Neuromorphic Adaptations of Restricted Boltzmann Machines and Deep Belief Networks

Bruno U. Pedroni, Srinjoy Das, Emre Neftci, Kenneth Kreutz-Delgado, and Gert Cauwenberghs

Abstract—Restricted Boltzmann Machines (RBMs) and Deep Belief Networks (DBNs) have been demonstrated to perform efficiently on a variety of applications, such as dimensionality reduction and classification. Implementation of RBMs on neuromorphic platforms, which emulate large-scale networks of spiking neurons, has significant advantages from concurrency and low-power perspectives. This work outlines a neuromorphic adaptation of the RBM, which uses a recently proposed neural sampling algorithm (Buesing et al. 2011), and examines its algorithmic efficiency. Results show the feasibility of such alterations, which will serve as a guide for future implementation of such algorithms in neuromorphic very large scale integration (VLSI) platforms.

I. INTRODUCTION

THE current state-of-the-art in many machine learning tasks involves Restricted Boltzmann Machines (RBMs) [1] and associated algorithms (e.g. Deep Belief Networks (DBNs)). Neurobiological hardware architectures, or neuromorphic hardware, can provide a low-power, massively parallel substrate for implementing these algorithms, yet how they can be mapped onto a biologically plausible spiking neural network is still a relatively unexplored area.

The two main reasons for understanding the bridge between machine learning algorithms and hardware implementations lie in processing and power. Compared to conventional approaches for implementing machine learning algorithms, such as standard processors, GPUs, or FPGAs, neuromorphic platforms provide a massively parallel continuous-time framework suitable for implementing such algorithms. Besides this, typical neuromorphic VLSI (Very Large-Scale Integration) MOS (metal-oxide semiconductor) circuits operate in subthreshold region, which provide extremely low-power regimes for operation of such algorithms [2].

To the best of our knowledge, this is the first paper that discusses how stochastic sampling techniques like Markov Chain Monte Carlo (MCMC) implemented using dynamics of spiking neurons can be used to perform learning in RBMs and DBNs. We analyze some of the issues involved in neuromorphic implementation of RBMs and DBNs, taking into account several of the limitations and characteristics unique to these types of hardware platforms. Due mainly to the unviability of directly obtaining spiking probabilities from the neurons and to the impracticality of preserving part of the neural network static (“clamping”) between “wake” and “sleep” phases during RBM training, these considerations play a strong role in modeling the neuromorphic hardware and are explained in detail in Section III.

Prior, an overview of Boltzmann and Restricted Boltzmann Machines will be shown in Section II. The experimental results of the individual considerations will be presented in Section IV, with conclusions and future work shown in Section V.

II. RESTRICTED BOLTZMANN MACHINES

A. Boltzmann Machines

A Boltzmann machine (BM) is a stochastic neural network where binary activation of “neuron”-like units depends on the other units they are connected to. A typical BM contains 2 layers - a set of visible units \( v \) and a set of hidden units \( h \). The machine learns arbitrary distributions of input vectors which are fed directly to the visible units. Neurons in both layers can be connected at the same level or with those from other layers. The weights are symmetric between all neurons and there are no self-connections for any neurons. Figure 1 demonstrates the general structure of a BM with 4 visible and 3 hidden units. Boltzmann machines in theory can be used for a whole range of learning tasks, like classification and combinatorial optimization problems [3].

For each data sample, the BM is trained in two phases. At first (“wake” phase) the states of the visible units are clamped to a particular vector sampled from the training set. Using a procedure called Gibbs sampling, values for the hidden units are generated. The hidden unit switches on based on its inputs from the visible units with a probability given by the logistic function (in (1), the argument to logistic is the sum of weights of all connected neurons). The generation of this stochastic value for each hidden unit (the visible layer being “clamped” at this stage) is conditioned on the given values of the other neurons for the hidden layer at that
time. In the second step ("sleep" phase) the network is allowed to run freely, and the hidden units attempt to "reconstruct" the visible data. This involves sampling from the joint probability distribution of the visible and hidden units (using the logistic activation as in the wake phase), which is done by a Markov Chain Monte Carlo (MCMC) method using alternating sampling between the visible and hidden units. The procedure is run until the network reaches equilibrium [4].

