



Reinforcement Learning With Low-Complexity Liquid State Machines

Wachirawit Ponghiran^{*†}, Gopalakrishnan Srinivasan[†] and Kaushik Roy

Department of ECE, Purdue University, West Lafayette, IN, United States

We propose reinforcement learning on simple networks consisting of random connections of spiking neurons (both recurrent and feed-forward) that can learn complex tasks with very little trainable parameters. Such sparse and randomly interconnected recurrent spiking networks exhibit highly non-linear dynamics that transform the inputs into rich high-dimensional representations based on the current and past context. The random input representations can be efficiently interpreted by an output (or readout) layer with trainable parameters. Systematic initialization of the random connections and training of the readout layer using Q-learning algorithm enable such small random spiking networks to learn optimally and achieve the same learning efficiency as humans on complex reinforcement learning (RL) tasks like Atari games. In fact, the sparse recurrent connections cause these networks to retain fading memory of past inputs, thereby enabling them to perform temporal integration across successive RL time-steps and learn with partial state inputs. The spike-based approach using small random recurrent networks provides a computationally efficient alternative to state-of-the-art deep reinforcement learning networks with several layers of trainable parameters.

Keywords: liquid state machine, recurrent SNN, learning without stable states, spiking reinforcement learning, Q-learning

OPEN ACCESS

Edited by:

Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:

Sadique Sheikh,
AICTX AG, Switzerland
Arash Ahmadi,
University of Windsor, Canada

*Correspondence:

Wachirawit Ponghiran
wponghir@purdue.edu

[†]These authors have contributed
equally to this work

Specialty section:

This article was submitted to
Neuromorphic Engineering,
a section of the journal
Frontiers in Neuroscience

Received: 10 April 2019

Accepted: 07 August 2019

Published: 27 August 2019

Citation:

Ponghiran W, Srinivasan G and Roy K
(2019) Reinforcement Learning With
Low-Complexity Liquid State
Machines. *Front. Neurosci.* 13:883.
doi: 10.3389/fnins.2019.00883

1. INTRODUCTION

High degree of recurrent connectivity among neuronal populations is a key attribute of neural microcircuits in the cerebral cortex and many different brain regions (Douglas et al., 1995; Harris and Mrsic-Flogel, 2013; Jiang et al., 2015). Such common structure suggests the existence of a general principle for information processing. However, the principle underlying information processing in such recurrent population of spiking neurons is still largely elusive due to the complexity of training large recurrent Spiking Neural Networks (SNNs). In this regard, reservoir computing architectures (Maass et al., 2002, 2003; Lukoševičius and Jaeger, 2009) were proposed to minimize the training complexity of large recurrent neuronal populations. Liquid State Machine (LSM) (Maass et al., 2002, 2003) is a recurrent SNN consisting of an input layer sparsely connected to a randomly interlinked reservoir (or liquid) of spiking neurons whose activations are passed on to a readout (or output) layer, trained using supervised algorithms, for inference. The key attribute of an LSM is that the input-to-liquid and the recurrent excitatory ↔ inhibitory synaptic connectivity matrices and weights are fixed *a priori*. LSM effectively utilizes the rich non-linear dynamics of Leaky-Integrate-and-Fire spiking neurons (Dayan and Abbott, 2003) and the sparse random input-to-liquid and recurrent-liquid synaptic connectivity for processing spatio-temporal

inputs. At any time instant, the spatio-temporal inputs are transformed into a high-dimensional representation, referred to as the liquid states (or spike patterns), which evolves dynamically based on decaying memory of the past inputs. The memory capacity of the liquid is dictated by its size and degree of recurrent connectivity. Although the LSM, by construction, does not have stable instantaneous internal states like Turing machines (Savage, 1998) or attractor neural networks (Amit, 1992), prior studies have successfully trained the readout layer using liquid activations, estimated by integrating the liquid states (spikes) over time, for speech recognition (Auer et al., 2002; Maass et al., 2002; Verstraeten et al., 2005; Bellec et al., 2018), image recognition (Srinivasan et al., 2018), gesture recognition (Chrol-Cannon and Jin, 2015; Panda and Srinivasa, 2018), and sequence generation tasks (Nicola and Clopath, 2017; Panda and Roy, 2017; Bellec et al., 2019).

