Hardware-Friendly Actor-Critic Reinforcement Learning Through Modulation of Spike-Timing-Dependent Plasticity

Nan Zheng, *Student Member, IEEE* and Pinaki Mazumder, *Fellow, IEEE*

Abstract—In this work, we propose a hardware-friendly reinforcement learning algorithm. The learning algorithm is based on an actor-critic structure implemented with spiking neural networks (SNNs). A biologically plausible and hardware-friendly spike-timing-dependent plasticity learning rule is formulated and employed in the training of SNNs. Several important aspects of applying the learning rule in a reinforcement learning context is studied, especially from the circuit designers' point of view. Pitfalls of potential noise mixing and correlated spikes are identified and properly addressed. To feature a low-power learning architecture, techniques such as down-sampling data for certain learning blocks, injecting quantization noise as noisy residues in neurons, and proper memory partitioning are proposed. A 1-D state-value function learning problem and a 2-D maze walking problem are examined in this paper to illustrate effectiveness of the proposed algorithm and learning rules. A low-power hardware architecture is proposed and examples are implemented with Verilog. Hardware complexity of the proposed algorithm is analyzed, and potential solutions to breaking memory bottleneck when the size of the problem gets large is also discussed.

Index Terms—Reinforcement learning, spiking neural network, hardware neural network, spike-timing-dependent plasticity, and actor-critic network

1 Introduction

Development and implementation of hardware-friendly machine learning algorithms have drawn many researchers' attention in recent years. The main objective is to push the capability of machine learning to the next level by leveraging the computational power of specialized hardware. Compared to a traditional software-only approach, specialized hardware is able to achieve much more computational power while consuming less energy. Available hardware platforms include universal field programmable gate arrays (FPGA) [1], [2], [3], [4], customized CMOS chips [5], [6], [7], [8], [9], [10], [11], [12], emerging nanotechnologies such as memristors [13], [14] and spintronics [15].

Various machine learning algorithms have been implemented on abovementioned hardware platforms, such as deep learning [8], [12], sparse learning [10], and support vector machine [16]. However, as one of the most important branches of machine learning, reinforcement learning is only reported in a limited number of articles [9].

Spiking neural networks (SNNs), as the third generation of neural networks, are promising technologies in helping accomplish numerous machine learning tasks. Traditionally, reinforcement learning is developed on the basis of artificial neural networks (ANNs), where the ANN is used as a universal function approximator. Recently, there are efforts attempting to build reinforcement learning with SNN, as it is hypothesized that this way of learning might actually be the mechanism as to how mammals learn [17], [18], [19], [20], [21]. Besides its biological feasibility, SNN is more hardware-friendly, especially when the size of the network is large. Address-event representation (AER) is able to conveniently interconnect individual sub-SNNs in a large network [5], [11], [22], and the parallel nature of a SNN enables a fast computation by exploiting parallel processing capability of specialized hardware. Furthermore, the event-driven computation provided by a SNN is much more energy-efficient compared to traditional computation. For example, it has been shown in [23] that customized SNN hardware is two orders of magnitude more energy-efficient than the traditional rate-based ANN implemented on FPGA. As a result, the focus of hardware implementation of neural network has been shifted from ANNs to SNNs in recent years. More and more SNNs, such as TrueNorth from IBM [11], CAVIAR in Europe [12], and neuromorphic chips from HRL [24] are built to tackle larger and more complicated tasks.

Spike-timing-dependent plasticity (STDP) existing in SNNs has been long believed to be the underlying mechanism with which the mammal brain learns [25], [26]. To enable SNNs with learning capability, STDP is often employed as a learning algorithm in a hardware SNN. In most cases, STDP is treated as a biologically plausible and empirically successful learning rule without its underlying principle being studied. There are numerous efforts from neuroscience community trying to explain the fundamental role of STDP in learning [27]. Hinton in [28] first hypothesized that STDP-based learning might be a form of gradient descent learning. Later on, similar ideas are shown in different contexts with various forms [17], [18], [19], [20].

• *The authors are with the Department of EECS, University of Michigan, Ann Arbor, MI 48105. E-mail: {zhengn, mazum}@umich.edu.*

Manuscript received 3 Feb. 2016; revised 17 May 2016; accepted 22 June 2016. Date of publication 26 July 2016; date of current version 20 Jan. 2017.
Recommended for acceptance by N. Jha.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2016.2595580

In this paper, the STDP learning rule is developed as a hardware-friendly learning mechanism. The main objective of this paper is not to reveal the exact role of the STDP learning rule in a biological context. Rather, our aim is to adapt the STDP learning rule into an economical way of learning on silicon by making reasonable simplifications and assumptions. It is shown that STDP, with minor adjustments, becomes a measure of gradient. As a result, the STDP learning rule turns out to be a gradient descent optimization method. With the learning rule justified, we focus on using it as an effective tool for actor-critic network based reinforcement learning. Hardware-oriented learning rules are formed, and implementation techniques that help achieve better learning performance are presented. Noise mixing problems are identified when the STDP learning rule is employed in reinforcement learning that tries to minimize temporal-difference (TD) error. High-frequency quantization noises are down-converted to baseband, which will saturate the learning process unless proper filtration is used. In addition, to ensure that the STDP learning rule performs properly, a statistical neuron model is needed such that there is little correlation among spikes. We propose a quantization noise injection technique that avoids using pseudo-random generators, saving power and area. Furthermore, two different sampling rates are proposed to be used in the reinforcement learning task to take advantage of both ease of routing of 1-bit spike signals and power efficiency of multi-bit calculations. In addition, the decimation gain associated with the down-sampling process is also helpful in reducing the number of bits needed for synapse weights in a fixed-point implementation. Two testing cases inspired by [18] are employed to examine all aspects of hardware implementation. A low-power hardware architecture for the algorithm is also proposed, examples of which are implemented in Verilog as synchronous digital circuits. In addition, hardware complexity is analyzed. To mitigate the limits posed by memory access, several possible solutions are also discussed.

2 Background

We first review a few concepts in an actor-critic based reinforcement learning. For brevity, only important terminologies that are closely related to this paper are covered. The interested reader is referred to the classic book on reinforcement learning [29].

In a reinforcement learning task, an agent tries to learn the optimal policies that maximize the rewards it receives from the environment over time. One of the most popular types of reinforcement learning is the actor-critic method. There are two networks in an actor-critic architecture. One is the actor network which is responsible for action selection. It attempts to learn the optimal policy with the help of a critic network. A policy, denoted by $\pi(s,\ a)$, is a function of state s and action a. It represents the probability of action a being selected at state s. The critic network is used to estimate the value of a state s_t under a policy $\pi(s,\ a)$.

$$V\ (s_t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1.} \tag{1}$$

In (1), γ is the discount factor used to determine the relative importance of future rewards. A small γ encounrages the agent to pursue short-term rewards, whereas a large γ promotes long-term rewards. r_t is the reward that the agent receives from the environment at time t. It should be noted that the state-value function $V(s_t)$ shown in (1) is implicitly a function of the current policy $\pi(s,\ a)$.

A common technique employed in learning state-value function is the temporal-difference learning. The TD error is defined as

$$\delta_t = \ \gamma V(s_{t+1}) - V(s_t) + r_{t+1.} \tag{2}$$

TD learning is a smart way of using a new estimation to update old estimations. TD error shown in (2) serves as the error signal in feedback that drives output from critic network to converge to the real state-value function. More specifically, when the TD error for all states is zero, output from the critic network satisfies (1).

Function approximators such as neural networks are often employed as critic and actor. To learn the correct state-value function, gradient-descent algorithm can be used. Suppose we use a function approximator $\phi(\cdot)$ to represent the state-value function such that $V\ (s_t) = \ \phi(\mathbf{w_t})$, where $\mathbf{w_t}$ is a vector of weights. To learn the correct weights, update rules shown in (3) can be utilized.

$$\mathbf{w_{t+1}} = \mathbf{w_t}\ + \eta\delta_t \nabla_{\mathbf{w_t}} V(s_t), \tag{3}$$

where η is the learning rate.

For the actor network, TD error can also be employed for learning purposes. Suppose the actor selects an action a_t in state s_t. If the TD error is positivie, it suggests that the outcome of the previous action is better than estimated, so the tendency of selecting a_t in state s_t should be strengthend, and vice versa.

One important aspect in an actor network design is to balance exploration and exploitation. Exploration is important especially at the early stages of learning. It helps gather more information about the problem. For example, more states in the state space can be reached through enough exploration. Too much exploration, however, results in slow learning. It is expected that exploitation dominates when the agent is well trained. Therefore, balancing between these two strategies is critical. There are two popular ways used in actor network for exploration purposes. One is ε-greedy, and another is softmax. Interested readers are referred to [29] for more information.

3 STDP as a Measure of Gradient

Let us consider two layers of neurons as shown in Fig. 1. Spike trains outputted by the pre-synaptic (input) neurons and post-synaptic (output) neurons are shown in (4) and (5), respectively

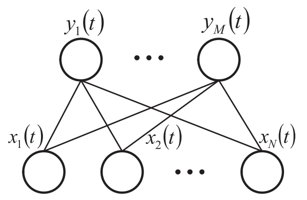


Fig. 1. Illustration of a two-layer neural network. $x_i(t)$ is the pre-synaptic (input) spike train, and $y_j(t)$ is the post-synaptic (output) spike train.

