Architectural Exploration for On-chip, Online Learning in Spiking Neural Networks

Subhrajit Roy, Sougata Kumar Kar and Arindam Basu

School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798
Email: subhrajit.roy@ntu.edu.sg, arindam.basu@ntu.edu.sg,

Abstract—**In the recent past there has been an increasing demand for area and energy efficient on-chip implementation of Machine Learning techniques. In this context we have proposed area and power optimized architectures for hardware implementation of a recently proposed supervised learning technique named Network Rewiring (NRW) for Dendritically Enhanced Readout(DER). We show that for the most optimized architecture there is a 8.5 times reduction in critical resources while the MAE has increased only by 1.76% compared to the non-optimized architecture. Moreover, for accommodating real-time training, we have also proposed an online version of the NRW rule. We also show that, though this online algorithm uses an averaging circuit having 4200 times lesser time constant compared to batch learning, yet it provides comparable performance due to the introduction of a voting mechanism.**

I. Introduction: Need for On-chip, Online Learning

It is a long standing debate in Machine Learning domain [1] on whether to train a network by batch or online learning. Traditionally online learning is chosen over batch learning when faster training of the network is required. But, we have slightly different motivations for considering online learning. Firstly, as we are considering on-chip learning, so to do batch learning of the neural network implemented in the chip the entire data set has to be stored in the chip itself. This requires a lot of storage space. On the other hand if online learning is done, storage of 'mini-batches' of the entire data set will suffice. Secondly, batch leaning requires averaging over the entire data set to find the 'true gradient', but as the hardware implementation of the neural network is done by analog circuits so averaging over the entire data set is not possible due to time constant limitations. In that case, the 'true gradient' needs to be found on the part of the data set which the averaging circuit is able to average on. Thirdly, as these neural network chips will be used as embedded sensors in real world scenarios where the input data distribution is changing over time online learning, unlike batch learning, is able to track the changes in the data and provide good approximation results.

In this article, we will focus on developing low-overhead architectures for on-chip implementation of a recently proposed learning algorithm named Network Rewiring (NRW) learning rule for Dendritically Enhanced Readout(DER) [2] of spiking neural networks. This can be used for classifying spikes from a brain-machine interface or bio-inspired sensors [3] and has the advantage of using hardware friendly binary synapses. Next, we will also propose an online variant of the NRW rule for real-time training .

II. Background and Theory

In [2], DER has been used as the readout of Liquid State Machine [4] known as Liquid State Machine with Dendritically Enhanced Readout (LSM-DER). In LSM-DER, the DER stage is responsible for extracting information from the liquid output and is trained using the NRW algorithm described in detail in [2] to perform binary classification and regression. Here, we present an overview of the same for the sake of completeness.

The DER architecture is shown in Fig. 1. It employs two nonlinear neuronal cells (NL-cell), each of which has m identical dendritic branches connected to it with each branch having k excitatory synapses. If $\mathbf{x}$ is an input vector to this system, then each synapse is excited by any one of the d dimensions of the input vector where $d >> k$. The output response of j^{th} dendritic branch is calculated as a nonlinear weighted sum of the k synaptic points connected to the branch and is given by $z_j = b(\sum_{i=1}^{k} w_{ij}x_{ij})$, where $b()$ is a model of the dendritic nonlinearity (b(x)=x^2/x_{thr} for x< x_{sat} in this work as in [2]), w_{ij} is the synaptic weight of the i^{th} synapse on the j^{th} the branch and x_{ij} the input arriving at that particular synaptic connection. Combining all the dendritic responses, the overall output $f(\mathbf{x})$ of the neuronal cell is given by:

$$f(\mathbf{x}) = \sum_{j=1}^{m} z_j = \sum_{j=1}^{m} b(\sum_{i=1}^{k} w_{ij}x_{ij}). \tag{1}$$

where $f()$ denotes the neuronal current-frequency conversion function. The output of DER was calculated by noting the difference of the output of the two NL-cells. The overall output of the circuit is given by

$$y = g[f_+(\mathbf{x}) - f_-(\mathbf{x})] \tag{2}$$

where the function g depends on the task. For example, it is a heaviside function for classification.