In this work, we propose such sparse randomly-interlinked low-complexity LSMs for solving complex Reinforcement Learning (RL) tasks, which involve an autonomous agent (modeled using the LSM) trained to select actions in a manner that maximizes the expected future rewards received from the environment. For instance, a robot (agent) learning to navigate a maze (environment) based on the reward and punishment received from the environment is an example RL task. The environment state (converted to spike trains) is fed to the liquid, which produces a high-dimensional representation based on current and past inputs. The sparse recurrent connections enable the liquid to retain decaying memory of past input representations and perform temporal integration across different RL time-steps. We present an optimal initialization strategy for the fixed input-to-liquid and recurrent-liquid connectivity matrices and weights to enable the liquid to produce high-dimensional representations that lead to efficient training of the liquid-to-readout weights. Artificial rate-based neurons for the readout layer takes the liquid activations and produces *action-values* to guide action selection for a given environment state. The liquid-to-readout weights are trained using the Q-learning RL algorithm proposed for deep learning networks (Mnih et al., 2015). In RL theory (Sutton and Barto, 1998), the Q-value, also known as the action-value, estimates the expected future rewards for a state-action pair that specifies how good is the action for the current environment state. The readout layer of the LSM contains as many neurons as the number of possible actions for a particular RL task. At any given time, the readout neurons predict the Q-value for all possible actions based on the high-dimensional state representation provided by the liquid. The liquid-to-readout weights are then trained using backpropagation (Rumelhart et al., 1986) to minimize the error between the Q-values predicted by the LSM and the target Q-values estimated from RL theory (Watkins and Dayan, 1992) as described in subsection 2.2. We adopt ϵ -greedy policy (explained in subsection 2.2) to select the suitable action based on the predicted Q-values during training and evaluation. Based on ϵ -greedy policy, a lot of random actions are picked in the beginning of the training phase to better explore the environment. Toward the end of training and during inference, the action corresponding to the maximum Q-value is selected

with higher probability to exploit the learnt experiences. We first demonstrate the utility of the sparse recurrent connections in enabling the LSM to perform temporal integration across RL time-steps by training it to perform the Cartpole-balancing RL task (Sutton and Barto, 1998) with partial state inputs. We feed only the cart position and pole angle to the LSM while suppressing the cart velocity and angular velocity of the pole. We show that the fading memory of the past cart position and pole angle retained by the liquid enables it to make better decisions without the velocity information compared to an LSM without recurrent connections. We then comprehensively validate the capability of the LSM and the presented training methodology on complex RL tasks like Pacman (DeNero et al., 2010) and Atari games (Brockman et al., 2016). We note that LSM has been previously trained using Q-learning for RL tasks pertaining to robotic motion control (Joshi and Maass, 2005; Berberich, 2017; Tieck et al., 2018). We demonstrate and benchmark the efficacy of appropriately initialized LSM for solving RL tasks commonly used to evaluate deep reinforcement learning networks. In essence, this work provides a promising step toward incorporating bio-plausible low-complexity recurrent SNNs like LSMs for complex RL tasks, which can potentially lead to much improved energy efficiency in event-driven asynchronous neuromorphic hardware implementations (Merolla et al., 2014; Davies et al., 2018).

2. MATERIALS AND METHODS

2.1. Liquid State Machine: Architecture and Initialization

Liquid State Machine (LSM) consists of an input layer sparsely connected via fixed synaptic weights to a randomly interlinked liquid of spiking neurons followed by a readout layer as depicted in **Figure 1**. Each spiking neuron fires an action potential that leads to either excitatory or inhibitory effect at all of its termination sites. Based on the terminology followed in Maass et al. (2002) and Diehl and Cook (2015), we term a neuron that leads to excitatory (inhibitory) effect an excitatory (inhibitory) neuron. The input layer (denoted by P) is modeled as a group of excitatory neurons that spike based on the input environment state following a Poisson process. The sparse input-to-liquid connections are initialized such that each excitatory neuron in the liquid receives synaptic connections from approximately K random input neurons. This guarantees uniform excitation of the liquid-excitatory neurons by the external input spikes. The fixed input-to-liquid synaptic weights are chosen from a uniform distribution between 0 and α as shown in **Table 1**, where α is the maximum bound imposed on the weights. The liquid consists of excitatory neurons (denoted by E) and inhibitory neurons (denoted by I) recurrently connected in a sparse random manner as illustrated in **Figure 1**. The number of excitatory neurons is chosen to be $4\times$ the number of inhibitory neurons as observed in the cortical circuits (Wehr and Zador, 2003). We use the Leaky-Integrate-and-Fire (LIF) model (Dayan and Abbott, 2003) to mimic the dynamics of both excitatory and inhibitory spiking neurons as described by the following differential equations:

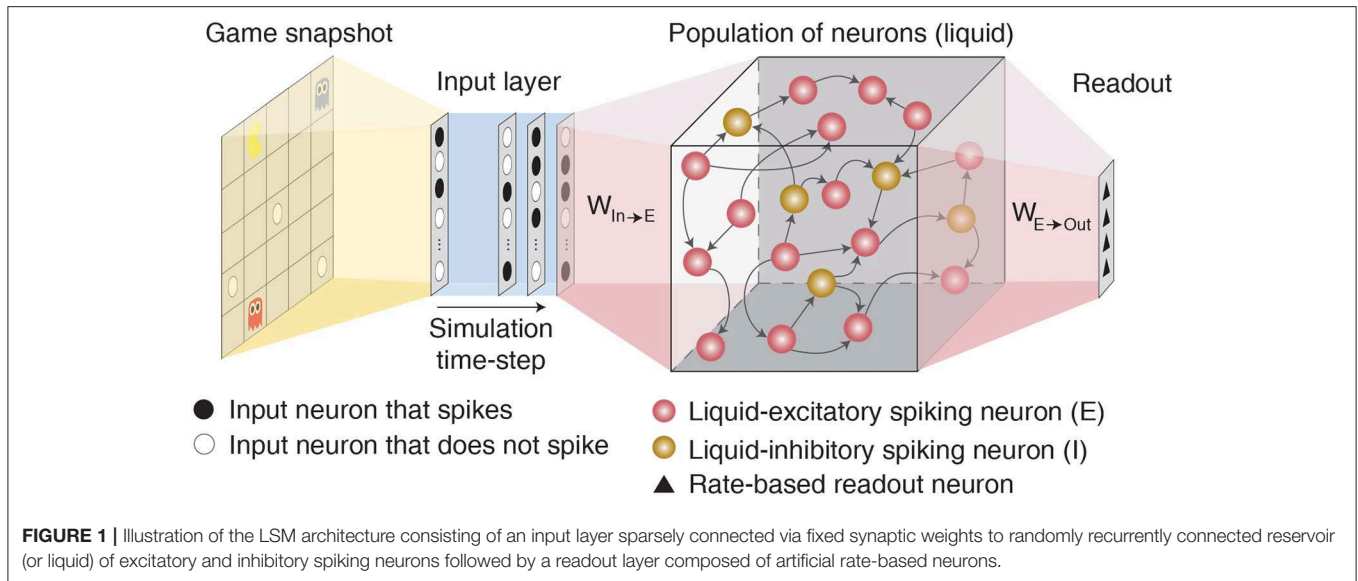


TABLE 1 | Synaptic weight initialization parameters for the fixed LSM connections for learning to balance cartpole, play Pacman, and play Atari game.

Connection type	Weight
INPUT-TO-LIQUID CONNECTIONS	
$P \rightarrow E$	[0, 0.6]
RECURRENT-LIQUID CONNECTIONS	
$E \rightarrow E$	[0, 0.05]
$E \rightarrow I$	[0, 0.25]
$I \rightarrow E$	[0, 0.3]
$I \rightarrow I$	[0, 0.01]

TABLE 2 | Leaky-Integrate-and-Fire (LIF) model parameters for the liquid neurons.

Parameter	Value
EXCITATORY AND INHIBITORY NEURONS	
V_{rest}	0
V_{reset}	0
V_{thres}	0.5
τ	20 ms
τ_{refrac}	1 ms
Δt (simulation time-step)	1 ms

$$\frac{dV_i}{dt} = \frac{V_{rest} - V_i}{\tau} + I_i(t) \quad (1)$$

$$I_i(t) = \sum_{l \in N_p} W_{li} \cdot \delta(t - t_l) + \sum_{j \in N_E} W_{ji} \cdot \delta(t - t_j) - \sum_{k \in N_I} W_{ki} \cdot \delta(t - t_k) \quad (2)$$