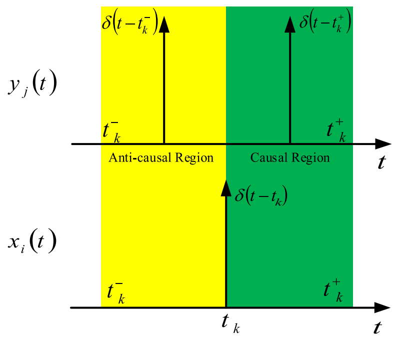


Fig. 2. Illustration of two regions divided by the spike timing of the input neuron. The causal and anti-causal regions are defined according to the causal relationship between input and output spikes.

$$x_i(t) = \sum_{k_i} E(t) * \delta\big(t - t_{k_i}\big) \tag{4}$$

$$y_j(t) = \sum_{k_j} E(t) * \delta\Big(t - t_{k_j}\Big), \tag{5}$$

where the shape of the spike is represented by the excitatory postsynaptic potential $E(t)$.

Setting aside the $E(t)$ term for now and assume spikes are just pulses. This way of representing spikes is the norm for hardware implementation. As illustrated in Fig. 2, spike timing can be used as a boundary to establish two regions: causal and anti-causal for any given input spike. More detailed definition of these two regions may vary with implementations. For example, for discrete-time implementations with Δt as the time resolution, $[-\Delta t, 0)$ and $[0, \Delta t)$ can be defined as the two regions if there is no delay between input neurons and output neurons, whereas $(-\Delta t, 0]$ and $(0, \Delta t]$ can be used for implementations where one unit time delay exists between input and output neurons.

Let us now focus on one pair of input and output neurons that are connected by a synapse. The subscript used to number the neurons is understood and dropped in places where index is not important. (6)-(8) define three quantities that help simplify notations in the following analysis. They are binomial random variables that represent whether certain events occur or not.

$$X_k = \int_{t_k^-}^{t_k^+} x(t)dt \tag{6}$$

$$Y_k^+ = \int_{t_k}^{t_k^+} y(t)dt \tag{7}$$

$$Y_k^- = \int_{t_k^-}^{t_k} y(t)dt\,. \tag{8}$$

The probability that only a causal spike occurs is

$$P\left(Y_k^+ = 1 \cap Y_k^- = 0 \middle| X_k = 1\right) = \frac{P\big(Y_k^+ = 1 \cap Y_k^- = 0 \cap X_k = 1\big)}{P(X_k = 1)}\,. \tag{9}$$

Similarly, the chance of only an anti-causal spike occurs is

$$P\left(Y_k^+ = 0 \cap Y_k^- = 1 \middle| X_k = 1\right) = \frac{P\big(Y_k^+ = 0 \cap Y_k^- = 1 \cap X_k = 1\big)}{P(X_k = 1)}. \tag{10}$$

We are interested in finding out what $\partial\rho(y(t))/\partial\rho(x(t))$ is, where $\rho(\cdot)$ represents the normalized density of a spike train. Let us for now consider the quasi-stationary case. That is, $\rho(x(t))$ and $\rho(y(t))$ have much narrower bandwidths compared to the operating frequency of neurons. We first define a quantity $stdp$ in (11).

$$stdp = \begin{cases} 1 & X_k = 1 \text{ and } Y_k^+ = 1 \text{ and } Y_k^- = 0 \\ -1 & X_k = 1 \text{ and } Y_k^+ = 0 \text{ and } Y_k^- = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Then we have

$$\rho\,(stdp(t)) = P\big(Y_k^+ = 1 \cap Y_k^- = 0 \cap X_k = 1\big) - P\big(Y_k^+ = 0 \cap Y_k^- = 1 \cap X_k = 1\big). \tag{12}$$

Next, we arrive at the first key equation with the help of (9)-(12).

$$\frac{\partial\rho(y(t))}{\partial\rho(x(t))} = E\left(\Delta Y_k \middle| X_k = 1\right) = \frac{\rho(stdp(t))}{\rho(x(t))}\,. \tag{13}$$

Intuitively, (13) indicates that $\partial\rho(y(t))/\partial\rho(x(t))$ can be obtained by observing how the output spike alter its statistical behavior upon an input spike which serves as a small perturbation to the network. One advantage of the pulse implementation (i.e. $E(t) = 1$) is that the current input spike only affects the next output spike. In other words, output spikes that are more than one unit time away from the current spike are uncorrelated. Therefore, causal and anti-causal region need only to span one unit time, leading to a very simple implementation in hardware. In this paper, we only discuss the case where $E(t) = 1$ since this is the common practice in most hardware implementations. In the case where a non-trivial $E(t)$ is used, the two regions need to expand accordingly as the effect of the pre-synaptic spike is distributed over time by $E(t)$. A larger time window is needed in this case to accumulate all effects caused by the pre-synaptic spikes.

Even though (13) is obtained by assuming there is only one pair of input and output neurons, it applies to a network of input and output neurons as shown in Fig. 1, as long as the spike timing of each neuron is reasonably uncorrelated. An intuitive example is that at any given time t_i, as shown in Fig. 2, the probabilities that the output neuron fires in either of the two regions are equal, when the input neuron spike is absent. When the input spike is present, the output spike is more likely to occur at one side of t_i, depending on whether the synapse is excitatory or inhibitory. Contributions from other input neurons appear as noise, and they can be easily filtered out if they are not correlated. The mild assumption that spike timing of each neuron is somewhat uncorrelated holds for most SNNs, especially for those that use a statistical

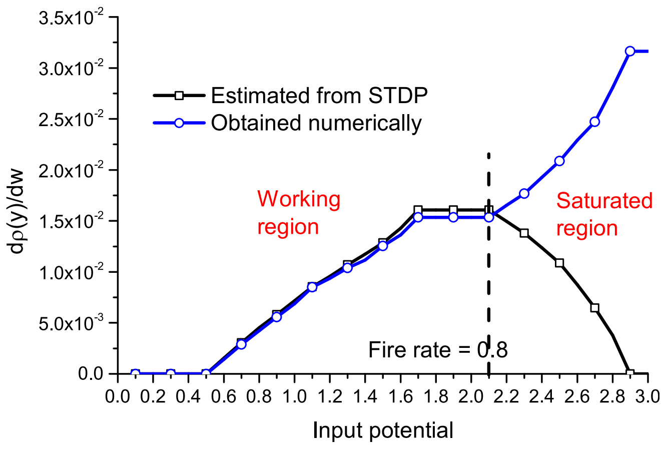


Fig. 3. Gradients obtained from STDP and numerical simulation as input membrane potential increases. Two sets of results match well until fire rate of the input neuron reaches 0.8. The gradient estimation becomes inaccurate in saturation region because it is difficult to distinguish a causal spike from an anti-causal spike when the input spike is too dense.

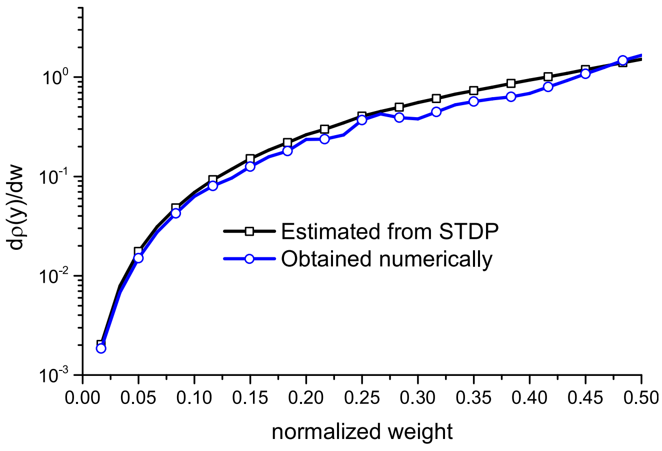


Fig. 4. Gradients obtained from STDP and numerical simulation as weight of the synapse increases. Two sets of results match well.

model. Even though the density of each neuron might be highly correlated (especially considering that sparse coding is often employed), the spike timing of individual spikes can be largely uncorrelated. Therefore, (13) can be readily employed in a large network, and individual gradients can be obtained simultaneously. This shares the same spirit as the simultaneous perturbation stochastic approximation (SPSA) [30].

Next, we assume (14) can approximately describe the input-output relation in the chosen neuron model, where $f_j(\cdot)$ is a function depending on the spiking neuron model used. Correctness of (14) is well justified, as it is actually the basis for artificial neural networks.

$$\rho\left(y_j(t)\right) = f_j\left(\sum_i w_{i,j}\rho(x_i(t))\right). \tag{14}$$

Then, with (13) and (14), we arrive at our second key equation.

$$\frac{\partial\rho\big(y_j(t)\big)}{\partial w_{i,j}} = \rho(x_i(t))f_j'\left(\sum_i w_{i,j}\rho(x_i(t))\right) = \frac{\rho\big(stdp_{i,j}(t)\big)}{w_{i,j}}. \tag{15}$$

Equation (15) is highly similar to the multiplicative STDP learning approach in literature. The denominator $w_{i,j}$ in the equation is not present in previous literature [17], [18]. We believe this denominator plays a key role in learning. Mathematically speaking, including the weight provides at least the sign information. If negative weights are allowed, then one definitely needs a term to change the sign in (15), which would otherwise induce a wrong direction for gradient descent. Another point is that the introduction of the denominator ensures an upper bound on $w_{i,j}$. This is also the primary reason that a multiplicative STDP learning rule is first introduced.