The NRW algorithm used to train DER primarily involves the following steps:

1) For connecting the input to the neuronal cells binary synapses have been used (w_{ij}=0 or 1), thus instead of

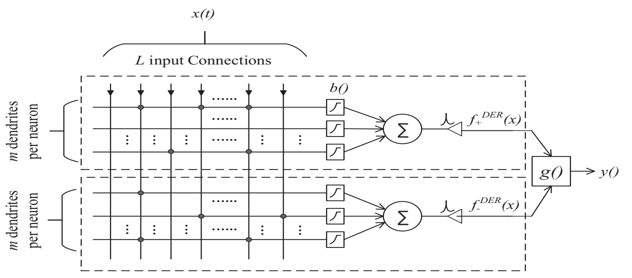


Fig. 1: Architecture of DER network.

weight changes we do connection changes. Since $d >> k$, the learning chooses the best k choices of connection for each branch.

2) At every step, a random set T of n_T synapses are selected for probable replacement. The performance index c_{ij} is calculated for each synapse in the set n_T as $c_{ij} =< x_{ij}b'_j(t-y) >$ for the positive cell and $c_{ij} = -< x_{ij}b'_j(t-y) >$ for the negative cell. Here, $<>$ indicate averaging over the training set. The synapse with the lowest value of c_{ij} in T is labeled for replacement.
3) The replacement synapse is chosen from a candidate set R having n_R of the d input lines. The set R is created by placing n_R 'silent' synapses from d input lines on the branch with the lowest c_{ij} synapse.

In this article, the values of m, k, n_T, n_R, x_{thr}, max_{iter} and x_{sat} were kept to be 7, 10, 7, 7, 1.8, 1000 and 75 respectively for the simulations, unless mentioned otherwise.

III. Architectural Explorations and Results

In this section, we first present architectures and modified NRW learning rules with less hardware requirement (in particular less number of c_{ij} calculations) followed by an online version of the NRW rule.

A. Architectural modification of DER for on-chip learning

a) Architecture I: First, let us consider the hardware requirements for on-chip learning using the architecture described in Section II. Since we randomly select any of the $m \times k$ number of synapses to be in T and use the corresponding c_{ij}, we need to have provision in hardware for calculating c_{ij} for all the synapses in the positive and the negative cells. After labelling the minimum synapse for replacement (using loser-take-all (LTA) blocks for comparing c_{ij}), the replacement set R has to be formed by placing nR 'silent' synapses on the branch with the minimum c_{ij} synapse. Since we do not know apriori which dendrite will have the minimum c_{ij} synapse, each branch has to be equipped with nR silent synapses with their own c_{ij} calculator and a winner-take-all (WTA) is used to find the one with maximum c_{ij}. Thus, the number of c_{ij} calculators required are $N_c = (k \times m + nR \times m)$ per cell.

b) Architecture II: As a first step to reduce the number of c_{ij} calculators, we note that only n_T of the $m \times k$ c_{ij} are used in any step. Hence, to form the target set T, we create a set of 'Copy synapses' denoted by C per cell by keeping one new synapse equipped with a c_{ij} calculator on each dendrite. It is formed by randomly taking one synapse from each dendrite of the cell. Thus, $n_T = m$ in this case. Even though we marginally increase the number of synapse circuits, we drastically reduce number of c_{ij} calculators. The input to the copy synapse on one branch (entries in an off-chip SRAM storing AER connections [2]) is same as one of the existing synapse inputs on that branch. The formation of the silent synapse set is same as the previous architecture. Thus, the number of c_{ij} calculators required per cell is $N_c = (m + nR \times m)$. The architecture of the positive cell is shown in Fig. 2(a).

TABLE I: Comparison of DER architectures

Arch.	N_c	MAE	Remarks
I	$k \times m + nR \times m$ (119)	0.0912	Least MAE=MAE_{min}, Highest $N_c = N_{c,max}$
II	$m + nR \times m$ (56)	0.0946	MAE≈ MAE_{min}, $N_c < N_{c,max}$
III	$m + nR$ (14)	0.1088	MAE= $1.1930 MAE_{min}$, $N_c << N_{c,max}$
IV	$k + nR$ (17)	0.1094	MAE= $1.1996 MAE_{min}$, $N_c << N_{c,max}$

c) Architecture III: As an immediate next step we try to reduce the number of c_{ij} calculators utilized by the silent synapses. The reason behind the usage of $nR \times m$ c_{ij} calculators per cell was that we did not know beforehand on which dendrite the nR silent synapses were to be placed. Two methods can be employed to circumvent this problem. In Method I, instead of keeping the silent synapses in the dendrite containing the minimum c_{ij} synapse for the current epoch (presentation of all the input patterns), we place them in the dendrite containing the minimum c_{ij} synapse for the previous epoch. This technique ensures that we have prior knowledge of where to place the silent synapses and so we would only need nR number of silent synapses per cell each having a c_{ij} calculator. Thus, now we only need $N_c = m + nR$ number of c_{ij} calculator circuits per cell. This architecture of the positive cell is portrayed in Fig. 2(b).

