Seminar Report on Stochastic Computational Deep Belief Network

Mohammed Muddasser

Chair for Hardware Oriented Computer Science

Abstract. Deep belief network (DBN) is a type of deep neural network (DNN). The seminar topic discusses the hardware realisation of a re-configurable DBN capable of online learning. It summarises the implementations and outlines the results of the technical paper referred. First phase of DBN training employs a fast-greedy algorithm which learns in an unsupervised way. It is then fine-tuned using adaptive moment estimation (ADAM), an supervised learning technique for faster convergence in the second phase. These algorithms are implemented in re-configurable structures using stochastic computation (SC) circuits. SC circuits have simple hardware realisations for many arithmetic operations. The various activation functions applied at each of the DBN layers are implemented using an approximate SC activation unit (A-SCAU). The A-SCAU implementation is designed invariant to SC correlations in the signals allowing for one circuit to be shared among all neurons. The model is evaluated on MNIST dataset for handwritten digit classification. It achieves similar and improved accuracy when compared to most other DNNs, while having a reduced chip area and significantly low power consumption.

1 Introduction

DNNs are powerful paradigms for detection and pattern recognition. A DNN consists of multiple layers of interconnected nodes called neurons. The output of each layer is linear function of the input defined by the weights of the interconnections, which is then fed to an activation function to learn a non-linearly separable relationship. Novel DNN architectures exist for various applications. But, efficient hardware realisations of these DNNs is of keen importance in the embedded and edge applications. The seminar discussion is primarily based on the reference paper [1]. Usually the DNNs in hardware are only designed for inference. But here, the paper implements a re-configurable SC-DBN with online learning capabilities. It has efficient power consumption and reduced chip area while having good accuracy when compared to other DNN models. These excellent specifications make them highly desirable in various resource-constrained systems. The following section discusses SC. The next section includes the training mechanism and hardware implementation of a SC-DBN divided into two subsections of unsupervised and supervised learning phases. The evaluation results are then discussed to end with conclusive remarks.

2 SC logic

A stochastic number A in the range [0,1] is represented as a Bernoulli bit stream $A = (A_1, A_2, ..., A_n)$ such that $P(A_i = 1) = a/n = A$ where n is the total length and a total number of 1's in the bit stream. For example $A = 6/8$ can be represented by a sequence (1,0,1,1,1,0,1,1) and is called unipolar notation of a number. In bipolar notation the range is [-1,1], where $A = (2a - n)/n$ [2]. The complex binary arithmetic computations now reduce to simple operations when implemented in SC circuits [3]. The SC multiplier is shown in the Fig. 1(a) which can be computed using a simple XNOR operation. Fig. 1(b) shows an SC adder implemented using a multiplexer (MUX) and $P(Sel) = 0.5$ [2]. Approximate circuits are used to implement the divider and square root accelerated with binary search as shown in the figures 1(c) and 1(d) respectively [2]. Here, INT refers to an integrator circuit that accumulates the input over every clock pulse and DFF is a D-flipflop to store the previous bit value. The circuits can be implemented with the current CMOS technology. The higher computational accuracy in SC demands longer stochastic sequences (SS) [3] increasing the overall latency. But, the probabilistic nature of the DBN with a greedy training approach make them highly suitable to be implemented with shorter sequence length (SL) SC.

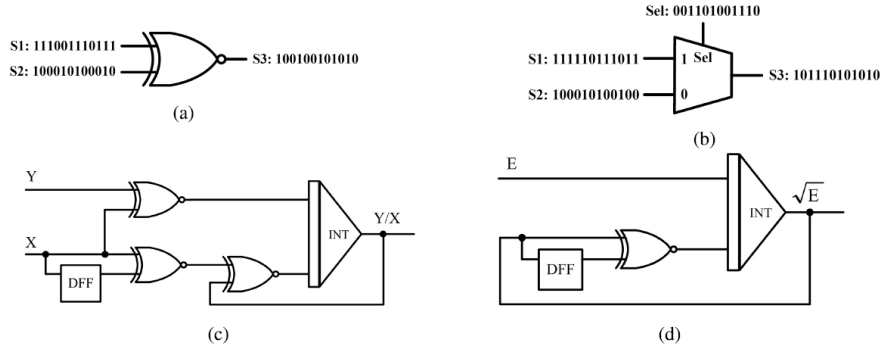


Fig. 1: (a) SC multiplier [1]. (b) SC adder [1]. (a) SC divider [1]. (a) SC square root [1].

