has a corresponding ReLU neuron also referred to as *inhibition neuron*. The function of the inhibition neuron is to collect and accumulate spiking activities from neighboring symbol neurons and convert them into inhibition spikes over the time domain.

Hard or soft WTA behavior can be achieved based on the weight of inhibition links. *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 weight of inhibition links is tuned to be moderate. This makes Bayesian neurons fire with different stable rates which is, *soft WTA* behavior. The soft WTA is key to building complex networks as the relative excitation levels can be further used by other network modules for robust inference.

D. Normalized winner-take-all

The original inference model requires that the belief value of all symbols in each lexicon must add up to 1. In a stochastic SNN, this means the total firing activities of neurons in each lexicon must be approximately the same. To achieve this, we introduce *normalized winner-take-all (NWTA)* network. Three neurons, *upper limiter* (*UL*), *lower limiter* (*LL*), and *exciter* (*Ex*), are added to the previously discussed WTA circuit as shown in Fig. 3. Both the UL and LL are regular integrate and fire neurons, which have input links from all symbol neurons (these links are omitted in the figure for the sake of simplicity). The links to UL have positive weights while the links to LL have negative weights. On the other hand, the UL has negative leakage and LL has positive leakage. The leak values are adjusted in proportion to the number of symbols in the NWTA network. The threshold of these neurons is adjusted for a desired range of firing rates for the network.

With this configuration the UL neuron builds up membrane potential every time it receives spikes from symbols which leak away at a constant rate. If the symbols are firing at a higher rate than the rate of leak, then the UL neuron fires indicating the symbols need to be inhibited. The UL neuron drives all the inhibitors with equal weights, hence suppressing all the symbol neurons equally without disturbing their relative excitation levels. On the other hand, the LL neuron builds up the membrane potential due to leak. The membrane potential drops only if the symbol neurons fire. If this firing rate is lower than required, then LL neuron fires indicating that the symbol neurons are firing at a lower rate than desired. One Ex neuron, which is driven by the LL neuron, provides equal excitation for all the symbol neurons. Similar to the inhibitor neurons the exciter neuron is also of type ReLU and it spreads amplitude of excitation over time, again without disturbing the relative excitation levels of the symbol neurons. Hence this recurrent behavior obtained from the controlled feedback through inhibition and excitation forces the symbol neurons to cumulatively fire in the desired rate-range.

E. Overall network creation

To build the overall stochastic SNN, each lexicon in the original inference model is implemented using the above mentioned NWTA network. Excitatory links that connect symbol neurons are established across different lexicons. This network will be referred to as *reference network* in the rest of the paper. Another network will be derived from it and be mapped to the TrueNorth processor, which will be discussed in the next section.

As a case study, we built a small scale SNN based on the trained weights extracted from the intelligent text recognition system (ITRS) [28]. ITRS is trained on a huge corpus of English text. Its knowledge base consists of conditional probabilities between neighboring words and phrases. The number of unique symbols in the knowledge base is about 974,000, which includes words and phrases. Based on this knowledge, it forms anticipations of the word at sentence level context. Given an observation consisting of a fuzzy list of likely word candidates at each word position, familiar information with high relevancy will be recalled resulting in a meaningful and semantically correct sentence. This model has an overall sentence accuracy of 94%.

We extract details only pertaining to few example sentence images to build simple inference networks for evaluating the feasibility of its implementation in stochastic SNN. Each symbol is a potential word or phrase at every word or phrase position of the sentence. All symbol 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 across lexicons and at the same time inhibit other neurons in the same lexicon. When network activity settles, neurons with the highest firing rate in each lexicon marks the sentence that is grammatically correct and semantically meaningful.

We randomly picked a set of 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 [28]. For example, given input candidates [{he, ho, be, bo, re, ro, ne, no} {soon} {found, round} {the, kho, une, unc} {place, placed}], the SNN settles at a grammatically correct sentence as [he soon found the place]. Fig. 4 shows the raster plot for one of the sentences. 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 z) coordinate of the neuron in the reference 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. More results will be provided in Section VI.