To verify (15), we conduct simulations on a network with five input neurons and one output neuron. In the first simulation, input membrane potential of one input neuron is swept, whereas input potentials of other input neurons are kept at fixed-yet-randomly-selected values. In the second simulation, spike densities of all five input neurons are fixed, while the weight of one synapse is swept. In both simulations, a leaky integrate-and-fire (LIF) neuron model is used. A small amount of noise is injected into all neurons to reduce correlation of spike timing of each neuron. In Fig. 3, results obtained from STDP match well the numerical results until the firing rate of the neuron reaches a certain level. This is caused by the discrete-time nature of the algorithm. When the firing rate is too large, it becomes difficult to classify an output spike as a causal or an anti-causal spike, since input spikes occupy almost every possible slots on the time-line. Fortunately, saturation can be avoided by keeping the time resolution of a discrete-time design fine enough with the price of a faster clock. In Fig. 4, good agreements are achieved between results estimated based on (15) and the gradient obtained numerically, justifying the existence of the denominator. Accuracy of (15) depends largely on how well the model in (14) can describe the actual dynamics of the chosen neuron model. Limitation of (14) lies in the fact that only the product of weights and the input spike density matter. That is, one would have obtained the same result if he or she scaled the weights and input spike density reversely. This is approximately true for an integrate-and-fire model when the weights and input spike density is not too large. Again, this can be achieved if the clock frequency is high enough.

4 Reinforcement Learning with STDP

In an actor-critic reinforcement learning task, the most critical part is the state-value function learning. With (15), weights of each synapses can be updated according to (16).

$$\frac{\partial w_{i,j}(t)}{\partial t} = \eta\delta(t)\frac{\partial\rho\big(y_j(t)\big)}{\partial w_{i,j}(t)} = \eta\delta(t)\frac{\rho\big(stdp_{i,j}(t)\big)}{w_{i,j}(t)}, \tag{16}$$

where η is the learning rate, and $\delta(t)$ is the temporal difference TD error as defined in (17).

$$\delta(t) = \gamma V(t) - V(t-\Delta t) + R(t), \tag{17}$$

where γ is the discount factor, and $R(t)$ is the reward received at time t.

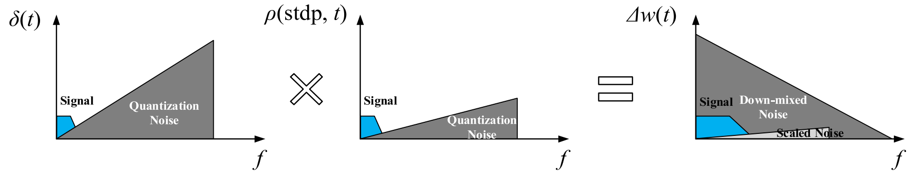


Fig. 5. Illustration of the noise down-mixing process. Quantization noises in $\delta(t)$ and $\rho(stdp_{i,j}(t))$ are mostly at high frequency. After multiplication, however, the noise is down-mixed to the baseband, saturating the desired signal. In addition, because the two quantization noises are correlated, the resultant down-mixed noise is severely biased.

To utilize (17), proper filtration is needed to suppress the down-mixed noise. To illustrate this, let us denote the TD error and spike-timing (ST) information as

$$\delta\ (t) = \delta_0\ (t) + n_\delta(t) \tag{18}$$

$$\rho\ \left(stdp_{i,j}(t)\right) = \rho_0\ \left(stdp_{i,j}(t)\right) + n_\rho(t), \tag{19}$$

where $\delta_0(t)$ and $\rho_0(stdp_{i,j}(t))$ are the signal part that we are interested in, and $n_\delta(t)$, $n_\rho(t)$ are the quantization noise.

With (18) and (19), we have

$$\frac{\partial w_{i,j}}{\partial t} = \frac{\eta}{w_{i,j}}\ \left(\delta_0(t)\rho_0\left(stdp_{i,j}\right) + \delta_0(t)n_\rho(t) + \rho_0\left(stdp_{i,j}\right)n_\delta(t) + n_\rho(t)n_\delta(t)\right). \tag{20}$$

Similar to a $\Sigma - \Delta$ modulator, the quantization noises shown in (18) and (19) have a big chunk of energy located at high frequency. This is particularly true for $n_\delta(t)$, to which a difference-like filter is applied as shown in (17). Scaled versions of noise $n_\rho(t)$ and $n_\delta(t)$ in (20) cause little trouble since the results in (20) are summed up over time, which is equivalent to passing through an integrator. Therefore, high-frequency noise in $n_\rho(t)$ and $n_\delta(t)$ are filtered out by the integrator. Consequently, the learning process is somewhat robust against this type of noise. The term $n_\rho(t)n_\delta(t)$ in (20) is much more troublesome, as they produce a down-mixed noise as shown in Fig. 5 and (21). In the equation, $\Phi(\cdot)$ stands for the autocorrelation, and $\mathrm{F}(\cdot)$ and $\mathrm{F}^{-1}(\cdot)$ stand for Fourier and inverse Fourier transform, respectively.

$$n_\rho(t)n_\delta\ (t) = \mathrm{F}^{-1}\ \left(\mathrm{F}\left(\Phi\left(n_\rho(t)\right)\right) * \mathrm{F}(\Phi(n_\delta(t)))\right). \tag{21}$$

Equation (21) cannot be solved analytically, as the phase information of noise spectrum is generally unknown. Qualitative analysis, however, is available with the help of numerical simulations. The down-mixed noise that appears at low frequency can be several orders of magnitude more significant than the desired signal part. To make things worse, the quantization noise $n_\delta(t)$ and $n_\rho(t)$ are correlated, making the down-mixed noise severely biased. Therefore, to avoid being saturated by the down-converted noise, proper filtration is needed before multiplication. From a circuit designer's point of view, one should use a sophisticated filter for $V(t)$ or $\delta(t)$, and simple and low-power filters for $stdp_{i,j}$. For example, a well-designed finite impulse response (FIR) filter with a growing stopband attenuation can be used for the state-value function, whereas a simple moving-average filter can be used for the spike timing information. There are two reasons for this. The first one is that $\delta(t)$ has much more high-frequency noise compared to $stdp_{i,j}$, as the noise is shaped twice: once by the neuron and the second time by (17). Another reason is that $\delta(t)$ only needs to be filtered once, and then it can be used for all $stdp_{i,j}$, whereas each $stdp_{i,j}$ needs to be filtered separately.

After filtration, decimation can be readily employed to reduce power consumption significantly, roughly $\log_2(M)/M$ times less than the original one, where M is the down-sampling rate. The down-sampling process is illustrated in Fig. 6. Decimation is also very helpful in a fixed-point implementation, which is very popular in hardware implementations. This is because decimation gain can effectively boost the weight changes, reducing the number of bits needed for storing the weights.

To demonstrate the proposed method, let us first consider a 1-D problem. A critic network similar to the one in [18] is employed in the simulation. The under-test critic network is shown in Fig. 7. An agent is moving under a fixed policy toward the target. Input neurons are place cells corresponding to different locations. The input potential $p_i(t)$ to each input neuron is computed as

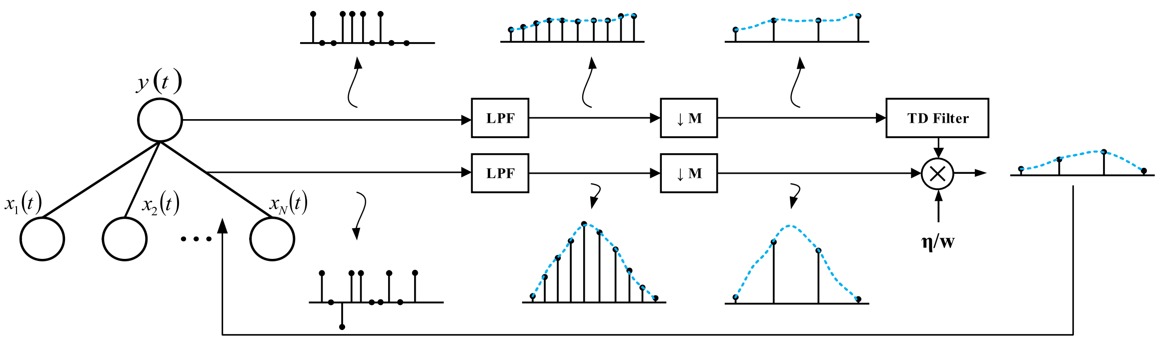


Fig. 6. Illustration of the decimation process. Down-sampling is taken place after filtration. This process effectively reduces the operating frequency of corresponding blocks. In addition, the associated decimation gain is also helpful in reducing the number of bits needed for the synapse weights.