3 DBN Working Principle

A DBN has an input layer with D neurons, L hidden layers each consisting of E_i neurons ($i = 1, 2, ..., L$) and one output layer with K neurons as shown in the Fig. 2.

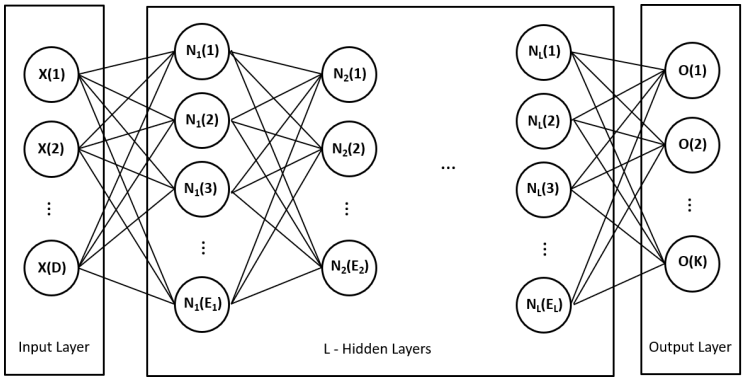


Fig. 2: A Deep Belief Network

3.1 Unsupervised Learning Phase

Fast greedy learning algorithm: The fast greedy algorithm is an unsupervised learning technique to learn the underlying structure of the dataset rather than just learning an input-output relationship. Here, every consecutive pair of layers of the DBN form a unique encoder-decoder pair called an autoencoder, which are trained as a Restricted Boltzmann Machine (RBM) [4]. An RBM is a deep generative learning model trained using contrastive divergence learning (CDL). CDL in principle trains by reducing the reconstruction error of the RBM autoencoder, which simplifies to Gibbs sampling [4]. The Gibbs sampling involves iterating over the one time Gibbs sampling (OTGS) given by (1a) [4]. Here, W is the weight matrix (weights of all connections between two layers), t the time step, ϵ and μ are learning rates, δ_P and δ_N are called the positive and negative difference of the OTGS. The OTGS is computed for all samples in an training epoch and is repeated until the difference $\delta_P - \delta_N$ reaches a certain minimum value [4] and then the RBM is said to be in equilibrium.

$$W(t) = \mu W(t-1) + \epsilon(\delta_P - \delta_N) \tag{1a}$$

$$\begin{cases} Y_E = \phi(X \cdot W) \implies y_i^e = \phi(\sum_{j=1}^{D} x_j.w_{ij}), \quad i = 1, ..., E; j = 1, ..., D & (1b) \\ \delta_P = X^T Y_E & (1c) \end{cases}$$

$$\begin{cases} Y_D = \phi(X \cdot W^T) \implies y_i^d = \phi(\sum_{j=1}^{D} y_j^e.w_{ij}^T), \quad i = 1, ..., D; j = 1, ..., E & (1d) \end{cases}$$

$$\begin{cases} Y_{E_2} = \phi(Y_D \cdot W) \implies y_k^{e_2} = \phi(\sum_{j=1}^{D} y_j^d.w_{kj}), \ k = 1, ..., E; j = 1, ..., D & (1e) \\ \delta_N = Y_D^T Y_{E_2} & (1f) \end{cases}$$

Equations (1a) - (1f) are referenced from [1]. The input layer (D dimensional) and the first hidden layer (E dimensional) form the first RBM auto-encoder pair. The computation of δ_P and δ_N is as follows. The first step is **Encode: X** $\rightarrow$ $\mathbf{Y}_E$. The input row vector X with elements x_j is encoded into Y_E based on (1b), where ϕ is an activation function of choice and y_i^e is the computation for each element of Y_E. w_{ij} is the weight of each connection. The positive difference, a correlation matrix δ_P is computed using (1c). The next step is reconstruction **Decode:** Y_E $\rightarrow$ $\mathbf{Y}_D$ based on (1c) where W^T is transpose of W. $\mathbf{Y_D}$ is not exactly equal to **X** due to the information loss in the encoding/decoding process. The entire training process for the RBM pair revolves around minimising the difference between Y_D and X. For which, the decoded Y_D is encoded again **Encode:** $Y_D \rightarrow \mathbf{Y}_{E_2}$ based on (1e) to compute the negative difference δ_N of the OTGS based on (1f). Once a pair achieves RBM equilibrium, then the weights between these layers are now frozen with respect to the training of higher layers. A new RBM is formed with the next two layers stacked in the network and trained. The step of freezing the trained weights of layer ensures to eliminate the 'explaining away' effect occurring at a particular layer due to conditional dependency between otherwise independent random variables at the input [4]. It prevents getting trapped in a local minima and stepping towards a global optimum. The process continues until the entire network is trained. The fast greedy algorithm can also be fine tuned by using the up-down algorithm [4].