where V_i is the membrane potential of the i -th neuron in the liquid, V_{rest} is the resting potential to which V_i decays to, with time constant τ , in the absence of input current, and $I_i(t)$ is the instantaneous current projecting into the i -th neuron, and N_p , N_E , and N_I are the number of input, excitatory, and inhibitory neurons, respectively. The instantaneous current is a sum of three terms: current from input neurons, current from excitatory neurons, and current from inhibitory neurons. The first term integrates the sum of pre-synaptic spikes, denoted by $\delta(t - t_l)$ where t_l is the time instant of pre-spikes, with the corresponding synaptic weights (W_{li} in Equation 2). Likewise, the second (third) term integrates the sum of pre-synaptic spikes from the excitatory (inhibitory) neurons, denoted by $\delta(t - t_j)$ ($\delta(t - t_k)$), with the respective weights W_{ji} (W_{ki}) in Equation 2. The neuronal membrane potential is updated with the sum of the input, excitatory, and negative inhibitory currents as shown

in Equation 1. When the membrane potential reaches a certain threshold V_{thres} , the neuron fires an output spike. The membrane potential is thereafter reset to V_{reset} and the neuron is restrained from spiking for an ensuing refractory period by holding its membrane potential constant. The LIF model hyperparameters for the excitatory and inhibitory neurons are listed in **Table 2**.

There are four types of recurrent synaptic connections in the liquid, namely, $E \rightarrow E$, $E \rightarrow I$, $I \rightarrow E$, and $I \rightarrow I$. We express each connection in the form of a matrix that is initialized to be sparse and random, which causes the spiking dynamics of a particular neuron to be independent of most other neurons and maintains separability in the neuronal spiking activity. However, the degree of sparsity needs to be tuned to achieve rich network dynamics. We find that excessive sparsity (reduced connectivity) leads to weakened interaction between the liquid neurons and renders the liquid memoryless. On the contrary, lower sparsity (increased connectivity) results in chaotic spiking activity, which eliminates the separability in neuronal spiking activity. We initialize the connectivity matrices such that each excitatory neuron receives approximately C synaptic connections from inhibitory neurons, and vice versa. The hyperparameter C is tuned empirically as discussed in subsection 3.1 to avoid common chaotic spiking activity problems that occur when (1) excitatory neurons connect

to each other and form a loop that always leads to positive drift in membrane potential, and when (2) an excitatory neuron connects to itself and repeatedly gets excited from its activity. Specifically, for the first situation, we have non-zero elements in the connectivity matrix $E \rightarrow E$ (denoted by W_{EE}) only at locations where elements in the product of connectivity matrices $E \rightarrow I$ and $I \rightarrow E$ (denoted by W_{EI} and W_{IE} , respectively) are non-zero. This ensures that excitatory synaptic connections are created only for those neurons that also receive inhibitory synaptic connections, which mitigates the possibility of continuous positive drift in the respective membrane potentials. To circumvent the second situation, we force the diagonal elements of W_{EE} to be zero and eliminate the possibility of repeated self-excitation. Throughout this work, we create a recurrent connectivity matrix for liquid with m excitatory neurons and n inhibitory neurons by forming an $m \times n$ matrix whose values are randomly drawn from a uniform distribution between 0 and 1. Connection is formed between those pairs of neurons where the corresponding matrix entries are lesser than the target connection probability ($= C/m$). For illustration, consider a liquid with $m = 1,000$ excitatory and $n = 250$ inhibitory neurons. In order to create the $E \rightarrow I$ connectivity matrix such that each inhibitory neuron receives synaptic connection from a single excitatory neuron ($C = 1$), we first form a $1,000 \times 250$ random matrix whose values are drawn from a uniform distribution between 0 and 1. We then create a connection between those pairs of neurons where the matrix entries are lesser than 0.1% ($1/1,000$). Similar process is repeated for connection $I \rightarrow E$. We then initialize connection $E \rightarrow E$ based on the product of W_{EI} and W_{IE} . Similarly, the connectivity matrix for $I \rightarrow I$ (denoted by W_{II}) is initialized based on the product of W_{IE} and W_{EI} . The connection weights are initialized from a uniform distribution between 0 and β as shown in **Table 1** for different recurrent connectivity matrices, unless stated otherwise. Note that the weights of the synaptic connections from inhibitory neurons are greater than that for synaptic connections from excitatory neurons to account for the lower number of inhibitory neurons relative to excitatory neurons. Stronger inhibitory connection weights help ensure that every neuron receives similar amount of excitatory and inhibitory input currents, which improves the stability of the liquid as experimentally validated in subsection 3.1.