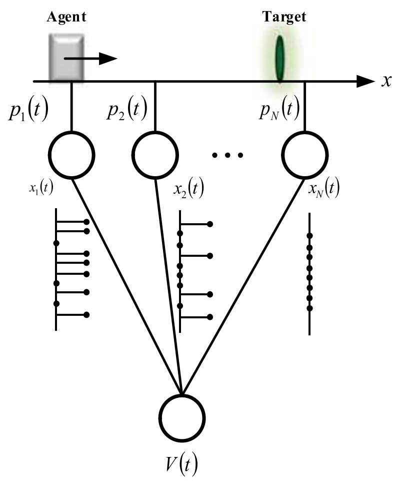


Fig. 7. Configuration of the 1-D test case. An agent is moving toward a target under a fixed policy. Each input neuron is a place cell that fires intensively when the agent is close to its center.

$$p_i(t) = k(x - c_i), \tag{22}$$

where x is the current location of the agent, c_i is the center of the input neuron i, and $k(\cdot)$ is the kernel function similar to those used in radial basis function (RBF) networks. The most popular choice is a Gaussian kernel.

Upon reaching the target, a reward is received by the agent. For each training (or iteration), the agent is first placed at a fixed starting point and is moved toward the target at a constant speed. Learning takes place when the task is performed. After each training, the agent is put back to the starting point, and a new training phase begins. The agent thus learns the state-value function in this process through TD learning.

Fig. 8 shows the state-value function obtained after a few trainings and the reference result obtained analytically. At the place $x = 300$, a reward is received by the agent. Each state is then assigned credits for the ultimate reward as shown in the figure, which forms an exponential curve. The back-propagation of TD error is clearly shown in the figure, and the results are well matched with the reference.

To demonstrate the importance of a proper filtration, Fig. 9 shows the root-mean-square error (RSME) between

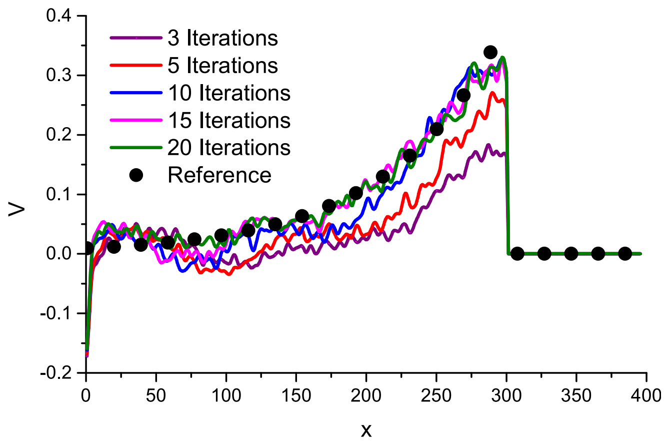


Fig. 8. Comparison of outputs of the critic with analytical result. Back-propagation of the TD error is clearly shown in the figure. Results obtained after 15 iterations start matching the reference result well.

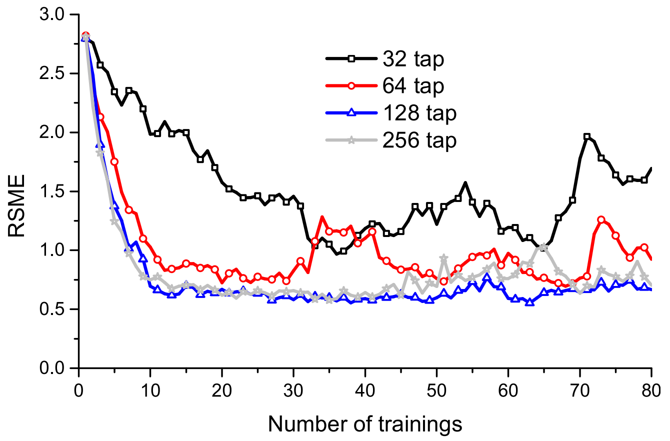


Fig. 9. Comparison of outputs of the critic network when filters of different orders are used. Learning rates are lowered for cases where only few taps are used. Moving average FIR filters are used for a fair comparison. The result obtained with a 32-order filter barely features a learning. The learning becomes more effective as the order of the filter increases. The improvement starts saturated when the order of the filter reaches 128.

output of the critic network and the reference when filters with different orders are applied to $stdp_{i,j}$ and $\delta(t)$. In order to give a fair comparison, all filters are moving average FIR filters. The only difference is the number of taps used. Learning rates are reduced for cases with fewer taps, which would otherwise diverge. As clearly shown in the figure, as the order of the filter increases, the learning is more effective.

We have previously assumed that input spikes are uncorrelated. This is a good assumption in general. Even when a deterministic IF neuron model is used, the input spike trains are reasonably uncorrelated if the input kernel is non-linear and the input is varying. In the case where one has to use a simple liner kernel, or in cases where one simply wants to use a statistical neuron model to improve the performance, noise can be added into the IF neuron model. Injecting noise, however, is an expensive operation, especially in a hardware implementation, since pseudo-random number generators that run as fast as neurons are needed for each neuron.

To tackle this problem, we propose to leverage the readily available quantization noise to implement the statistical model. Fig. 10a shows an IF model with a noisy threshold. Another statistical model, the noisy residue model shown in Fig. 10b behaves similarly to a noisy threshold model. The difference is that noise is injected every clock cycle in a noisy threshold model and every spike in a noisy residue model. In Fig. 10c, instead of injecting white noise residue, quantization residue is employed as "free" noise. With such a noise injection mechanism, each IF neuron behaves as a first-order $\Sigma - \Delta$ modulator. Even though the quantization noise added to each neuron, in this case, depends on the input of that neuron, quantization noises of neurons become reasonably uncorrelated in practical cases where inputs are varying. This is because the quantization noise is a function of integration of all past states of that neuron, which would normally differ for neurons. In addition to injecting noise, a discrete-time IF neuron with a quantization residue is closer to a continuous-time IF neuron, as it suffers less from the spike timing quantization problems. Therefore, we utilize this neuron model in the rest of this paper unless otherwise stated.

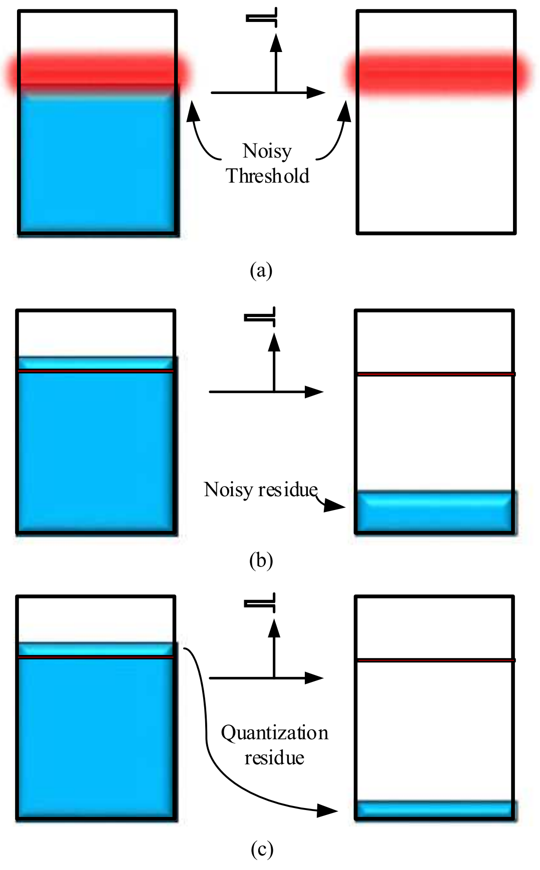


Fig. 10. Illustration of three different ways of injecting noise into neuron models: (a) noisy threshold, (b) noisy residue where the noise is white, (c) quantization residue.

RSMEs obtained from three cases are plotted in Fig. 11 for a Gaussian kernel and Fig. 12 for a triangular kernel. The first case is learning with a deterministic IF model. The second case is training with a statistical IF model, where a noisy residue with a white spectrum is used. The third case is the IF model with a quantization noise residue. As shown in the figures, all three testing cases can achieve impressive learning results when a Gaussian kernel is used. The deterministic model is slightly inferior to the other two cases. When the triangular kernel is employed, the deterministic model behaves remarkably worse than the other two statistical models. The average magnitudes of correlations between each pair of neurons are compared in Fig. 13. The correlation is measured for two adjacent input neurons when the agent is in between them. As clearly shown in the figure, the poorly behaved triangular kernel based deterministic model has a high correlation among spikes. With the help of the proposed quantization noise injection, a low correlation value can be achieved and the learning performance is significantly improved.

Next, a 2-D test case with a critic-actor network is examined. An agent is placed at a random starting point in a 120 by 120 maze. It moves at a constant speed, and the moving direction is controlled by the actor network. The objective of the agent is to reach the target while avoiding walls and obstacles. The agent initially has no knowledge about the target, the walls and the obstacles. It learns from each trial through reinforcement. The critic network in the previous 1-D example can be readily adapted into this 2-D maze problem. To complete the actor-critic network, we still need

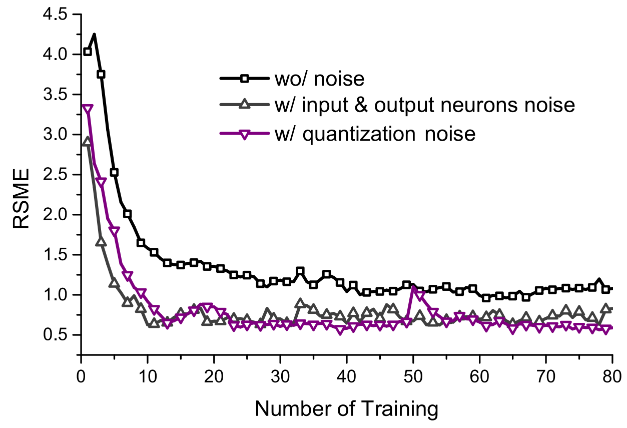


Fig. 11. Comparison of RSMEs obtained with a Gaussian kernel and three different neuron models: the deterministic model, the statistical model with white noise injected as membrane potential residue, and the statistical model with quantization noise residue.