d) Architecture IV: In Method II for reduction of silent synapse calculators, we place the silent synapses randomly on any one of the dendrites. In this case, the target set formation technique used for Architecture II cannot be used. The Target Set T in this method comprises of all the synapses present in the randomly chosen dendrite for silent synapse placement making $n_T = k$. So, for each cell we require k and nR c_{ij} calculators for the copy synapse and the silent synapse set respectively.

Software simulations have been done for each of the above four architectures for the Spike Train Classification task [2], [4] and the average Mean Absolute Error (MAE) for 10 trials has been provided in Table I. From Table I, it is evident that Architecture III gives a good trade off between performance and hardware overhead by using 8.5 times less c_{ij} calculators than Architecture I while hurting performance by only 1.76%.

Table I summarizes the key points of the above four architectures.

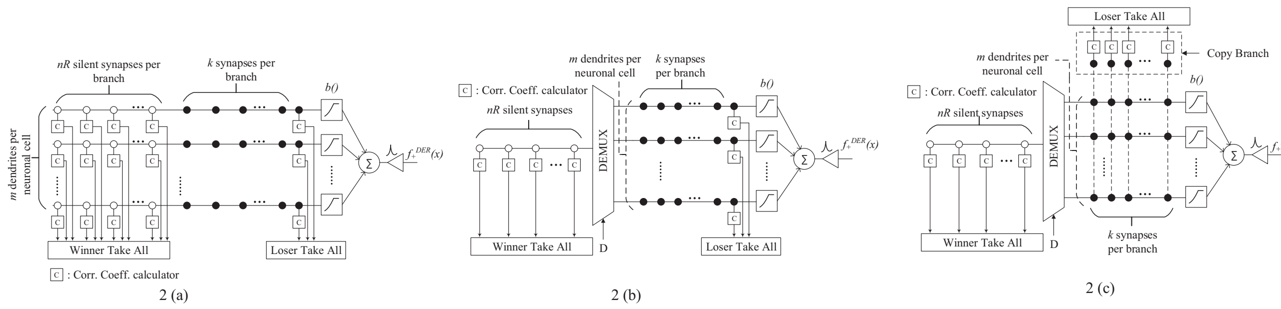


Fig. 2: This figure shows three different architectures described in this article for on-chip implementation of DER

B. Online NRW learning rule

Here, we present a modified NRW learning rule suitable for online learning of P patterns. One practical constraint when the c_{ij} calculator is implemented using analog circuits is that the time duration t_{sp} over which the averaging circuit operates is of the order of milliseconds, which may even be smaller than the duration T_p of a single pattern encountered in real world pattern sets. Hence, we can never find maximum or minimum of $c_{ij} = sgn(t-y)\int_0^{PT_p}(x_{ij}b'_j)$ Our proposed rule takes into account such cases as well.

1) After presentation of P_{sub} number of patterns of the entire data set P, a set L_{sub} is formed by choosing the input dimensions where at least one spike has occurred. The initial connection of both the cells of DER are initialized from the set L_{sub}.
2) The Target Set T and Replacement Set R is initialized from the set L_{sub}.
3) A sub-Target Set T_{sub} and a sub-Replacement Set R_{sub} (which are subsets of T and R respectively) is formed for each cell every t_{sp} seconds. T_{sub} and R_{sub} comprises of the entries of T and R respectively which received a non-zero inputs in the last t_{sp} seconds.
4) The averaging circuit finds $c'_{ij} = sgn(t-y)\int_0^{t_{sp}}(x_{ij}b'_j)$ for the synapses in T_{sub} and R_{sub}–an estimate of the true c_{ij} used in the earlier algorithm. The WTA and LTA circuits calculate the minimum c_{ij} synapse from T_{sub} and maximum c_{ij} synapse from R_{sub} respectively. This step has a loss in information since we no longer have information to get the true c_{ij}.
5) To make up for information lost in the earlier step, we introduce a voting mechanism to find most probable maxima and minima. After presentation of P_{conn} number of patterns, the synapse that has been selected as the minima for the most number of times is tagged as the candidate for replacement. This synapse is replaced by the silent synapse which had maximum c_{ij} value for most number of times. Thus the network rewiring learning takes place after P_{conn} number of input patterns have been presented. The MAE is checked for the next P_{MAE} patterns and if the average P_{MAE} does not decrease, the connections are switched back.
6) Steps 3-6 are repeated max_{iter} number of times after which the algorithm is terminated and the connection corresponding to the best performance is saved as the final connection.