\[
P(t) = \frac{1}{1 + e^{-t}}
\]  

(1)

The equations for the weight update in BMs are shown in (2) and (3). Here, \( L \) denotes the likelihood function, which is optimized to find the optimum weight matrix (\( w_{ij} \)). \( x \) represents values of the visible units, while \( z \) represents the hidden units. \(<\cdot\rangle\) denotes expectation taken over pairs of these variables. \( P(z|x) \) denotes the probability distribution of the hidden units (\( z \)) conditioned on the visible units (\( x \)) for which sampling is done in the wake phase of the algorithm. \( P(x,z) \) denotes the joint probability distribution of the visible (\( x \)) and hidden (\( z \)) units, which is estimated by MCMC sampling during the sleep phase.

\[
\frac{\partial L}{\partial w_{ij}} = \sum_{x,x'} \left[ \langle x'z' \rangle_+ - \langle x'z' \rangle_- \right]
\]  

(2)

\[
\langle x'z' \rangle_+ = \int x'z' p(z|x)dz
\]  

(3)

\[
\langle x'z' \rangle_- = \int x'z' p(x,z)dxdz
\]  

(4)

However, for BMs this method of training presents some practical issues. One of the primary challenges is the prohibitively high time taken by the machine in order to collect equilibrium statistics over the data distribution it attempts to learn during each of the wake and sleep phases. This time can grow exponentially with the size of the machine and with the magnitude of the connection strengths.

B. Restricted Boltzmann Machines and the Contrastive Divergence algorithm

RBMs are a particular case of BMs where the connectivity between units of the same layer is not allowed for simplification of learning [5] (see Figure 2). This feature helps accelerate the training phase by allowing for parallel or “block” Gibbs sampling at each layer. This means that in the wake phase, when the visible units are clamped to the input data, the hidden units are statistically independent of each other and therefore random samples for these are generated in parallel, independent of values of neighboring units in that layer. In addition to this, the total time for sampling during training is also shortened by using an algorithm called “Contrastive Divergence” (CD) [6].

In Contrastive Divergence, the visible unit reconstruction in the sleep phase is done by sampling conditioned on the values of the hidden units for that particular iteration. This procedure can be repeated several times, but relatively good convergence is obtained for the equilibrium distribution even for one iteration (CD-1, shown in Figure 3).

Equation (5) shows the weight update in RBMs using CD. Here, \( w_{ij} \) denotes the weights between visible unit \( v_i \) and hidden unit \( h_j \), and \( \langle \cdot \rangle \) denotes expectations taken over \( v_i, h_j \) pairs during wake phase (visible layer clamped to input data) and sleep/recon phase (visible layer reconstructed from active values on hidden units). The learning rate is represented by \( \eta \).

\[
\Delta w_{ij} = \eta \left( \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{recon}} \right)
\]  

(5)

C. Deep Belief Networks

Constructed by stacking RBMs and performing layer-by-layer training, DBNs use the hidden unit activations of one layer as the training examples for the next layer [7]. The structure of a DBN is shown in Figure 4, where the \( h_1 \) layer will act as the visible layer for the \( h_1-h_2 \) pair, while the \( h_2 \) layer will act as the visible layer for the \( h_2-h_3 \) pair. The training between these layers can use samples (0 or 1) or mean activations (spiking probabilities) of the units. The label units (not shown) will be used for training the last layer of the DBN when a discriminative RBM is used (explained in Section III-C).
III. CONSIDERATIONS FOR NEUROMORPHIC RBM IMPLEMENTATION

Several changes are necessary for implementation of the Restricted Boltzmann Machine on a neuromorphic platform. These considerations are required for implementing the algorithm in an “online” manner using the hardware resources to the extent possible. Implementation using such an approach can result in realization of the low-power, concurrent advantage outlined in Section I.

A. Neural sampling

The sampling technique for the conventional, or machine learning (ML), RBM uses Gibbs sampling. Recently, Buesing et al. [8] have proposed a method called “neural sampling” where it is shown that under some conditions the stochastic firing activity of spiking-neuron networks can be interpreted as probabilistic inference via MCMC sampling. This framework, which uses a time-irreversible Markov chain, works in both discrete and continuous time and is described briefly below.