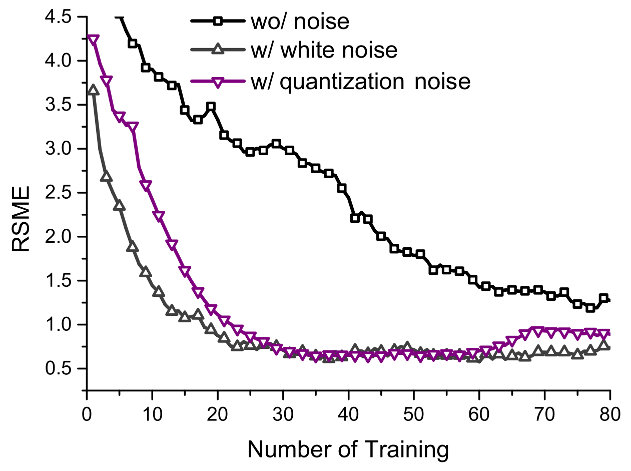


Fig. 12. Comparison of RSMEs obtained with a triangular kernel and three different neuron models: the deterministic model, the statistical model with white noise injected as membrane potential residue, and the statistical model with quantization noise residue.

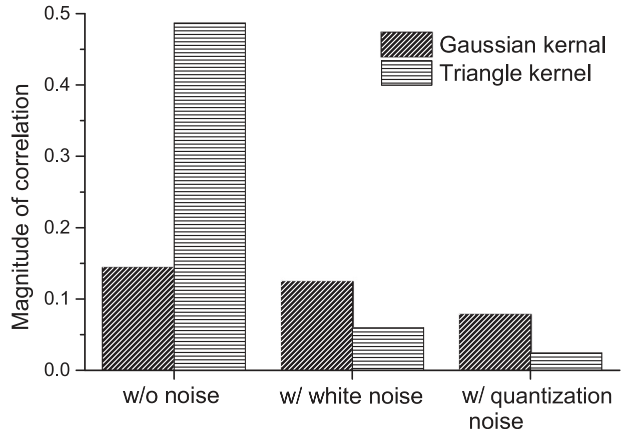


Fig. 13. Magnitudes of correlations obtained from the simulation. Correlations of spikes from adjacent neurons are measured. The deterministic neuron model with a Gaussian kernel and a varying input can achieve reasonably low correlations, whereas a statistical model is needed when a simple kernel is used to ensure a low correlation among spikes. The proposed quantization noise injection is effective in randomize the spike timing of each neuron.

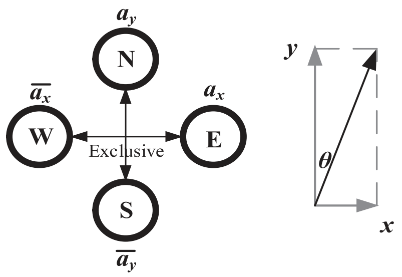


Fig. 14. Actor neurons employed in the maze problem. Action a_k and $\overline{a_k}$ are mutual exclusive in nature. The actual moving direction is interpolated according to the relative firing rate of each neuron.

the actor network. The actor neurons we use are similar to the critic neurons. In the maze problem, we propose to use four actor neurons, as shown in Fig. 14, that represent north, south, west and east, respectively. At every time unit, neuron N and neuron S compare their potentials, and the one with a larger potential spikes. After firing, membrane potentials of both neurons are reset to zero. A similar operation is conducted for E and W neurons. This method behaves similarly to a winner-take-all method. However, this way of operating avoids the trial-and-error tuning for the inhabitation weights in [18]. To conduct exploration, noise is injected into the actor neurons. The same learning rule as shown in (16) is employed for the actor neurons. In this maze problem, we leverage the mutual exclusiveness and the orthogonality of actions, leading us to a policy as shown in (23).

$$\begin{aligned}\pi\left(s,\ a_k\right) =&\ P\left(a(t),\ s(t)\right)\\ &= P\left(\sum_i\left(w_{i,j}^{a_k} - w_{i,j}^{\overline{a_k}}\right)\rho(x_i(t)) + n_k\ > 0\right).\end{aligned} \tag{23}$$

where $\pi(s,\ a)$ is the policy, $s(t)$ is the current state, $a(t)$ is the chosen action, a_k is the action k, $\overline{a_k}$ is the opposite of a_k, and n_k is the injected noise.

Equation (23) behaves similarly to a Gibbs softmax method [29] in the sense that the preferred actions (action with larger weights) is chosen with a high probability, while other actions are chosen with low probabilities for exploration purposes. Weights are updated according to (16) such that the taken actions are reinforced if a positive TD error is received and vice versa.

A more general policy as shown in (24) can be applied when mutual-exclusiveness nature is not present. In (24), the most preferred action is taken with the highest probability.

$$\begin{aligned}\pi\left(s,\ a_k\right) =&\ P\left(a(t),\ s(t)\right)\\ &= P\left(\underset{k}{\text{argmax.}}\left(\sum_i w_{i,j}^{a_k}\rho(x_i(t)) + n_k\right) = k\right).\end{aligned} \tag{24}$$

Spikes from each actor neurons are filtered, and the moving direction is determined by

$$\theta\left(t\right) = \tan^{-1}\left(\frac{\rho(E(t)) - \rho(W(t))}{\rho(N(t)) - \rho(S(t))}\right). \tag{25}$$

Two sets of simulations are conducted, and the results are shown in Figs. 15, 16, 17, and 18. Eighty-one place cells are used. For each training, the agent is placed at a random starting point in the maze. The agent then starts wandering in the maze at a constant speed. Its moving direction is determined according to (25). The agent is bounced back and receives a punishment (negative reward) when it hits the obstacle or walls. Whenever the agent reaches the target (less than 3 percent of the total area), a reward is delivered and the current

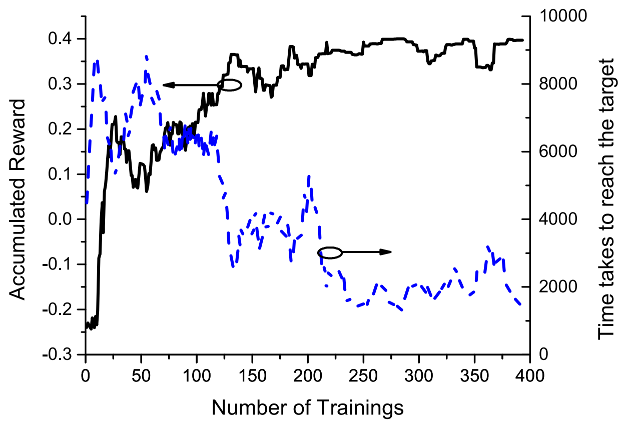


Fig. 16. Accumulated reward obtained and time it takes to reach the target versus the number of training conducted. Maze configuration is shown in Fig. 18. As the training goes on, the agent gradually learns about its environment and starts receiving more rewards.

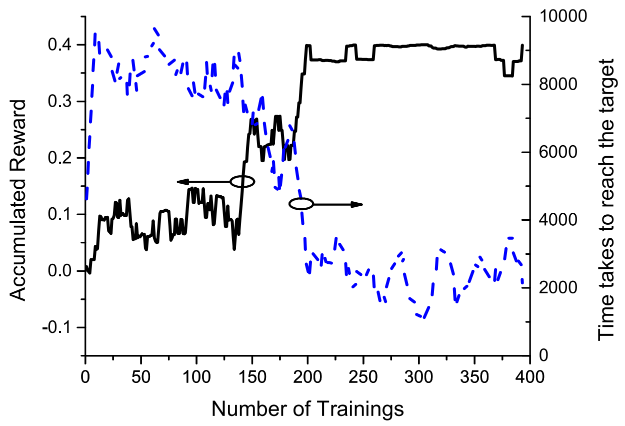


Fig. 15. Accumulated reward obtained and time it takes to reach the target versus the number of training conducted. Maze configuration is shown in Fig. 17. As the training goes on, the agent gradually learns about its environment and starts receiving more rewards.

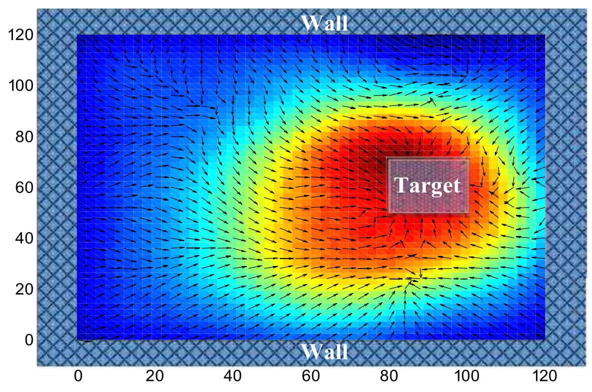


Fig. 17. State-value function and preferred actions outputted by the critic and actor networks. The brighter the color is, the higher chance the agent “thinks” there is a reward. Arrows in the figure represent the preferred moving directions.