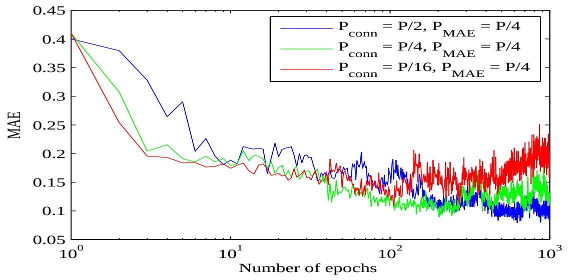
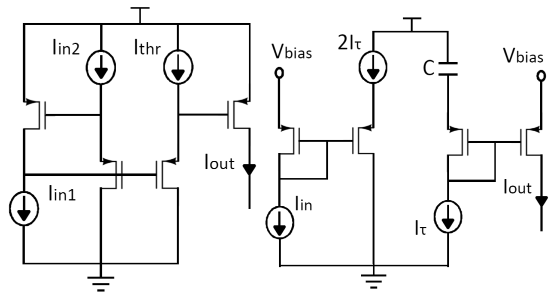


Fig. 3: Setting $P_{MAE} = \frac{P}{4}$, the convergence curves have been shown for different values of P_{conn}.

In the extreme case when $P_{conn} = P$, the connection changes are made after the presentation of the entire batch. When P_{conn} is large, we have more information available to make better decisions about the replacement leading to more stable convergence of the algorithm. However, the connection changes in this case are less frequent and so the training becomes slow. On the other hand, when P_{conn} is small the frequency of connection changes increase and so the training is fast at the cost of convergence issues. This trend can be observed in Fig. 3 where the convergence curves of the learning rule, averaged over 10 trials, has been shown for different values of P_{conn} with fixed P_{MAE}. We have also obtained the MAE for different values of P_{MAE} keeping the value of P_{conn} constant at $\frac{P}{4}$. The average MAE over 10 trials when P_{MAE} has been kept as $\frac{P}{8}$, $\frac{P}{4}$ and $\frac{P}{2}$ are 0.1317, 0.1127 and 0.1098 respectively. These results show the characteristic tradeoffs involved in online learning. When $P_{conn} = P$ and $P_{MAE} = P$, the MAE obtained is 0.1075 i.e. we have achieved similar performance as compared to batch learning, though we have used an averaging circuit with 4200 times lesser time constant. Our voting mechanism has artificially stretched the time constant of the circuit thereby making this possible.

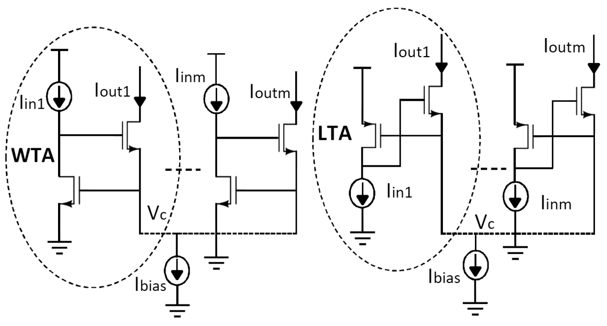
IV. VLSI Implementation of Building Blocks

To implement the different architectures for on-chip learning described earlier, we need to design the different blocks with



(a) (b)

Fig. 4: Schematic of (a) the current mode multiplier and (b) the LPF circuit needed for calculating average c_{ij}.