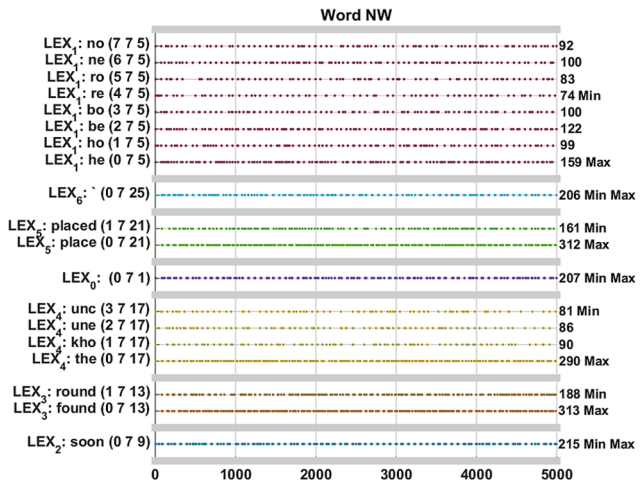


Fig. 4. Reference network results

IV. TRUENORTH IMPLEMENTATIONS

A. Background of TrueNorth processor

TrueNorth is a neurosynaptic processor created by IBM. It is based on the brain's parallel processing architecture and is highly efficient, scalable and flexible. It implements general purpose programmable spiking neurons. The digital architecture of a TrueNorth chip consists 4096 cores [9] each with 256 neurons and 256 axons connected via 256x256 directed synaptic connections, thus providing 1 million programmable neurons and 268 million configurable synapses. TrueNorth uses an efficient event-driven architecture. Address event representation (AER) is adopted for spike representation and communication between neurons. These spike events are sparse in time and active power is proportional to firing activity thus making it highly efficient. Each neuron is independently configurable with a wide variety of neuron models including stochastic ones. Corelets are used as design language for creating networks. Using the Corelet Programming Environment (CPE) these corelets can be programmed to the chip and evaluated or can be simulated using their 1:1 hardware simulator called Compass [29]. The TrueNorth chip is a low power platform. It uses low leakage transistors for minimizing passive power consumption. Active power is minimized due to the event-driven architecture of the logic where computation is performed only when required [30].

B. Design flow

Our second step is to transform the reference network to the TrueNorth implementation. This involves 3 more steps as shown in Fig. 5. Based on the reference network, whose construction is described in the previous section, corresponding shadow networks are created, which comply with the physical constraints posed by the TrueNorth hardware. The shadow network is further flattened to corelets where corelet level details are added. The flattening process results in one corelet per lexicon. These corelets are now connected with each other to build the physical network. The connected corelets are finally simulated using the compass simulator and the TrueNorth chip is programmed for evaluation in real-time.

C. Shadow Network Creation

For every reference network we generate an equivalent TrueNorth compatible shadow network. This network complies with the restrictions imposed by the platform. Due to the hardware restrictions of TrueNorth, some simplifications of the Bayesian neuron must be adopted. For TrueNorth implementation, we replace the Bayesian

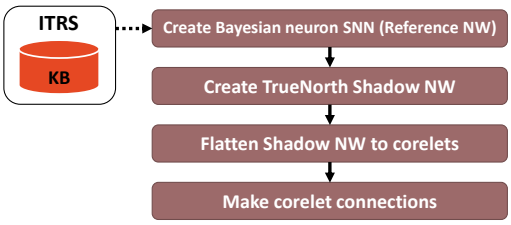
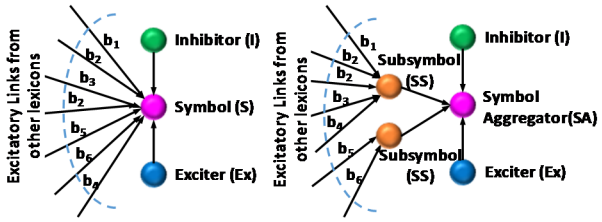