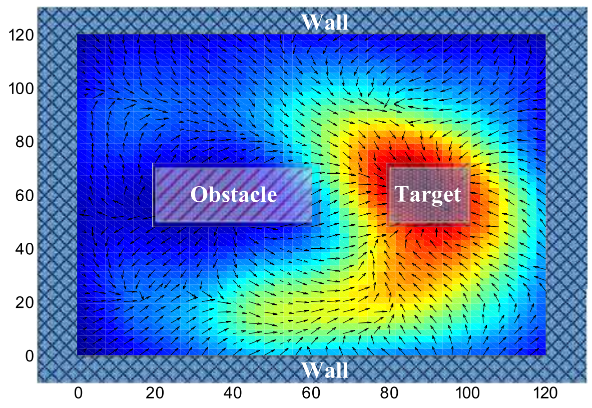


Fig. 18. State-value function and preferred actions outputted by the critic and actor networks. The brighter the color is, the higher chance the agent "thinks" there is a reward. Arrows in the figure represent the preferred moving directions. Compared to Fig. 17, color around the obstacle is darker, indicating a possible punishment. Actions are changed accordingly to avoid the obstacle.

training cycle is terminated. The training cycle may also be terminated if the running time of the agent exceeds a preset time-out limit. Once a training cycle ends, the agent is put at a new random starting point in the maze and another training cycle begins. Maze configurations are shown in Figs. 17 and 18. For the first set of simulations (Figs. 15 and 17), no obstacle is placed in the maze. For the second set of simulations (Figs. 16 and 18), one obstacle is placed in the maze.

Accumulated rewards received by the agent and the time it takes for the agent to reach the target during each training cycle are plotted in Figs. 15 and 16. For a better illustration, results in the figures are filtered to show the trend of successful learnings. As shown in the figures, the agent initially is naive about the environment. Therefore, little or even negative reward is received. The agent cannot reach the target within the time-out limit, which is set at 10,000 in our experiments. After a number of trainings, the agent learns how to avoid walls and obstacles and move toward the target. Figs. 17 and 18 show outputs of critic network and preferred actions picked by the actor network. Color maps in the figures represent the relative magnitude of output of the critic network. Arrows in the figure represent the preferred action at that point. The brighter the color is, the larger the output of the critic network is. A larger critic output means that the agent "thinks" there is a higher chance it can receive reward in those regions. Accordingly, actor network leads the agent toward those rewarding regions.

5 Hardware Architecture

A hardware architecture for the proposed algorithm is shown in Fig. 19. This architecture is generic, and many detail circuit implementations can fit into it. For example, neuron blocks in Fig. 19 can be implemented as analog circuits to take advantage of power efficiency of analog circuits at the low bit-precision region [31]. Alternatively, asynchronous digital neurons [11] can be employed to save clock distribution power and power dissipated in clocking flip-flops. In the figure, there are two memory banks. They are partitioned according to their chance of being visited. Memory A stores weight information and tick-level spike timing information (see Fig. 20 for the definition of a tick), and it may be accessed at every clock cycle. Memory B stores the decimated ST information, and it shall be accessed at every unit interval (see Fig. 20 for the definition of a unit interval).

For a proof-of-concept demonstration, the proposed algorithm is implemented with synchronous digital circuits using Verilog. The timing diagram is shown in Fig. 20 for illustration purposes. The minimal time unit that a spike can occur is called a tick following the convention used in [11]. Within a tick, several clock cycles are needed to run through all input neurons. Each input neuron takes two clock cycles to complete its evaluation: one clock cycle for reading memory and the other for writing. During the read

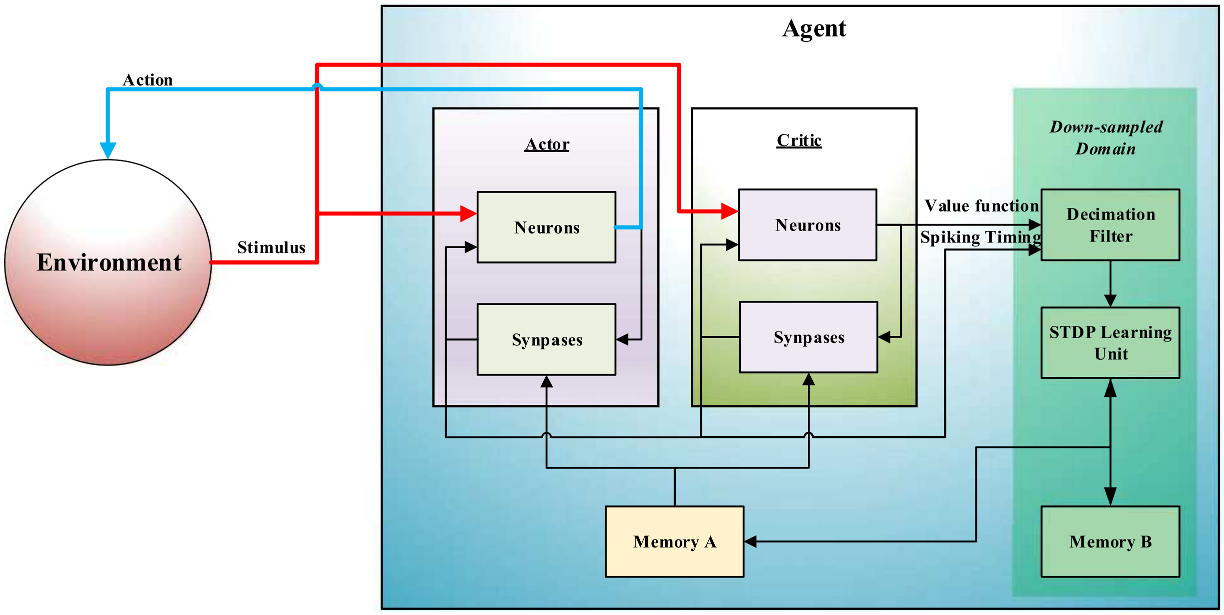


Fig. 19. A hardware architecture for the STDP learning rule based actor-critic reinforcement learning. Actor and critic are two SNNs with same inputs. Memory A and memory B are partitioned according the frequency they are accessed. Memory A stores synapse weights and tick-level spike timing information, and it may be read or write at every tick. Memory B stores decimated spike timing information, and its chance of being accessed is much lower than memory A.

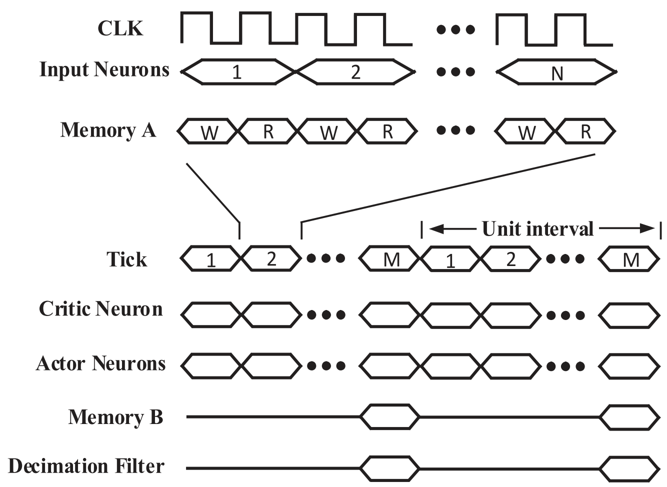


Fig. 20. An example of timing diagram for the proposed hardware architecture. Each input neuron takes two clock cycles to be evaluated. One cycle for reading weight, and another cycle for writing spike timing information. States of all neurons are updated every tick. Blocks in the down-sampled domain operate at a much lower rate.

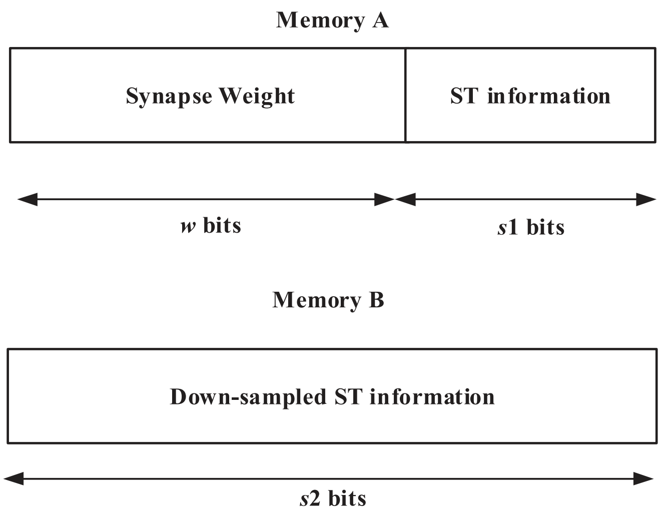


Fig. 21. Memory contents in the implementation example. Memory A stores w-bit weights and $s1$-bit ST information. Memory B stores $s2$-bit down-sampled ST information.

cycle, weights are read out from the memory if the current input neuron spikes at present tick or previous tick. During the write cycle, ST field is updated. At the end of each tick, all neurons are evaluated. Logic units, such as adders and comparators used in evaluating neuron states are shared among all neurons. Parallel computation can be introduced for large networks with little overhead, as computations in spiking neural networks are parallel in nature.

