Implementation of a Restricted Boltzmann Machine in a Spiking Neural Network

Srinjoy Das — **Author**
Department of Electrical and Computer Engineering — Student
University of California, San Diego — UCSD
srinjoyd@gmail.com

Bruno Umbria Pedroni — **Author**
Department of Bioengineering — Student
University of California, San Diego — UCSD
bpedroni@ucsd.edu

Abstract

Restricted Boltzmann Machines (RBMs) have been demonstrated to perform efficiently on a variety of applications, such as dimensionality reduction and classification. Implementing RBMs on neuromorphic hardware has certain advantages, particularly from a concurrency and low-power perspective. This paper outlines some of the requirements involved for neuromorphic adaptation of an RBM and attempts to address these issues with suitably targeted modifications for sampling and weight updates. Results show the feasibility of such alterations which will serve as a guide for future implementation of such algorithms in VLSI arrays of spiking neurons.

1 Introduction

1.1 Machine learning in a neuromorphic framework

Restricted Boltzmann Machines (RBM) [1] and associated algorithms (e.g. Deep Belief Networks (DBNs)) are the current state-of-the-art in many machine learning tasks. 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. This paper addresses some of the issues involved for neuromorphic implementation of RBMs taking into account some of the limitations and characteristics unique to hardware platforms.

1.2 Motivation and advantages

Compared to conventional approaches for implementing machine learning algorithms in software, GPU (Graphical Processing Units) or FPGAs (field-programmable gate arrays), neuromorphic platforms provide a massively parallel continuous-time framework suitable for implementing such algorithms. 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].

2 Restricted Boltzmann Machines

2.1 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 distribution 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. Weights between connected neurons are symmetric between hidden and visible layers. 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].

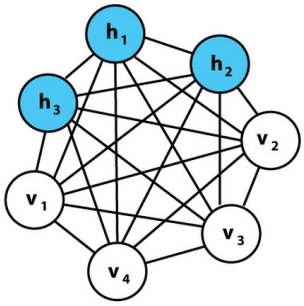

Figure 1: Boltzmann Machine with 4 visible and 3 hidden units

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 (Eq 1, argument to logistic is 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].

$$P(t) = \frac{1}{1 + e^{-t}} \tag{1}$$

The equations for the weight update in BMs are shown below. Here $\mathcal{L}$ 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. <> 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. The weight updates are realized by Eq. 2 and 3.

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{x \in X} [\langle x^i z^j \rangle_+ - \langle x^i z^j \rangle_-] \tag{2}$$

$$\langle x^i z^j \rangle_+ \stackrel{\text{def}}{=} \int x^i z^j p(\mathbf{z}|\mathbf{x}) d\mathbf{z}, \tag{3}$$

$$\langle x^i z^j \rangle_- \stackrel{\text{def}}{=} \int x^i z^j p(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z}.$$

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.

2.2 Restricted Boltzmann machines and Contrastive Divergence

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.

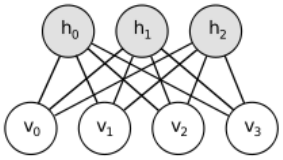

Figure 2: Restricted Boltzmann Machine with 4 visible and 3 hidden units

In addition to the above, the total time for sampling during training is also shortened by using an algorithm called "Contrastive Divergence" (CD) [6]. In this process, 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).

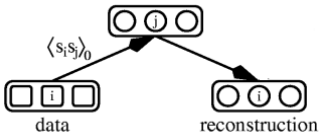

Figure 3: Wake and sleep phases of the Contrastive Divergence algorithm

The equation for weight update in RBMs using CD is shown below.

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}) \tag{4}$$

Here w_{ij} denotes the weight between visible unit v_i and hidden unit h_j, and $< >$ 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 ϵ.

3 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 1.1.

3.1 Neural sampling

The sampling technique for the conventional or machine learning (ML) RBM uses Gibbs sampling. The Gibbs transition operator assumes the system to be time-reversible, which is not true if the dynamics of the neurons are used to generate the samples on a neuromorphic platform. Recently, Buesing *et. al* [7] have proposed a method called "neural sampling" which works with time-irreversible systems and where it is shown that under some conditions the stochastic firing activity of networks of spiking neurons can be interpreted as probabilistic inference via MCMC sampling. This framework, which works in both discrete and continuous time, uses a different kernel from Gibbs sampling which is described briefly below.