Fig. 5. Design flow

neuron with a stochastic integrate and fire neuron to obtain similar characteristics. From the model described above we infer that there are two computational stages which are unique to the Bayesian model in contrast to the regular integrate and fire neuron model. The first being the exponent function and the other being the Poisson firing. We suggest skipping the exponent computation and using the accumulation of weighted spikes to directly compute the membrane potential. The Bayesian neuron must operate over a small region of the exponential to maintain its dynamic range. This can be approximated with a linear function, which is inherent to the accumulation of membrane potential in an integrate and fire neuron. For Poisson spike generation we suggest randomly varying the threshold after every spike generation. The Bayesian neuron's output firing rate is exponentially proportional to the cumulative rate of input weighted spikes. By limiting the range of threshold change to a small interval which satisfies the exponential distribution with unit rate, we achieve a firing pattern similar to Poisson spiking behavior as described in the model. The general behavior of neuron is still similar to the Bayesian neuron model even with these simplifications. The TrueNorth neuron is configured with this behavior for symbol neurons to obtain a stochastic neuron. The rest of the neurons used in the network can be directly configured to TrueNorth neurons.

Another hardware limitation of TrueNorth is that, although a neuron may receive connections from many axons, the weights of these synaptic links can have only a combination of four different values. Consider a symbol neuron in Fig. 6 (a), it has connections from the inhibitor, the exciter and numerous excitatory links from symbol neurons in other lexicons. All these links have different weights. Hence it cannot be directly implemented as a TrueNorth neuron. Our solution is to decompose a symbol neuron into multiple *sub-symbol* (*SS*) neurons and a *symbol aggregator* (*SA*). For each symbol neuron in the reference network, first, we scale the weight of its incoming links to an integer range. After scaling, these weights are binned for quantization, based on user specified bin width. Different binning strategies can be used. All links falling into the same bin will be assigned the same weight, which is the rounded average of their original weight. For every 4 bins, a sub-symbol neuron is created as the receptor of all links falling into these bins. The sub-symbol neurons are of type ReLU and connect to a symbol aggregator neuron via link with unit weight. Hence, their function is to collect and accumulate input spikes and relay the results to the aggregator over time. The symbol aggregator is a stochastic integrate and fire neuron as previously discussed. It aggregates inputs and generates spikes that will be sent to inhibitors, UL, LL neurons, as well as sub-symbol neurons in other lexicons.

An example of SA and SS neurons is given in Fig. 6 (b). In the figure, the symbol in the reference network has 7 incoming connections falling into 6 bins. In the shadow network, 2 SS neurons are created. The first one receives 5 incoming connections distributed over 4 different bins, while the second one receives the rest of the incoming links originally connecting to the symbol neuron. Both SS neurons connect to an SA neuron, which also receives input from the inhibitor and exciter. Since all SS neurons connect to the SA neuron with unit

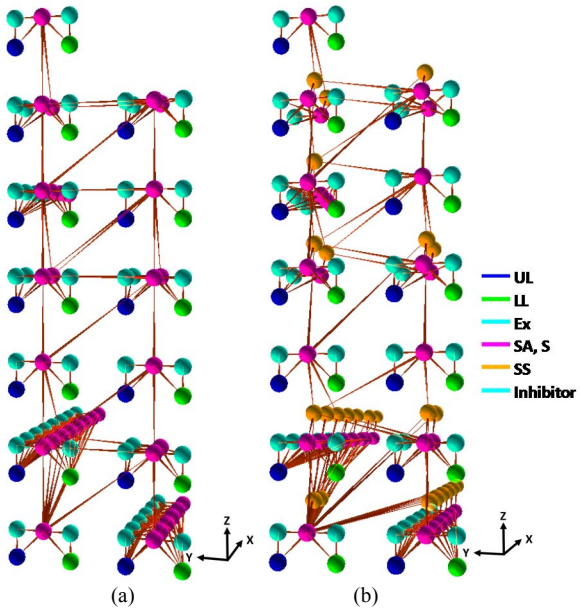


(a) Reference network (b) Shadow network
Fig. 6. Comparing reference network and shadow network