In each unit interval, spikes from the critic neuron is filtered, and the TD error is produced. Spike timing information is read out from memory B, and updated weights are computed. It is worth noting that even though the access of memory B and filtration are illustrated at the last tick of each unit interval, they can be spread out within one unit interval in a time-interleaved fashion to save hardware resources or relax the requirement on the critical path delay.

In (16), a division operation is needed. Fixed-point division is relatively expensive in hardware. It normally takes multiple clock cycles to complete, or otherwise a pipelined divider is usually needed. Therefore, we employed an approximate division in our implementation: that is, we replaced the divisor with the closest number that is a power of two. Consequently, the division operation can be easily accomplished by a round-and-shift operation. We compare simulation results obtained by the exact division and approximate division in Fig. 22. There is hardly any noticeable difference between the two results. Fig. 22 also shows the result obtained with no division at all (the learning rate is, of course, adjusted accordingly). To further study the effect of the denominator in (16), we compare learning performances under three learning rates for the case where approximate division is employed, as well as the case where no division is used. The results are illustrated in Fig. 23. It is noticed that learning conducted according to (16) is more robust to changes in learning rate. The converging process is also faster and smoother. On the contrary, learning without division is very sensitive to the changes in learning rate. To avoid divergence, one has to tolerate a slow learning speed. This may also possibly explain why learning is possible in literature, where a STDP learning rule without division is used. The observation that learning is still feasible (however slow), even when low-accuracy divisions are used, is good news for implementations with analog neurons. Analog neurons have unique advantages such as a lower power consumption and a smaller footprint compared to digital neurons. The STDP learning rule is particularly useful in this case because the exact dynamics of an analog neuron cannot be controlled completely due to the process, voltage and temperature (PVT) variation. This problem is, however, mitigated with the use of the STDP learning rule, as spike timing provides a good way of estimating gradients regardless of the model parameters (such as leakage, threshold). Despite of the energy-efficiency of analog neurons, routing of spikes should be conducted through digital circuitry to ensure robustness and scalability.

As shown in Fig. 20, most operations conducted are reading and writing memory, as it is normally the case in hardware neural networks. In terms of energy consumption, memory access is also likely to be the most expensive operation in a hardware implementation. Fortunately, the event-based nature of SNNs makes the access of memory event-triggered, saving power consumption. Fig. 21 shows the content of each word in memory A and B. Each word in memory A has w bits for weights and $s1$ bits for spike timing information. The number of bits needed for weights varies with tasks. A parametric sweep on w is conducted for the 1-D critic example. The result is shown in Fig. 24. As shown in the figure, the numbers of bits needed for learning and performing the task are different. It is well known that neural networks are able to work well, even when the computation precision is low. Therefore, only few bits are needed for a neural network to conduct a task as shown in Fig. 24. When learning is taking place, however, small weight updates need to be built up in the memory at each reinforcement. That is why more bits are needed for learning. This phenomenon is previously reported in literature [10]. The technique of partitioning the memory into learning and performing banks can also be used in our case to reduce the run-time power consumption after the training is completed.

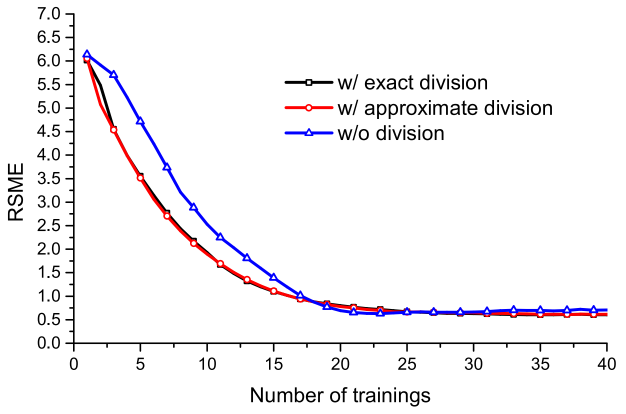


Fig. 22. RSMEs of outputs of critic network obtained when exact division shown in (16) is used, when simple division with rounded divisor is used and when no division is used at all. The approximate division does not yield noticeable performance degradation compared to the exact division. Results obtained with no division is slightly worse.

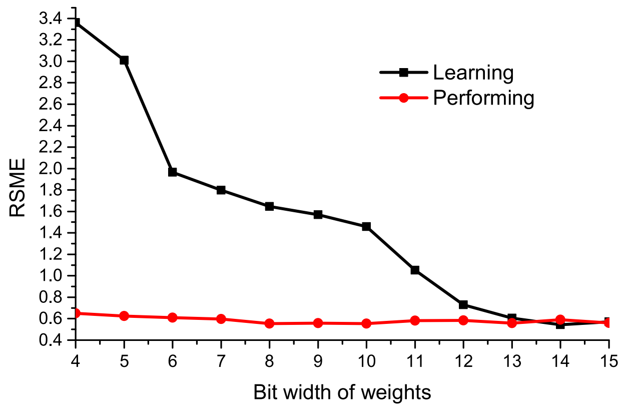


Fig. 24. RSMEs of outputs of critic network versus the number of bits used to represent synapse weights. Only few bits are necessary to perform a task after being well trained. Much more bits are needed for a learning to be successfully conducted. This is mainly because small changes in weights need to be accumulated during learning, which needs a high resolution in weights.

The number of memory bits used for storing weights is

$$N_{weight} = \prod_k N_k(1 + N_{action})w, \tag{26}$$

where N_k is the number of states associated with dimension k, and N_{action} is the number of possible actions.

Equation (26) is obviously affected by the well-known curse-of-dimensionality. From the performance point of view, a large memory size often limits the speed of accessing memory. Fortunately, in a SNN, memory can be made local to neurons to boost the performance in a large network, as has already been demonstrated by the TrueNorth [11]. In addition, thanks to the highly correlated nature of the sparse coding, memory hierarchy used in modern computer architecture can be readily employed to eliminate the bottleneck in memory accessing. Therefore, speed seems not to be a problem even when the size of the network becomes large. The great amount of memory needed to solve a huge task, however, still poses a problem on power consumption. Fortunately, the number of dimensions can usually be reduced by choosing state space smartly, while the size of each dimension can be made compact enough through a self-adapting kernel similar to the technique that is well-known for RBF networks [32]. In addition, recent progress in nanotechnologies has significantly sped up memory scaling [33], [34], enabling the capability to solve larger problems with reinforcement learning.

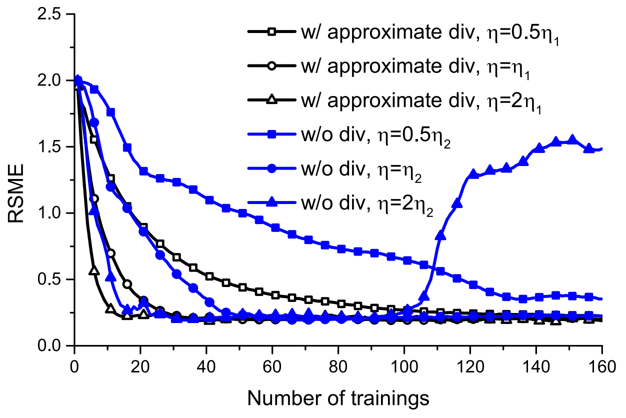


Fig. 23. Comparison of learning performances achieved with multiplicative STDP learning rules (w/ dividing by weights) and non-weight-dependent STDP learning rules (w/o dividing by weights). Three different learning rates are employed. η_1 and η_2 are two manually chosen constants.

$s1$ and $s2$ in Fig. 21 depend on the chosen decimation filter and down-sampling ratio. In our implementation, a cascaded integrator-comb (CIC) filter is used for decimating $stdp_{i,j}$. This choice can help reduce the number of bits needed in memory. When the first order CIC filter is used, $s1$ is equal to $\log_2 M$, where M, again, is the down-sampling ratio. $s2$ is equal to $(N_{tap}/M + 1)\log_2 M$, where N_{tap} is the order of filter used to filter the state-value function. Memory B serves as a delay unit for ST information. The delay is used to match the group delay induced by filtering the state-value function outputted by the critic network.

In a naive manner, the memory requirement for the spike timing field is similar to (26) except w in the equation is replaced with $(s1 + s2)$ in this case. We can, however, achieve a much smaller number, considering that only a small portion of states in a large state space is active at a particular period of time. When the sizes of dimensions of the problem become large, a specialized array of memory can be built for storing spike timing information. This memory behaves as a cache in the sense that it only stores spike timing information associated with synapses that are recently active. The difference, however, is that upon a cache miss, a new entry is created instead of visiting the main memory (there is no main memory in this case anyway). Whenever an entry has a $\rho(stdp_{i,j}(t))$ that is equal to zero, it is deleted from the cache. The main idea is that all inactive neurons have a $\rho(stdp_{i,j}(t))$ of zero, and we know it as a given. Therefore, the memory requirement for holding spike timing information is significantly relieved for a large design.

6 Conclusion and Future Work

In this paper, we formulate biologically plausible and hardware-friendly STDP learning rules. The STDP learning rule turns out to be a gradient-descent optimization algorithm.