In the discrete time framework shown in Figure 4, each neuron has a refractory period τ during which it is not allowed to spike (states 1 to τ). At state 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 τ. The effect of the spike is assumed to last for τ 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 4 for a fixed duration of time steps greater than τ.

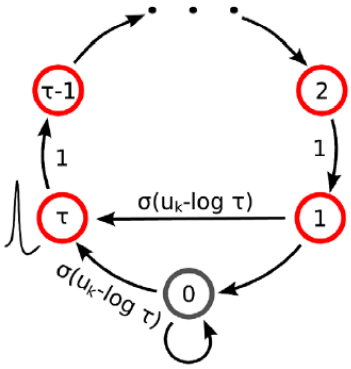

Figure 4: Neural sampling (discrete time) with refractory period of τ steps [7]

3.2 Weight updates using states

The Contrastive Divergence algorithm used in the ML implementation of the RBMs uses a combination of states and probabilities of transition for visible/hidden units when updating the weights and biases. Since obtaining the probability values of the visible and hidden units in a physical neuromorphic application of the RBM may not be easily realizable it was decided to use the states of both the visible and hidden units for update purposes.

3.3 Discriminative versus generative 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. In this architecture the visible units receive both the data and labels (see Figure 5). 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.

To maximize the number of online operations on a neuromorphic platform, the approach (b) outlined above was chosen.

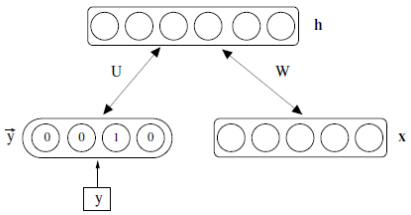

Figure 5: Visible units (data *x* and labels *y*) for discriminative RBM

3.4 Softmax versus binary sampling

For the discriminative RBM outlined in 3.3 (b), if the number of visible units is equal to the number of data labels, then a softmax (multinomial) sampling step is required, which cannot be generated using the statistics of the neuronal firing as described in 3.1 and may require additional circuit overhead for online implementation. Using a simple encoding of the visible units representing the labels (i.e. follow an encoding scheme which uses ceil($\log_2 n$) units instead of n units), this overhead can be reduced as the sampling procedure then becomes identical to that used for the visible units representing the raw data. Both approaches have been tested to examine the tradeoff between reducing complexity of operations versus classification accuracy.

4 Experiments and results

4.1 Test case for neuromorphic RBM (NRBM)

The realized tests were a classification task of 1,000 MNIST dataset digits, with the RBMs 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 4.3-4.6 showing the number of errors (among the 1,000 test samples) along the epochs of training.

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). Figure 6 shows three examples of binary representations of MNIST dataset digits.

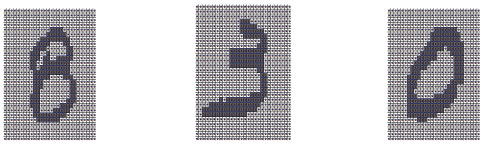

Figure 6: Three examples of MNIST dataset digits

4.2 Methodology

Both Matlab and BRIAN were used to test the NRBM considering all the neuromorphic adaptations with respect to the corresponding machine learning version. The algorithmic modifications (items listed in Section 3) were tested using Matlab to verify the robustness of these changes and their impact on RBM performance for classification. Following this, BRIAN was used to verify that the statistics of neuronal firing using stochastic integrate and fire models match those of Matlab using the neural sampling approach. In this way, computationally expensive verification is done on Matlab while BRIAN is used to test the viability of the modified scheme in a continuous time framework using more biologically and neuromorphically realistic models with stochastic inputs.

4.3 Machine Learning RBM

The tests for the ML RBM presented an average of 68 errors (of 1,000 samples), with an average minimum number of errors of 55 (see Figure 7). This results in an accuracy of 93.2%.