In the discrete time framework shown in Figure 5, each neuron has a refractory period \( \tau \), during which it is not allowed to spike (states 2 to \( \tau \)). At states 0 and 1, the neuron can spike with a probability which mimics the logistic function of the ML case but with a scaling factor of \( \tau \). The effect of the spike is assumed to last for \( \tau \) steps. Constructing the transition operator in this manner allows it to converge to the equilibrium distribution without the time-reversibility assumption made in the Gibbs case. Each step of sampling (visible or hidden neurons) involves running this operator shown in Figure 5 for a fixed duration of time steps greater than \( \tau \).

B. Weight updates using states

The Contrastive Divergence algorithm used in the ML implementation of the RBMs uses a combination of states (0 or 1) and probabilities of transition for visible/hidden units when updating the weights and biases. In the neural sampling framework, neuron activities represent realizations of the random variable. As a consequence, the system does not have access to true probabilities. For this reason, one strategy for weight updates is to use the states (0 or 1) of both the visible and hidden units.

C. Generative versus discriminative RBM

For classification purposes, given a labeled dataset, RBMs can be used in two different ways:

a) Use the RBM purely as a generative model to learn the probability distribution of the dataset during training and use the activation of the hidden units to train an offline classifier which uses the labels. During the test phase, the activations of the hidden units are used to feed into this offline classifier to predict the data labels.

b) Train the RBM in a discriminative framework using both the dataset and its labels [9]. In this architecture, the visible units receive both the data \( x \) and labels \( y \) (see Figure 6). During the test phase, the free energy is calculated offline over all labels and the minimum value is selected as the prediction for the data label.

![Fig. 5. Neural sampling (discrete time) with refractory period of \( \tau \) steps [8].](image)

![Fig. 6. Visible units (data \( x \) and labels \( y \)) for discriminative RBM [9].](image)

For ease of implementation on a neuromorphic platform approach (b) was chosen.

D. Softmax versus binary sampling

The discriminative RBM outlined in Section III-C implements a one-hot representation of the label units where the number of visible label units is equal to the number of data labels (i.e., classes). For this representation a softmax (multinomial) sampling step is required, which cannot be generated directly using the statistics of the neuronal firing as described in Section III-C and may require additional circuit overhead for online implementation. Using a simple encoding of the visible units representing the labels, this overhead can be avoided since the sampling procedure for the label units then uses the binary statistics generated by the spiking neurons. This method is identical to that used for the visible units representing the data. This approach is also useful when the number of classes is very high and, thus, it may not be feasible to implement a corresponding large number of label neurons.

E. Spike averaging

Training in RBMs occurs by updating the weights of neuron connections. This update takes into account the spiking probabilities of neurons, which, unfortunately, is physically unviable. A strategy used to overcome this issue considers spike rates of neurons in gradient calculations. This is attractive not only because hard decisions (neuron states, as outlined in Section III-B) are replaced by soft decisions (spike rates) in the weight update, but also is feasible in terms of hardware. As will be seen in Section IV-I, this implementation also contributes to lower error rate variance.
F. Bayesian network model

The acyclic properties of Bayesian models were also explored in the experiments. To eliminate the clamping steps of the Contrastive Divergence algorithm, the multiple layers of neurons can be sequentially linked, with a hidden-visible double layer for each $k$ step of CD-$k$. Figure 7(a) shows the structure for training an RBM using CD-2. In this fashion, the only clamping that occurs is at the visible layer, while the hidden layers and the reconstruction layers remain in a “free-running” mode. Additionally, the neural Bayesian network model is completely viable in practical hardware implementations; this greatly simplifies processing (only samples from the first and last two layers are required for the weight updates) and reduces data traffic.

Besides training of the RBM, a Bayesian model can be used for prediction (testing). Figure 7(b) illustrates the clamping at the bottom visible layer (data) and the free-running mode up to the label neurons, which will generate a prediction based on the input data.

IV. EXPERIMENTS AND RESULTS

A. Test case for the RBMs and DBNs

The realized tests were a classification task of 1,000 MNIST dataset digits, with the RBMs and DBNs trained using a 5,000 sample MNIST dataset. The data was divided into 50 batches of 100 samples each, with the graphical results in Sections IV-C through F showing the number of errors along the epochs of training. For each of the epochs, the prediction was performed using the same 1,000 test samples.