Based on the learning rule, actor-critic network based reinforcement learning is presented. It is found that quantization noise is down-mixed to the baseband during the process of weights update which aims to minimize TD error. Proper filtration is therefore recommended to be used at output of the critic to eliminate the noise prior to mixing. Decimation after filtration is proposed as an effective way to reduce the operating frequency of learning related blocks, as well as to reduce the number of bits needed to represent synapse weights in a learning task. Effects of correlated spikes are also studied, and a quantization noise injection technique is proposed to help reduce the correlations, boosting learning performance. A 1-D problem of the state-value function learning and a 2-D problem of maze walking are studied. The agent equipped with the proposed algorithms can effectively learn tasks. In addition, a low-power hardware architecture is proposed and examples are implemented in Verilog. To break the memory bottleneck as the size of the network grows large, possible solutions such as utilizing memory partitioning, hierarchy memory structure, and emergent memory technologies are also discussed.

In this paper, only two layers of neurons are considered, as it is the number of layers needed to complete tasks that are of interest. However, it should be straightforward to adapt (16) into a more general form which can be applied to a multi-layer configuration. The main difference will be that region partitioning shown in Fig. 2 needs to be adjusted according to the delay associated with the number of layers. In addition, besides the single-task scenario considered in this paper, multi-task situations can also be handled by the proposed method. For example, techniques such as the scaling factor proposed in [35] can be employed to handle multiple rewards. In future work, more general learning rules for multi-layer SNNs will be studied. In addition, building neuromorphic chips with analog neurons that takes advantage of the STDP learning rule will be another research direction.

Acknowledgments

This work is supported by National Science Foundation under grant CCF1421467.

References

[1] A. Cassidy, A. G. Andreou, and J. Georgiou, "Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis," in *Proc 45th Annu. Conf. Inf. Sci. Syst.*, 2011, pp. 1–6.

[2] Y. Maeda and T. Tada, "FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 688–695, Feb. 2003.

[3] N. D. Patel, N. S. Kiong, and G. G. Coghill, "Neural network implementation using bit streams," *IEEE Trans. Neural Netw.*, vol. 18, no. 5, pp. 1488–1504, Sep. 2007.

[4] M. J. Pearson, et al., "Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated FPGA approach," *IEEE Trans. Neural Netw.*, vol. 18, no. 5, pp. 1472–1487, Sep. 2007.

[5] B. V. Benjamin, et al., "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[6] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, "Energy-efficient neuron, synapse and STDP integrated circuits," *IEEE Trans. Biomed. Circuits Syst.*, vol. 6, no. 3, pp. 246–256, Jun. 2012.

[7] K. Joo-Young, K. Minsu, L. Seungjin, O. Jinwook, K. Kwanho, and Y. Hoi-Jun, "A 201.4 GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine," *IEEE J. Solid-State Circuits*, vol. 45, no. 1, pp. 32–45, Jan. 2010.

[8] L. Junjie, S. Young, I. Arel, and J. Holleman, "A 1 TOPS/W analog deep machine-learning engine with floating-gate storage in 0.13 μm CMOS," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 270–281, Jan. 2015.

[9] P. Junyoung, H. Injoon, K. Gyeonghoon, N. Byeong-Gyu, and Y. Hoi-Jun, "Intelligent network-on-chip with online reinforcement learning for portable HD object recognition processor," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 61, no. 2, pp. 476–484, Feb. 2014.

[10] P. Knag, K. Jung Kuk, T. Chen, and Z. Zhengya, "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, Apr. 2015.

[11] P. A. Merolla, et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, 2014.

[12] R. Serrano-Gotarredona, et al., "CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[13] I. E. Ebong and P. Mazumder, "CMOS and memristor-based neural network design for position detection," *Proc. IEEE*, vol. 100, no. 6, pp. 2050–2060, Jun. 2012.

[14] D. Shukai, H. Xiaofang, D. Zhekang, W. Lidan, and P. Mazumder, "Memristor-based cellular nonlinear/neural network: Design, analysis, and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 6, pp. 1202–1213, Jun. 2015.

[15] F. Deliang, S. Yong, A. Raghunathan, and K. Roy, "STT-SNN: A spin-transfer-torque based soft-limiting non-linear neuron for low-power artificial neural networks," *IEEE Trans. Nanotechnol.*, vol. 14, no. 6, pp. 1013–1023, Nov. 2015.

[16] P. Junyoung, K. Joonsoo, O. Jinwook, L. Seungjin, K. Joo-Young, and Y. Hoi-Jun, "A 92-mW real-time traffic sign recognition system with robust illumination adaptation and support vector machine," *IEEE J. Solid-State Circuits*, vol. 47, no. 11, pp. 2711–2723, Nov. 2012.

[17] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Comput.*, vol. 19, pp. 1468–1502, 2007.

[18] N. Frémaux, H. Sprekeler, and W. Gerstner, "Reinforcement learning using a continuous time actor-critic framework with spiking neurons," *PLoS Comput. Biol.*, vol. 9, no. 4, p. e1003024, 2013.

[19] W. Potjans, M. Diesmann, and A. Morrison, "An imperfect dopaminergic error signal can drive temporal-difference learning," *PLoS Comput. Biol.*, vol. 7, 2011, Art. no. e1001133.

[20] W. Potjans, A. Morrison, and M. Diesmann, "A spiking neural network model of an actor-critic learning agent," *Neural Comput.*, vol. 21, pp. 301–339, 2009.

[21] E. Vasilaki, N. Frémaux, R. Urbanczik, W. Senn, and W. Gerstner, "Spike-based reinforcement learning in continuous state and action space: When policy gradient methods fail," *PLoS Comput. Biol.*, vol. 5, pp. e1000586–e1000586, 2009.

[22] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, May 2000.

[23] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *Int. J. Comput. Vis.*, vol. 113, pp. 54–66, 2015.

[24] N. Srinivasa and J. M. Cruz-Albrecht, "Neuromorphic adaptive plastic scalable electronics: Analog learning systems," *IEEE Pulse*, vol. 3, no. 1, pp. 51–56, Jan. 2012.

[25] J. Sjöström and W. Gerstner, "Spike-timing dependent plasticity," *Spike-Timing Dependent Plasticity*, vol. 5, 2010, Art. no. 35.

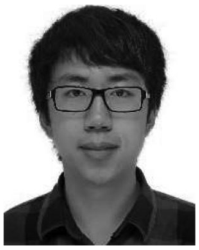
[26] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neurosci.*, vol. 3, pp. 919–926, 2000.

[27] H. Markram, W. Gerstner, and P. J. Sjöström, "Spike-timing-dependent plasticity: A comprehensive overview," *Frontiers Synaptic Neurosci.*, vol. 4, pp. 35–44, 2012.

[28] G. Hinton, "How to do backpropagation in a brain," in *Proc. NIPS'2007 Deep Learn. Workshop*, p. 15, 2007.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.

[30] J. C. Spall, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins APL Tech. Dig.*, vol. 19, pp. 482–492, 1998.
[31] R. Sarpeshkar, "Analog versus digital: Extrapolating from electronics to neurobiology," *Neural Comput.*, vol. 10, pp. 1601–1638, 1998.
[32] J. Lian, Y. Lee, S. D. Sudhoff, and S. H. Zak, "Self-organizing radial basis function network for real-time approximation of continuous-time dynamical systems," *IEEE Trans. Neural Netw.*, vol. 19, no. 3, pp. 460–474, Mar. 2008.
[33] M. Barangi and P. Mazumder, "Straintronics-based random access memory as universal data storage devices," *IEEE Trans. Magn.*, vol. 51, no. 5, May 2015, Art. no. 3400408.
[34] Y. Yilmaz and P. Mazumder, "Resistive memory structure for single or multi-bit data storage," U.S. Patent 20 150 332 763, 2015.
[35] P. Raicevic, "Parallel reinforcement learning using multiple reward signals," *Neurocomputing*, vol. 69, pp. 2171–2179, 2006.



Nan Zheng (S'13) received the BS degree in information engineering from Shanghai Jiao Tong University, Shanghai, China, in 2011, and the MS degree in electrical engineering from the University of Michigan, Ann Arbor, in 2014, where he is currently working toward the PhD degree in electrical engineering. In the summer of 2012, he had an internship at Qualcomm, CA, where he worked on developing antenna system for the next-generation communication network. His research interests include low-power circuit design, modeling, and optimization. He is a student member of the IEEE.

Pinaki Mazumder (S'84-M'87–SM'95-F'99) received the PhD degree from the University of Illinois at Urbana-Champaign, Urbana, in 1988. He is currently a professor in the Department of Electrical Engineering and Computer Science, University of Michigan (UM), Ann Arbor. He was for six years with industrial R&D centers that included AT&T Bell Laboratories, where in 1985, he started the CONES Project—the first C modeling-based very large scale integration (VLSI) synthesis tool at India's premier electronics company, Bharat Electronics, Ltd., India, where he had developed several high-speed and high-voltage analog integrated circuits intended for consumer electronics products. He is the author or coauthor of more than 200 technical papers and four books on various aspects of VLSI research works. His current research interests include current problems in nanoscale CMOS VLSI design, computer-aided design tools, and circuit designs for emerging technologies including quantum MOS and resonant tunneling devices, semiconductor memory systems, and physical synthesis of VLSI chips. He is a fellow of the American Association for the Advancement of Science (2008). He was a recipient of the Digital's Incentives for Excellence Award, BF Goodrich National Collegiate Invention Award, and Defense Advanced Research Projects Agency Research Excellence Award. He is a fellow of the IEEE.