weight, there is no limitation of the number of SS neurons that can be created, as long as there are enough hardware neuron resources. The above procedure is not necessary for neurons other than the symbol neuron, i.e. inhibitor, Ex, LL and UL neurons. They can directly be implemented as a TrueNorth neuron, because their incoming links have the same weight, in other words, they all fall into the same bin. The visualizations for the case of reference network and its equivalent TrueNorth compatible shadow network is shown in Fig. 7. Different type of neurons are shown in different colors as illustrated in the legend. Compared to the reference network in Fig. 7. (a), the shadow network in Fig. 7. (b) has the added SS neurons (in orange) and their links. These network are for the same sentence as shown in Fig. 4. The left and right sub-network represent the word and phrase lexicons respectively. The symbols in each lexicon are shown along X-axis.

D. Flattening Shadow Network

We have developed a parameterized lexicon corelet in CPE. The parameters are the number of symbols, their associated weights, input connectivity and output connectivity. The corelet has one input connector that receives incoming signals from other corelets and two output connectors, one of them is the internal connector that sends the output to the input connector of other corelets; and the other is an external connector that sends the output to the I/O pins. In the TrueNorth architecture, a neuron can only drive a single axon which has a fan out of 256 within a single core. In cases where a neuron has to send spikes to downstream neurons in multiple cores then the upstream neuron must undergo splitting. To achieve this the splitting neuron is configured with weight equal to 1 and threshold equal to 1.



(a) (b)
Fig. 7. SpNSim 3D NW visualization a) Reference NW with Bayesian neurons b) TrueNorth equivalent shadow NW

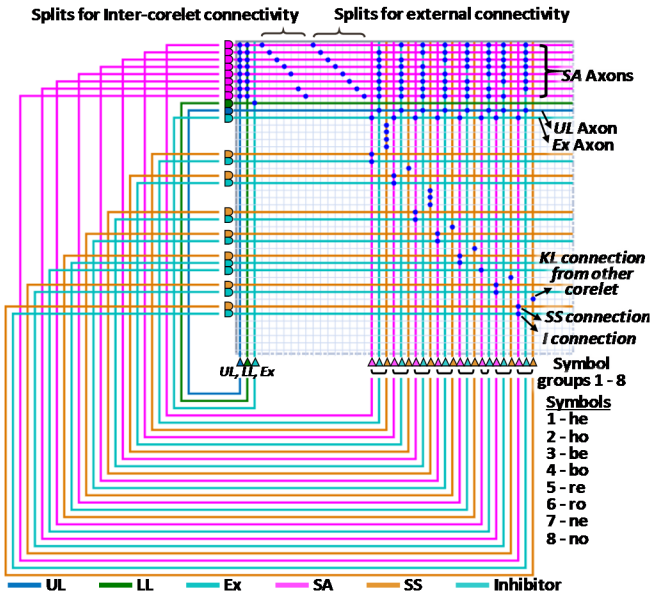


Fig. 8. Crossbar connections for lexicon 1

Fig. 8 shows a flattened network for lexicon 1 of the example sentence given in Fig. 4, which is the crossbar of a core. The rows of the crossbar are the axons and the columns are dendrites. The neurons are placed along the bottom of the crossbar. A dot in the crossbar represents a synapse. The figure shows wiring for only loopback connections. The dendrites, axons, loopback connections and the neurons are depicted based on the color legend given in the figure. From the crossbar we can see that UL (i.e. dark blue) and LL (i.e. green) neurons have synapses from all magenta axons looped back from SA neurons. Those magenta colored axons also generate splits for inter-corelet and external connectivity, and connect to all inhibitor (i.e. cyan) neurons. Most SA (i.e. magenta colored) neurons have 3 synapses, coming from orange (i.e. SS), blue (i.e. Ex) and cyan (i.e. inhibitor) axons. All SS (i.e. orange colored) neurons have synapses from axons that are not colored, which means links from other corelets (i.e. lexicons). With these connections, we flatten the NWTA network. The pseudo code to setup cores and neurons is given below. For convenience the SA, I and SS neurons associated with a symbol is referred to as symbol group (SG).