The liquid-excitatory neurons are fully-connected to artificial rate-based neurons in the readout layer for inference. The readout layer, which consists of as many output neurons as the number of actions for a given RL task, uses the average firing rate/activation of the excitatory neurons to predict the Q-value for every state-action pair. We translate the liquid spiking activity to average rate by accumulating the excitatory neuronal spikes over the time period for which the input (current environment state) is presented. We then normalize the spike counts with the maximum possible spike count over the LSM-simulation period, which is computed as the LSM-simulation period divided by the simulation time-step, to obtain the average firing rate of the excitatory neurons that are fed to the readout layer. Since the number of excitatory neurons is larger than the number of output neurons in the readout layer, we gradually reduce the dimension by introducing an additional fully-connected hidden

layer between the liquid and the output layer. We use ReLU non-linearity (Nair and Hinton, 2010) after the first hidden layer but none after the final output layer since the Q-values are unbounded and can assume positive or negative values. We train the synaptic weights constituting the fully-connected readout layer using the Q-learning based training methodology that is described in the following subsection 2.2.

2.2. Q-Learning Based LSM Training Methodology

At any time instant t in RL task, the agent receives the environment state s_t and picks action a_t from the set of all possible actions. After the environment receives the action a_t , it transitions to the next state based on the chosen action and feeds back an immediate reward r_{t+1} and the new environment state s_{t+1} . As mentioned in the beginning, the goal of the agent is to maximize the accumulated reward in the future, which is mathematically expressed as

$$R_t = \sum_{t=1}^{\infty} \gamma^t r_t \quad (3)$$

where $\gamma \in [0, 1]$ is the discount factor that determines the relative significance attributed to immediate and future reward. If γ is chosen to be 0, the agent maximizes only the immediate reward. However, as γ approaches unity, the agent learns to maximize the accumulated reward in the future. Q-learning (Watkins and Dayan, 1992) is a widely used RL algorithm that enables the agent to achieve this objective by computing the state-action value function (or commonly known as the Q-function), which is the expected future reward for a state-action pair that is specified by

$$Q_{\pi}(s, a) = E[R_t | s_t = s, a_t = a, \pi] \quad (4)$$

where $Q_{\pi}(s, a)$ measures the value of choosing an action a when in state s following a policy π . If the agent follows the optimal policy (denoted by π_*) such that $Q_{\pi_*}(s, a) = \max_{\pi} Q_{\pi}(s, a)$, the Q-function can be estimated recursively using the Bellman optimality equation that is described by

$$Q_{\pi_*}(s, a) = E[r_{t+1} + \gamma \max_{a_{t+1}} Q_{\pi_*}(s_{t+1}, a_{t+1}) | s, a] \quad (5)$$

where $Q_{\pi_*}(s, a)$ is the Q-value for choosing action a from state s following the optimal policy π_* , r_{t+1} is the immediate reward received from the environment, $Q_{\pi_*}(s_{t+1}, a_{t+1})$ is the Q-value for selecting action a_{t+1} from the next environment state s_{t+1} . Learning the Q-values for all possible state-action pairs is intractable for practical RL applications. Popular approaches approximate Q-function using deep convolutional neural networks (Lillicrap et al., 2015; Mnih et al., 2015, 2016; Silver et al., 2016).

In this work, we model the agent using an LSM, wherein the liquid-to-readout weights are trained to approximate the Q-function as described below. At any time instant t , we map the current environment state vector s_t to input neurons firing at a rate constrained between 0 and ϕ Hz over certain time

period (denoted by T_{LSM}) following a Poisson process. The maximum Poisson firing rate ϕ is tuned to ensure sufficient input spiking activity for a given RL task. We follow the method outlined in Heeger (2000) to generate the Poisson spike trains as explained below. For a particular input neuron in the state vector, we first compute the probability of generating a spike at every LSM-simulation time-step based on the corresponding Poisson firing rate. Note that the time-steps in the RL task are orthogonal to the time-steps used for the numerical simulation of the liquid. Specifically, in-between successive time-steps t and $t + 1$ in the RL task, the liquid is simulated for a time period of T_{LSM} with 1ms separation between consecutive LSM-simulation time-steps. The probability of producing a spike at any LSM-simulation time-step is obtained by scaling the corresponding firing rate by 1,000. We generate a random number drawn from a uniform distribution between 0 and 1, and produce a spike if the random number is lesser than the neuronal spiking probability. At every LSM-simulation time-step, we feed the spike map of the current environment state and record the spiking outputs of the liquid-excitatory neurons. We accumulate the excitatory neuronal spikes and normalize the individual neuronal spike counts with the maximum possible spike count over the LSM-simulation period to obtain the high-dimensional representation (activation) of the environment state as discussed in the previous subsection 2.1. Note that the liquid state variables, such as the neuronal membrane potentials are not reset between successive RL time-steps so that some information of the past environment representations are still retained. The capability of the liquid to retain decaying memory of the past representations enables it to perform temporal integration over different RL time-steps such that the high-dimensional representation provided by the liquid for the current environment state also depends on decaying memory of the past environment representations. However, it is important to note that appropriate initialization of the LSM (detailed in subsection 2.1) is necessary to obtain useful high-dimensional representation for efficient training of the liquid-to-readout weights as experimentally validated in section 3.