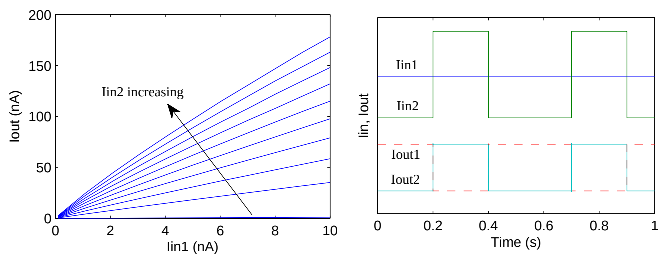
(a) (b)

Fig. 5: Schematic of (a) the WTA cell and (b) the LTA cell needed to identify target and replacement candidate for NRW learning.

a low-power and area overhead. Hence, we chose the sub-threshold analog circuit design method. The main blocks needed in this design are DPI synapse, squaring circuit and c_{ij} calculator. The DPI synapse and the squaring circuit are described in [5]–hence, we focus on the c_{ij} calculator in this paper. The c_{ij} calculator mainly includes a multiplier and a current mode LPF for averaging. The WTA and LTA circuits are also needed for choosing maximum and minimum c_{ij} of target and replacement set respectively. We describe these blocks next.

The schematic of the current mode multiplier circuit is shown in Fig. 4(a). Its output current can be expressed as $I_{out} = x_{ij} * b'_j / I_{thr}$ where x_{ij} is the individual synapse current, b'_j is the respective branch current before squaring and I_{thr} is a normalizing current.

In the c_{ij} calculator, the multiplier output is averaged by a



(a) (b)

Fig. 6: SPICE simulation results of (a) the current mode multiplier and (b) the WTA circuit.

current mode LPF shown in Fig. 4(b). It is desired to have a large averaging time period–but this involves a direct tradeoff with capacitor size. In our case, the $C = 500fF$ and $I_\tau = 10pA$ leading to a time constant of $1.5ms$.

Now the current from the averaging circuit is passed through either WTA or LTA. Every synapse in the copy synapse set has one LTA sub-circuit each and all these sub-circuits in a cell are connected together to form the full LTA. Similarly all the WTA sub-circuits in the silent synapse set for a particular cell are connected together to find out the winner among the silent synapses. WTA and LTA circuit schematics are shown in Fig. 5(a) and (b) respectively. The basic principle of operation is same for both WTA and LTA. For example, in the WTA, the WTA sub-circuit with maximum input steals all of the I_{bias} current becoming the winner.

Simulation result of the multiplier and WTA are done in a 0.35 um CMOS process and are shown in Fig. 6(a) and (b) respectively. The bias current of the WTA, I_{bias}, is $10nA$. The resolution of the WTA is around $20pA$ while the comparison time is approximately $100\mu s$. Figure 6(a) shows single quadrant multiplication while 6(b) shows the sub-circuit with higher current wins.

V. Conclusion

In this article we have looked into various strategies which can be employed while implementation of DER in hardware. We have tried to find an architecture which is able to optimize both area and power, while maintaining acceptable performance. In this search we have devised an optimized architecture, termed in this paper as Architecture III, which uses 8.5 times less resources than the non-optimized architecture (Architecture I) while having a performance degradation of only 1.76%. We have also devised a novel learning rule for DER which is able to train the DER network for online learning. Although this learning rule uses an averaging circuit with 4200 times lesser time constant rather than its batch counterpart, still it provides similar MAE due to our unique voting mechanism. After that we have looked into some basic blocks required for the on-chip implementation of DER. In future we will be doing this on-chip implementation and employ this chip for real time online learning.

References

[1] Yuan Yao, “On complexity issues of online learning algorithms,” *IEEE Transactions on Information Theory*, vol. 56, no. 12, pp. 6470–6481, Dec 2010.
[2] S. Roy, A. Basu, and S. Hussain, “Hardware efficient, Neuromorphic Dendritically Enhanced Readout for Liquid State Machines,” in *IEEE BioCAS*, Nov 2013, pp. 302–305.
[3] V. Chan, S.C. Liu, and A.V. Schaik, “AER EAR: A matched silicon cochlea pair with address event representation interface,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 48 –59, jan. 2007.
[4] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: a new framework for neural computation based on perturbations,” *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.
[5] S. Roy, A. Banerjee, and A. Basu, “Liquid state machine with dendritically enhanced readout for low-power, neuromorphic vlsi implementations,” *IEEE Transactions on Biomedical Circuits and Systems*, 2014 (submitted).