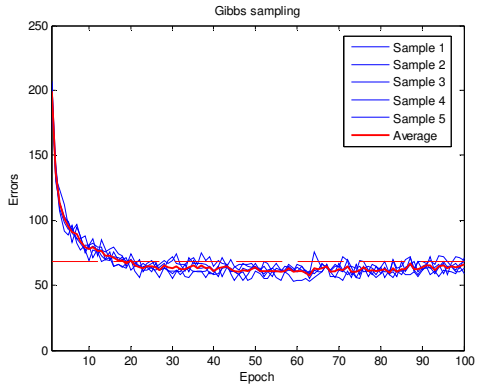

Figure 7: Results for Machine Learning RBM with CD-1

4.4 Results for NRBM with CD-1

The tests for the NRBM with CD-1 presented an average of 103 errors (of 1,000 samples), with an average minimum number of errors of 87 (see Figure 8). This results in an accuracy of 89.7%.

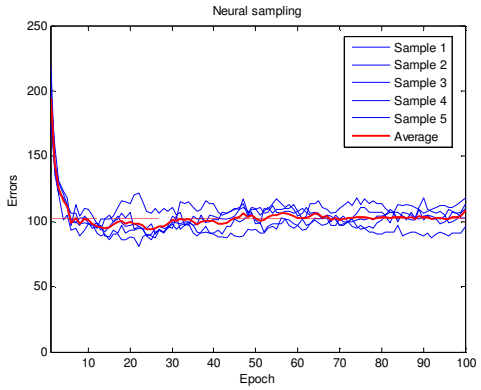

Figure 8: Results for NRBM with CD-1

4.5 Results for NRBM with CD-2

The tests for the NRBM with CD-2 (Contrastive Divergence runs 2 times for every data sample during training) presented an average of 87 errors (of 1,000 samples), with an average minimum number of errors of 70 (see Figure 9). This results in an accuracy of 91.3%.

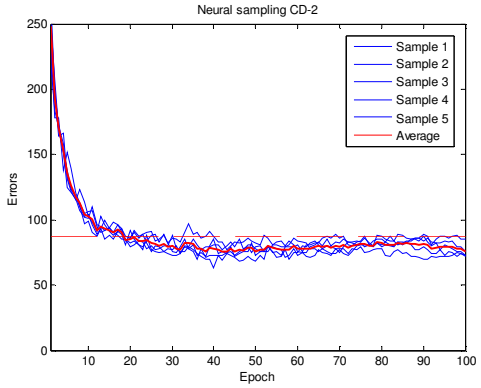

Figure 9: Results for NRBM with CD-2

4.6 Results for NRBM with binary classifier sampling and CD-2

The tests for the NRBM with binary classifier sampling (refer Section 3.4) and CD-2 presented an average of 193 errors (of 1,000 samples), with an average minimum number of errors of 150 (see Figure 10). This results in an accuracy of 85% using average minimum.

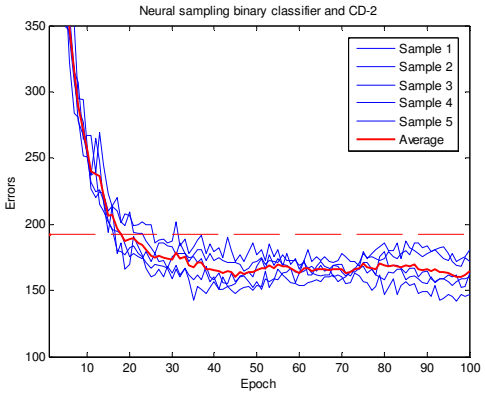

Figure 10: Results for NRBM with binary classifier and CD-2

4.7 Results for BRIAN

BRIAN modeling was done by using stochastic integrate and fire neuron models to test the validity of the neural sampling results in continuous time. Instead of running CD over the whole training and then using the results on test datasets, a comparison was done between the Matlab and BRIAN models for hidden unit activation based on 10 randomly chosen MNIST test data samples. The hidden unit computations were done using the same weight matrix and test data for both simulators. For BRIAN the number of spikes generated over the length of the test was used to determine the neuron's activation probability. For Matlab this was done by binary sampling over the logistic activation probability of the hidden units. Table 1 shows the total number of mismatches, between Matlab and BRIAN, over all 500 hidden units. As shown in the table, the delta is less than 10%, which demonstrates that the neural sampling algorithm can be used in a continuous-time framework without any significant impact on accuracy of results.