Setup Cores and Neurons

```
numReserveAxons ← (2 + number of SA neurons)
NurnLst ← All neurons in lexicon //UL,LL,Ex,Con_Splits,SG_1-n
NurnLst_itr ← begin(NurnLst) // initialize iterator
splitsDone ← false
done ← false
while not done
    ADD Core
    RESERVE axons for SA, UL, LL and EX in the core
    for i ← 1 to 256
        curNurn ← NurnLst(NurnLst_itr)
        DETERMINE resources required, numNeurons for splits, numAxons for all
        other neurons
        if resource available in core then
            ASSIGN neuron type
            if (curNurn = UL or LL) then
                CREATE crossbar connection from the SA axon to UL and LL
            else if (curNurn = Ex) then
                CREATE crossbar connection from the LL axon to Ex
            else if(curNurn = SS) then
                CREATE crossbar connection from the corelet input axons to SS
            else if(curNurn = SA) then
                CREATE crossbar connection from SS, I, Ex to SA
            else if(curNurn = I) then
                CREATE crossbar connection from SA, UL to I
            else if(curNurn = split) then
                CREATE crossbar connections from given axon.
            INCREMENT NurnLst_itr
            if (end(NurnLst) = NurnLst_itr) then //check for end of list
                if splitsDone then
                    done ← true
                    Break
                else
                    splitsDone ← true
                    COMPUTE number of splits required for UL, Ex and SA neurons to
                    support number of cores created
                    NurnLst ←split neurons to feed reserved axons
                    NurnLst_itr ← begin(NurnLst) // initialize iterator
        else
            Break
```

The core and neuron setup process involves making crossbar connections and configuring neurons. The first step in this process as shown in the pseudo code is to determine the number of axons which have to be reserved in a core. These axons will be shared among all neurons of the core. Then a neuron list is prepared which involves all neurons in the lexicon including the splits for connectors. Neurons from this list are sequentially added to cores and appropriate synapses are configured. If a neuron cannot be accommodated due to a lack of enough neurons for making splits or due to not enough axons for all other neuron types, then an additional core is added and the process continues. Once all the neurons are assigned then the number of splits required to support the reserved axons in each core is computed and these additional inter-core split neurons are assigned using the same method as described above. The resulting core allocation is as shown in Fig. 9. Hence the number of cores and neurons instantiated are based on the network topology.

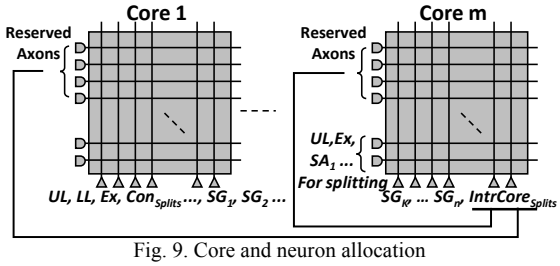


Fig. 9. Core and neuron allocation

After the core and neuron configuration is done the connectivity inside cores is now established. This involves linking the neuron outputs to axons. The connectivity between input connector and axons is established followed by connectivity between neurons and output connectors. Any unused neurons in all cores are configured to be disconnected. This completes the corelet creation process, which is repeated till all the lexicons are flattened. In this way we map the network in a compact manor to TrueNorth.

E. Creating connected corelets

After creating all the corelets they are connected based on the connectivity information between lexicons which represents KLs, resulting in a fully formed stochastic SNN. This list of corelets is compiled and run on both the compass simulator and on the TrueNorth chip.

The example sentence utilizes 13 cores with the above algorithm. In general, the number of corelets generated will be equal to the number of lexicons in the sentence and each corelet will use as many cores required to map all the symbols in the lexicon.