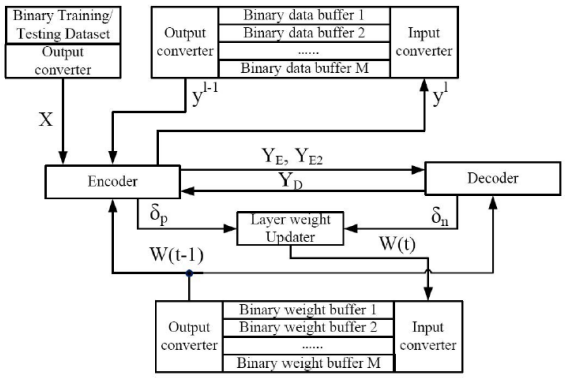


Fig. 3: A Deep Belief Network circuit block

Fig. 3 shows a SC-DBN block. The layer weight updater computes the one time Gibbs sampling given by (1a) and is built using SC circuits discussed in 2. The input and output converters are Stochastic number generators (SNGs) using Random number generators (RNGs) and linear shift registers to convert the data from binary to SS and vice versa [2]. A batch training approach is employed while training each RBM. Here a set of samples are processed in each epoch and the

intermediate results are stored in the data buffers while the computed weights are stored in the weight buffers. The RBM's are trained one layer at a time which allows us to reuse a single SC-DBN block for each layer. The inference too is sequential where the SC-DBN block is shared by each layer and now the data buffers are used to store these intermediate signal values. The encoder-decoder design is shown in the Fig. 4 and 5, which computes the equations (1b), (1c), (1d), (1e), and (1f).

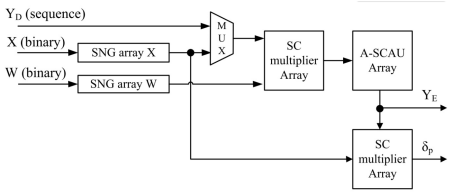


Fig. 4: SC Encoder [1]

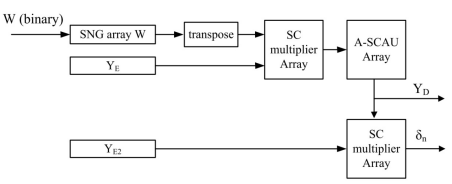


Fig. 5: SC Decoder [1]

Activation function and A-SCAU: The widely used activation functions in a DBN are sigmoid function: $\phi(x) = \frac{1}{2}(tanh(\frac{x}{2})+1)$, rectifier linear unit (ReLU): $\phi(x) = min(1, max(0, x))$ and pure line function: $\phi(x) = min(1, max(-1, x))$. ReLU eliminates the random fluctuations due to Gibbs sampling and hence is widely preferred [5]. The A-SCAU implements these using approximation functions, based on the equation (2a), where p, r and s are variable parameters. The optimal approximated parameter values for sigmoid are $p = 0$, $r = 4$, $s = \frac{1}{2}$, for ReLU $p = 0$, $r = 1$, $s = 0$ and for pure line function: $p = -1$, $r = 1$, $s = 0$.

$$\Psi(x) = min(1, max(p, x/r + s)) \tag{2a}$$

$$x = \sum_{n=1}^{D} k_i = \sum_{j=1}^{n} \frac{c_i}{n} - D \tag{2b}$$

$$\Psi(x) = min(2^m - 1, max(2^{m-1} \cdot (p+1), \frac{2^{m-1}}{nr} \cdot T)) \tag{2c}$$

An A-SCAU circuit is shown in the Fig. 6 (a). The D dimensional stochastic input is from the SC multiplier of the encoder/decoder. The APC is accumulative parallel counter that sums all bits at its input. With a parallelisation of q the APC adds $q \cdot D$ bits at its input to produce a m bit binary output. The Linear Approximation Unit (LAU) accumulates the value for n cycles (where, n is the SL) as per (2b) to compute the summations of the equations (1a), (1c) and (1d). The RNG's in the SNG are prone to correlation issues between the consecutive SS generated, inhibiting SC resource sharing. As the summation is over full input sequence as per (2b), the A-SCAU is invariant to any correlation

allowing it to be shared by all neurons in the same training step. This results in only three RNGs used in complete implementation. The LAU implements an approximate activation equation (2c) obtained by substituting (2b) in (2a). The parameters of (2c) are matched with optimal p, r and s values. The A-SCAU can be implemented completely using the basic SC circuits from 2. The details of the derivation of LAU equation. (2c), the LAU algorithm and it's SC implementation can be found [1]. The activation function is applied to previous summation computed and generates a SS output using the RNG and comparator.