Each MNIST sample is a 28x28 pixel image (totaling 784 visible units) of a digit ranging from 0 to 9. The number of hidden units was set at 500, resulting, therefore, in a weight matrix of 784 lines (visible units) by 500 columns (hidden units).

B. Methodology

Matlab was used to test the NM-RBM considering all the neuromorphic adaptations with respect to the corresponding machine learning version. The robustness of these changes and their impact on RBM performance for classification were verified based on prediction error rate. The neural sampling technique (shown in Section III-A) was tested in five scenarios: CD-2, CD-$k$, spike averaging, Bayesian network, and DBN. The reduced error variance during training of the spike averaging model is also verified.

C. Results for Machine Learning RBM with CD-2

Out of the 1,000 test samples, the ML-RBM (see Figure 8) presented an average of 65 errors in the 50th epoch, with a minimum average of 64 errors. This represents an accuracy of 93.6%.

D. Results for NM-RBM with CD-2

Out of the 1,000 test samples, the NM-RBM (see Figure 9) presented an average of 68 errors in the 50th epoch, with a minimum average of 65 errors. This represents an accuracy of 93.5%, which is very similar to the ML-RBM, and shows that neural sampling performs just as well as Gibbs sampling.

E. Results for NM-RBM with CD-$k$

The tests for the NM-RBM with CD-2, CD-3, and CD-4 are shown in Figure 10. This shows that a convergence is more quickly obtained when executing the Contrastive Divergence algorithm with more steps (larger $k$).
F. Results for NM-RBM with spike averaging

The tests for the NM-RBM with spiking statistics are shown in Figure 11. Here, ML-RBM is the neural sampling shown in Section IV-D, and spike(10) and spike(20) represent simulations with spike averages performed after 10 and 20 refractory cycles, respectively. This approach gives lower variance in error rates by reducing sampling noise which can occur when states are used in gradient calculations.

G. Results for Bayesian NM-RBM

The acyclic properties of Bayesian networks were tested in a neural network with a more “continuous” sampling procedure. The simulations for a model with CD-2 (see Figure 12) presented – out of the 1,000 test samples – an average of 65 errors in the 50th epoch, with a minimum average of 65 errors. This represents an accuracy of 93.5%, which is very similar to the ML-RBM and the NM-RBM. These results, along with the qualities presented in Section III-E, place the Bayesian NM-RBM model as the most probable candidate for a basis of a neuromorphic implementation.

H. Results for NM-RBM on DBN

The results for simulations of two-layered deep belief networks are shown Figure 13. The DBNs presented clear improvement in error rate over RBMs, even when preserving the same number of hidden units as in the other tests (as in the case of 500 hidden units for NM DBN 250-250).

I. Error variance comparison

The error variance comparison between ML-RBM, NM-RBM, and NM-RBM with spike averaging models is shown in Figure 14. It is clear that using soft decision (spike rates) in the gradient calculation accounts for error rates with lower variances.
V. CONCLUSIONS AND FUTURE WORK

A. Conclusions

The work presented here provides insights into how stochastic sampling techniques like Markov Chain Monte Carlo (MCMC) implemented using dynamics of spiking neurons can be used to perform learning in RBMs and DBNs. This consequently leads to continuous-time, low-power implementations of these algorithms on neuromorphic VLSI platforms.

The results in Section IV also demonstrate that Contrastive Divergence, which was originally developed in the Machine Learning Gibbs sampling framework, gives satisfactory results for neural sampling. Similarly, weight updates made in the neuromorphic framework using either states or probability estimates generated from spike averaging result in comparable performance to Gibbs sampling.

The spike averaging and the Bayesian network models of the neural RBM presented promising results for neuromorphic implementations. The Bayesian network model considered a more continuous-time structure of the algorithm and also showed that the algorithmic performance was preserved, besides reducing error variance.

B. Future work

Further neuromorphic enhancements in the framework presented include consideration of fixed bit-width weights in hardware – instead of arbitrary precisions, sparsification of the weight matrix owing to a limited number of synapses available on VLSI neuromorphic hardware, and possible implementation of weight updates in a fully online manner using STDP (spike-timing dependent plasticity).

REFERENCES