V. Design Environment

In this section we briefly describe the tools used and the design environment details which is as shown in Fig. 10. The input of the design environment is the confabulation model and the trained knowledge base, which provides lexicon details and knowledge link information. The network creator generates the reference network and

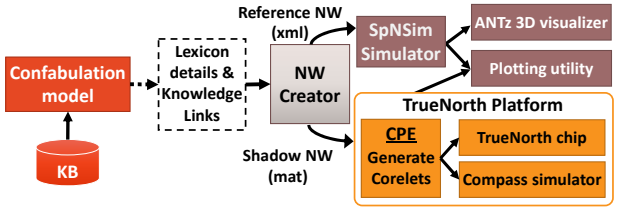


Fig. 10. Design Environment

the shadow network.

The reference network is sent to an in-house spiking neural network simulator, SpNSim for functional verification [19]. SpNSim is a multithreaded and scalable simulation platform built using C++. It is flexible and vectorized for efficient evaluation and capable of handling large-scale networks of mixed neuron models. It is also capable of training stochastic SNNs using STDP. In SpNSim, the network topology along with the network parameters are specified through a XML input file. The XML input file also includes definition of runtime parameters to specify the dynamic behavior (e.g. the starting and ending of learning phase, test phase, etc.) of the network. SpNSim is capable of generating visualization data which is used to render 3D neural networks using an open source 3D data visualization tool called ANTz. This is a very helpful feature for debugging and analyzing complex neural networks. There is an accompanying plotting utility developed in MATLAB for plotting and analyzing spikes. It is also capable of analyzing neuron parameters and visualizing weight evolution for debug purpose.

The tools that convert the reference network to the shadow network are developed in C++. The shadow network is sent to CPE where it is flattened and corelets are generated. Since CPE is a MATLAB based tool, the shadow network is saved as a MAT file, which contains the neuron details, weights, connectivity and network topology information required for creating corelets. Finally, the connected corelets are simulated using the IBM compass simulator or executed on the TrueNorth processor. After evaluating the neural network we convert the TrueNorth spike files to SpNSim spike file format and create raster plots and signal to noise ratio (SNR) plots using the plotting utility of SpNSim.

VI. Experiments and Results

Experiments have been carried out to validate the functionality of the TrueNorth implementations generated from the above design flow. Multiple networks are built, each capable of confabulating one sentence from a set of possible words. These networks do not have inputs, the stochastic firing gets amplified due to resonance and produce meaningful sentences as outputs.

A random set of 100 sentences from "Ali Baba and Forty Thieves" were picked from noisy document images as our test cases. This text was not used for training. The reference networks were simulated using SpNSim and the results are compared to the output of the SNN running on TrueNorth. For 85% of the test cases, TrueNorth generated the same sentence as the simulated reference network with a sentence accuracy of 70%. Fig. 11 shows the raster plot for results obtained from TrueNorth. The labels and axes represent the same information as in Fig. 4 except that the Y-axis labels show the corelet connector pin numbers.

There is no timing control on the neurons to reset its operation over window intervals. These are free running neurons that amplify or suppress the excitation inherently. Therefore, window analysis is performed as post processing on the collected spikes to determine a feasible window over which the spikes must be accumulated to get statistically significant results. It is important to note that, these are resonant

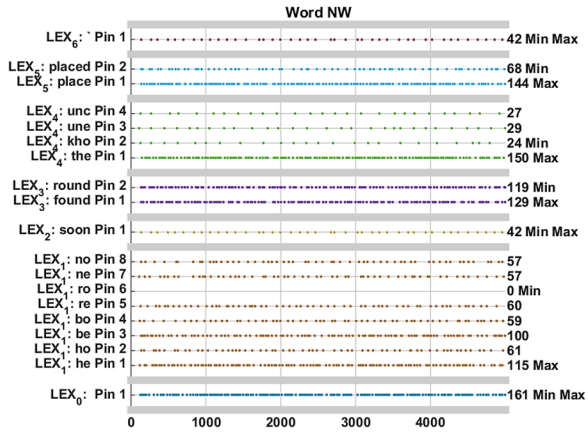


Fig. 11. TrueNorth sentence results

networks without external stimulus resulting in sparse spiking patterns. When stimulated by external inputs, corresponding neurons and their downstream neighbors will increase the spiking density. Also the spiking density can be tuned by varying the thresholds of UL and LL neurons, based on the application requirement. For window analysis, the number of spikes is counted for each window and plotted. The gap between the desired curve and rest of the curves represents the signal to noise ratio (SNR). The larger the gap, the higher is the SNR for detection. The effects of sampling the output for different window sizes is shown in Fig. 12 for lexicon 1. From the figure it is evident that a window size of about 500 ticks is enough to reliably determine the confabulated result, which is used in all the experiments.