3.2 Supervised Learning Phase

The DBN is now still 'underfit' to perform on the actual data. The supervised phase now learns the input-output relationship. Back propagation is a technique used to compute the gradients with respect to the DNN parameters. The gradients are derivatives of the activation functions which are simple operations implemented in SC. Vanishing gradient leads to slow learning when the gradients calculated are too small and usually occur due to incorrect weight initialisation. But here the unsupervised learning initialises the DBN with optimal weights and hence eliminates the issue. The gradient circuit design can be found in [6]. It is used in the ADAM algorithm which iterates continuously until a minima is reached. The ADAM algorithm is described by the following pseudo-code: (from [1])

$$
\begin{cases}
t = t + 1, & \text{(3a)} \\
g_t = \nabla_\theta f_t(\theta_{t-1}), & \text{(3b)} \\
m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, & \text{(3c)} \\
v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, & \text{(3d)} \\
\hat{m}_t = m_t / (1-\beta_1^t), & \text{(3e)} \\
\hat{v}_t = v_t / (1-\beta_2^t), & \text{(3f)} \\
\theta_t = \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) & \text{(3g)}
\end{cases}
$$

Here, α is the learning rate. β_1 and β_2 are exponential decay rates in the range [0,1). g_t the gradient at time step t. The learning rate is adapted for each parameter based on the running average of the magnitudes of previous first and second order gradients called moments. These first ($\hat{m}_\mathrm{t}$) and second ($\hat{v}_\mathrm{t}$) order moments are used to update the parameters θ_t at every time step. These eliminate the problems of ill conditioning in dataset and result in faster convergence [7]. The typical values are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ based on [7]. The (3b) and (3c) are implemented by the the SC circuit 6 (b). β_1^t and β_2^t are computed by 6, where, K-bit shift register stores the previous value β (c). Circuit 6 (d) computes (3g), where, $\alpha = (1-p)$ and $\epsilon = p\tau$. All the blocks are built using the basic SC circuits discussed in section 2.

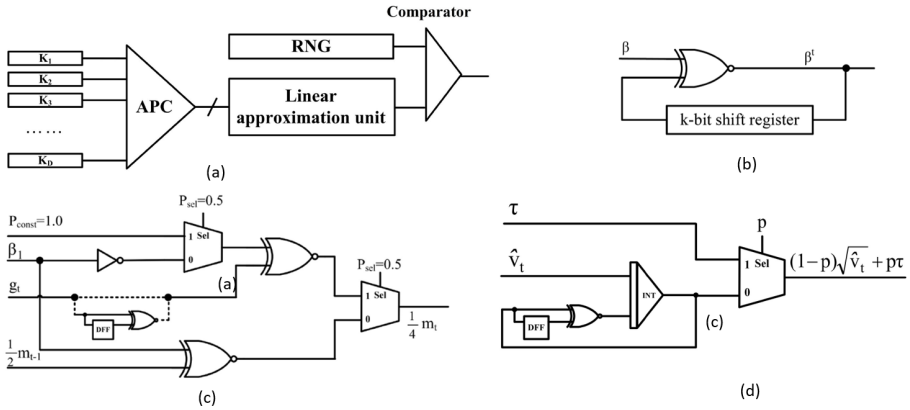


Fig. 6: (a) A-SCAU circuit [1]. (b) Computes equations (3b) and (3c) [1]. (c) Computes β_1^t and β_2^t [1]. (d) Computes equation (3g).