MNIST digit	3	5	6	7	2	1	1	9	7	4
Difference	9	13	16	14	6	43	39	9	15	10

Table 1: Differences in hidden unit activation between Matlab and BRIAN

5 Conclusions and future work

5.1 Conclusions

Matlab results show the viability of the neuromorphic related modifications performed for RBM implementation listed in Section 3. Using neural sampling with CD-2 instead of Gibbs sampling approaches the quality of results obtained with the standard machine learning implementation. These results also demonstrate that Contrastive Divergence, which was originally developed in the Machine Learning Gibbs sampling framework, gives satisfactory results for neural sampling. Similarly, changes made for the weight updates in the neuromorphic framework to reduce state storage do not seem to have significant impact on performance. While using the discriminative version of the machine allows a more "online" implementation, it seems that substituting the softmax sampling with its binary counterpart reduces the accuracy to ~85%. The reason for this is that the labels now reside in a 4 instead of a 10-dimensional space, affecting separability. Using more steps of CD may lead to some improvement in results, but a better approach may be to use binary coding (e.g., repetitive coding) for the labels using available free neurons from the neuromorphic platform. This will enable maintaining the binary sampling scheme, which reduces hardware complexity, while possibly improving accuracy of classification. BRIAN results show that the continuous time version of this sampling can be viable for RBMs and therefore can be used for machine learning tasks like classification.

5.2 Future work

It has been shown for Machine Learning that error rates for classification tasks can be further reduced by stacking RBMs and doing learning at each pair of layers separately [8]. These machines are called Deep Belief Networks and, based on the neural sampling results obtained for RBM, it may be possible to use a similar approach to implement DBNs in a neuromorphic framework. Further, since order of sampling is not required in the case of neural sampling, this algorithm can also be used to implement BMs where some degree of lateral connectivity is allowed (semi-restricted BM [9]).

Further neuromorphic enhancements in this framework 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 with synaptic differential equations using STDP (spike-timing dependent plasticity). As seen in Figure 11, over 90% of the 392,000 weights in the developed RBM possess values between -0.25 and 0.25 and, therefore, this information should be taken into account when performing the fixed bit-width and sparsification adaptations to the weights.

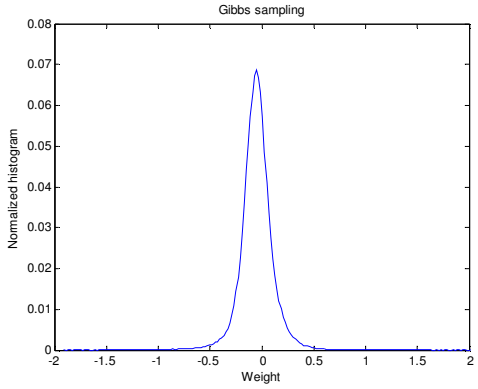

Figure 11: Normalized histogram of the 392,000 weights

Acknowledgments

We would like to acknowledge the help and incentive offered during this work by our colleagues Emre Neftci and Tommy Sprague.

References

[1] Hinton, G.E., Sejnowski, T.J., and Ackley, D.H., "Boltzmann Machines: Constraint Satisfaction Networks that Learn", *Technical Report CMU-CS-84-119*, May 1984.

[2] Mead, C., "Analog VLSI and Neural Systems". Reading, MA: Addison-Wesley, 1989.

[3] Aarts, E.H.L. and Korst, J.H.M., "Boltzmann Machines and Their Applications", *PARLE Parallel Architectures and Languages*, Europe, 1987.

[4] Ackley, D.H., Hinton, G.E., and Sejnowski, T.J., "A Learning Algorithm for Boltzmann Machines". *Cognitive 9*, pp. 147-169, 1985.

[5] Smolensky, P., "Information processing in dynamical systems: Foundations of harmony theory". *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations. Cambridge, MA: MIT Press/Bradford Books, pp. 194-281, 1986.

[6] Hinton, G.E., "Training products of experts by minimizing contrastive divergence". *Neural Computation*, 14(8), pp. 1771-1800, August 2002.

[7] Buesing, L., Bill, J., Nessler, B., and Maass, W., "Neural Dynamics as Sampling: A Model for Stochastic Computation in Recurrent Networks of Spiking Neurons", *PLOS Computational Biology*, Volume 7, Issue 11, November 2011.

[8] Hinton, G.E., Osindero, S., and Teh, Y.W., "A fast learning algorithm for deep belief nets", *Neural Computation*, 18:1527-1554, 2006.

[9] Osindero, S. and Hinton, G.E., "Modeling image patches with a directed hierarchy of Markov random fields", *NIPS 20*, Cambridge, MA, 2008.