While making the network compatible to TrueNorth, we had to scale and quantize the weights. By increasing the bin width, more connections will share the same weight and will result in lower performance. This effect is shown in Fig. 13 for different bin widths. Since these networks are small the effect is small but the trend is visible. As a rule of thumb, reducing bin width increases precision but at the cost of additional resources.

After programming the TrueNorth chip we measure the power consumption to run the network as characterized in [30]. To find the actual power consumption just for the resources utilized on the chip, first the leakage power P_{leak} is measured when the system is idle then another power measurement P_{total} is made with the network running. The active power is computed as $P_{active} = P_{total} - P_{leak}$. The leakage power is scaled to get the leakage power for only the cores utilized out of 4096 cores, $P_{leak\_s} = P_{leak} * NumCores / 4096$. Finally, the total scaled power is computed as $P_{total\_s} = P_{active} + P_{leak\_s}$. Power measurements were made for four sentences, each for all the four bin widths. Hence 16 networks were used to measure power. On average the total scaled

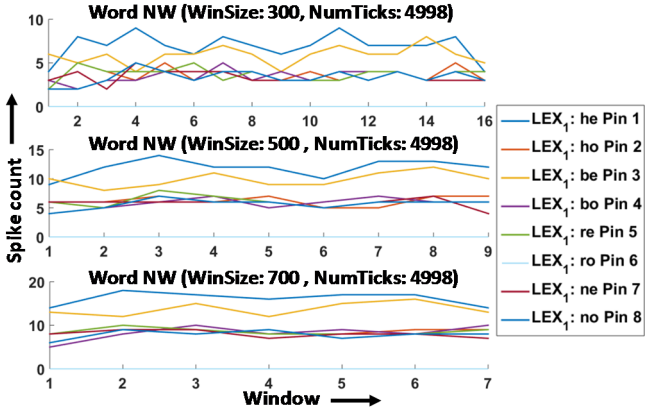


Fig. 12. Effect of window size on lexicon 1

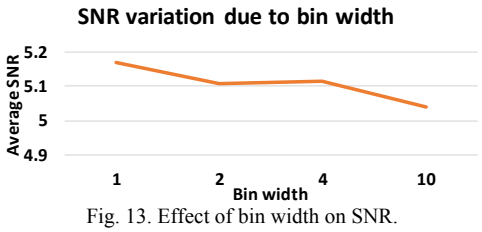


Fig. 13. Effect of bin width on SNR.

power is 0.205mW when running the chip at 0.8V and 0.442mW when running the chip at 1.0V.

VII. Conclusion

In this paper we have demonstrated that a stochastic integrate and fire neuron model can be used instead of more complex Bayesian neuron model for inference related tasks in spiking neural networks. To achieve this, we have proposed a normalized winner-take-all network topology. We have implemented several examples to verify that both kinds of networks, ones with Bayesian neuron model and another with a stochastic integrate and fire neuron model produce similar results. The stochastic integrate and fire neuron model based network has been successfully programmed to the TrueNorth chip and evaluated. We have shown that Bayesian inference computation can be performed in very low power and efficient manner by performing a sentence confabulation task using spiking neural networks. Finally, the proposed normalized winner-take-all concept will be used as a building blocks for more complex inference networks in future.

Acknowledgement

Received and approved for public release by AFRL on 3/29/2016, case number 88ABW-2016-1574. Any Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL or its contractors. We would like to thank Andrew Cassidy and Rodrigo Alvarez-Icaza for helping us perform network level power analysis on TrueNorth.