4 Evaluation

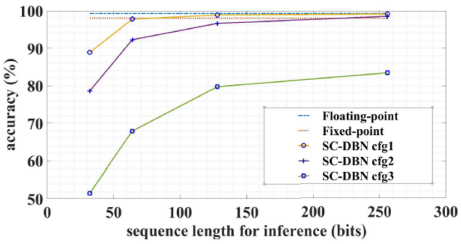
The SC-DBN is used to classify MNIST dataset consisting of images, 28x28 pixels of handwritten digits from 0 to 9. The DBN has input layer of 784 (28x28) neurons, 2 hidden layers of 100 and 200 neurons and an output layer of 10 neurons. A 256 bit sequence with 16x parallelisation produces a total SL of 4096 improving the performance. Based on Fig. 7(a) the SC-DBN has better accuracy than most other DNN models except for its own 32 bit floating point (FP) implementation, with respect to which it has 0.12% to 0.37% lower accuracy. Fig. 7(b) shows that accuracy increases as the SL increases reaching a maximal of 99.15%, with SL of 256 with 16x. A VHDL DBN ASIC is implemented with ST-28nm technology library and Synopsys Design Suite. First an inference only SC-DBN model has 1.3 times higher energy consumption and 21 times higher latency in comparison to 32 bit non-pipelined FP DBNbased on Fig. 7(c). It is due to the SC. With online learning there is a gain in chip area and reduced latency per epoch Fig. 7(d). But, the overall latency is still higher compared to the 8 bit implementation. Inclusion of ADAM circuit with metrics as in Fig 7(e) results is faster convergence, with the number of epochs reduced from 200 to 31, hence reducing the overall latency and power consumption.

5 Conclusion

The referenced paper implements a SC-DBN which is re-configurable and hence having online learning capability. It has the benefits of unsupervised learning, but free of explaining away effect and vanishing gradients. The implementation uses one unsupervised and one supervised circuit block shared by all neurons sequentially in a training step. A single A-SCAU implements three different

Network	sequence length (bit)	accuracy (%)
SC-DBN (16× parallelization)	32	89.90
	64	97.78
	128	**98.90**
	256	**99.15**
8-bit fixed point	–	98.10
32-bit floating-point	–	99.27
Integral stochastic NN [19]	64	97.73
Hybrid SC-binary NN [30]	128	99.01
SC DNN [18]	1024	97.59
FPGA-RBM [29]	1024	94.28
FPGA-DBN [17]	4096	94.10

(a)



(b)

	SC-DBN	8-bit fixed-point circuit	32-bit floating-point circuit (pipelined, non-pipelined)
Area (μm^2)	23345	86875	(437767, 357548)
Power (mW)	1.12	4.01	(24.86, 18.32)
Frequency (MHz)	134.7	167.3	(159.7, 90.2)
Cycle (/sample)	128/256	296	(412, 4)
Latency (μs/sample)	0.94/1.90	1.77	(2.580, 0.044)
Energy (nJ/sample)	1.05/2.12	7.10	(64.14, 0.81)

(c)

	SC-DBN	8-bit fixed-point circuit	32-bit floating-point circuit
Back-propagation circuit area (μm^2)	33150	116413	656651
Learning control unit area (μm^2)	3525	1785	1829
Total area (μm^2)	60019	205072	1096247
Latency per epoch (μs)	7.60	6.92	9.43
Total latency (200 epochs) (ms)	1.52	1.38	1.88
Energy per sample (μJ)	4.37	13.11	117.40

(d)

ADAM area (μm^2)	27181
Total area (μm^2)	87200
Total latency (31 epochs) (μs)	529.1
Energy per sample (μJ)	1.10

(e)

Fig. 7: (a) Accuracy for pre-trained DNNs [1]. (b) Accuracy vs SL relationship [1]. (c) Hardware metrics of SC-DBN with inference only [1]. (d) Hardware metrics for online learning SC-DBN [1]. (e) ADAM circuit metrics [1].

activation functions and is free of signal correlations allowing to be shared by all neurons during training and inference. The long latency issues due to SC and sequential use of shared circuits is considerably reduced with the use of ADAM. It results in a SC-DBN with good accuracy, reduced chip area and significantly low power consumption as compared to the corresponding binary implementations.

References

1. Liu, Y., Wang, Y., Lombardi, F., Han, J.: An energy-efficient online-learning stochastic computational deep belief network. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **8**(3) (2018) 454–465
2. Alaghi, A., Hayes, J.P.: Survey of stochastic computing. ACM Trans. Embed. Comput. Syst. **12**(2s) (2013)
3. Gaines, B.R. In: Stochastic Computing Systems. Springer US, Boston, MA (1969) 37–172
4. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation **18**(7) (2006) 1527–1554 PMID: 16764513.
5. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proc. 14th Int. Conf. Artif. Intell. Stat. (2011) 315–323
6. Liu, Y., Liu, S., Wang, Y., Lombardi, F., Han, J.: A stochastic computational multilayer perceptron with backward propagation. IEEE Transactions on Computers **67**(9) (2018) 1273–1286
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014)