References

[1] Ananthanarayanan, R.; Esser, S.K.; Simon, H.D.; Modha, D.S., "The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses," in *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on* , vol., no., pp.1-12, 14-20 Nov. 2009

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

[3] 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.

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

[5] T. A. Engel, W. Chaisangmongkon, D. J. Freedman, and X. J. Wang, "Choice-correlated Activity Fluctuations Underlie Learning of Neuronal Category Representation," *Nature Communications* 6, Mar., 2015.

[6] Lee, Honglak, Peter Pham, Yan Largman, and Andrew Y. Ng. "Unsupervised feature learning for audio classification using convolutional deep belief networks." In *Advances in neural information processing systems*, pp. 1096-1104. 2009.

[7] Lee, Honglak, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. "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.

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

[9] Merolla, Paul A., John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface." *Science* 345, no. 6197 (2014): 668-673.

[10] Amir, Arnon, Piyali Datta, William P. Risk, Andrew S. Cassidy, Jeffrey Kusnitz, Steven K. Esser, Alexander Andreopoulos et al. "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores." In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pp. 1-10. IEEE, 2013.

[11] Esser, S.K.; Andreopoulos, A.; Appuswamy, R.; Datta, P.; Barch, D.; Amir, A.; Arthur, J.; et al., "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, vol., no., pp.1-10, 4-9 Aug. 2013

[12] Diehl, Peter U., Guido Zarrella, Andrew Cassidy, Bruno U. Pedroni, and Emre Neftci. "Conversion of Artificial Recurrent Neural Networks to Spiking Neural Networks for Low-power Neuromorphic Hardware." *arXiv preprint arXiv:1601.04187* (2016).

[13] Diehl, Peter U., Bruno U. Pedroni, Andrew Cassidy, Paul Merolla, Emre Neftci, and Guido Zarrella. "TrueHappiness: Neuromorphic Emotion Recognition on TrueNorth." *arXiv preprint arXiv:1601.04183* (2016).

[14] 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

[15] 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

[16] 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.

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

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

[19] Khadeer Ahmed, Amar Shrestha and Qinru Qiu, "Simulation of Bayesian Learning and Inference on Distributed Stochastic Spiking Neural Networks," *2016 International Joint Conference on Neural Networks (IJCNN)*, in press.

[20] Benjamin, B.V.; Peiran Gao; McQuinn, E.; Choudhary, S.; Chandrasekaran, A.R.; Bussat, J.-M.; Alvarez-Icaza, R.; Arthur, J.V.; Merolla, P.A.; Boahen, K., "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," in *Proceedings of the IEEE*, vol.102, no.5, pp.699-716, May 2014

[21] Schemmel, J.; Brüderle, D.; Grübl, A.; Hock, M.; Meier, K.; Millner, S., "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, vol., no., pp.1947-1950, May 30 2010-June 2 2010

[22] Stromatias, Evangelos, et al. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms." *Frontiers in neuroscience* 9 (2015).

[23] Neil, Dan, and Shih-Chii Liu. "Minitaur, an event-driven FPGA-based spiking network accelerator."*Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 22, no. 12 (2014): 2621-2628.

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

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

[26] Hecht-Nielsen, Robert. *Confabulation theory: the mechanism of thought*. Heidelberg: Springer, 2007.

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

[28] 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

[29] Preissl, R.; Wong, T.M.; Datta, P.; Flickner, M.; Singh, R.; Esser, S.K.; Risk, W.P.; Simon, H.D.; Modha, D.S., "Compass: A scalable simulator for an architecture for cognitive computing," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for* , vol., no., pp.1-11, 10-16 Nov. 2012

[30] Cassidy, Andrew S., Rodrigo Alvarez-Icaza, Filipp Akopyan, Jun Sawada, John V. Arthur, Paul A. Merolla, Pallab Datta et al. "Real-time scalable cortical computing at 46 giga-synaptic OPS/watt with." In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 27-38. IEEE Press, 2014