The high-dimensional liquid activations are fed to the readout layer that is trained using backpropagation to approximate the Q-function by minimizing the mean square error between the Q-values predicted by the readout layer and the target Q-values following (Mnih et al., 2015) as described by the following equations:

$$\theta_{t+1} = \theta_t + \eta (Y_t - Q(s_t, a_t | \theta_t)) \nabla_{\theta_t} Q(s_t, a_t | \theta_t) \quad (6)$$

$$Y_t = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1} | \theta_t) \quad (7)$$

where θ_{t+1} and θ_t are the updated and previous synaptic weights in the readout layer, respectively, η is learning rate, $Q(s_t, a_t | \theta_t)$ is vector representing the Q-values predicted by the readout layer for all possible actions given the current environment state s_t using the previous readout weights, $\nabla_{\theta_t} Q(s_t, a_t | \theta_t)$ is the gradient of the Q-values with respect to the readout weights, and Y_t is the vector containing the target Q-values that is obtained by feeding the next environment state s_{t+1} to the LSM while using

the previous readout weights. To encourage exploration during training, we follow ϵ -greedy policy (Watkins, 1989) for selecting the actions based on the Q-values predicted by the LSM. Based on ϵ -greedy policy, we select a random action with probability ϵ and the optimal action, i.e., the action pertaining to the highest Q-value with probability $(1 - \epsilon)$ during training. Initially, ϵ is set to a large value (closer to unity), thereby permitting the agent to pick a lot of random actions and effectively explore the environment. As training progresses, ϵ gradually decays to a small value, thereby allowing the agent to exploit its past experiences. During evaluation, we similarly follow ϵ -greedy policy albeit with much smaller ϵ so that there is a strong bias toward exploitation. Employing ϵ -greedy policy during evaluation also serves to mitigate the negative impact of over-fitting or under-fitting. In an effort to further improve stability during training and achieve better generalization performance, we use the experience replay technique proposed by Mnih et al. (2015). Based on experience replay, we store the experience discovered at each time-step (i.e., s_t , a_t , r_t , and s_{t+1}) in a large table and later train the LSM by sampling mini-batches of experiences in a random manner over multiple training epochs, leading to improved generalization performance. For all the experiments reported in this work, we use the RMSProp algorithm (Tieleman and Hinton, 2012) as the optimizer for error backpropagation with mini-batch size of 32. We adopt ϵ -greedy policy, wherein ϵ gradually decays from 1 to 0.001–0.1 over the first 10% of the training steps. Replay memory stores one million recently played frames, which are then used for mini-batch weight updates that are carried out after the initial 100 training steps. The simulation hyperparameters for Q-learning are summarized in Table 3.

3. EXPERIMENTAL RESULTS

We first present results motivating the importance of careful LSM initialization for obtaining rich high-dimensional state representation, which is necessary for efficient training of the

TABLE 3 | Q-learning simulation parameters.

Parameter	Value
Readout weights update frequency	Once every game-step
Warm up steps before training begins	100
Batch size for experience replay	32
Experience replay buffer size	1×10^6
Discount factor	0.95
Initial exploration probability during training	1
Final exploration probability during training (Cartpole)	1×10^{-3}
Final exploration probability during training (Pacman & Atari)	1×10^{-1}
Exploration probability during evaluation (Cartpole & Atari)	5×10^{-2}
Exploration probability during evaluation (Pacman)	0
Learning rate for RMSProp algorithm	2×10^{-4}
Term added to denominator for RMSProp algorithm	1×10^{-6}
Weight decay for RMSProp algorithm	0
Smoothing constant for RMSProp algorithm	0.99

liquid-to-readout weights. We then demonstrate the utility of the recurrent-liquid synaptic connections of careful LSM initialization using classic cartpole-balancing RL task (Sutton and Barto, 1998). We then validate the capability of appropriately initialized LSM, trained using the presented methodology, for solving complex RL tasks like Pacman (DeNero et al., 2010) and Atari games (Brockman et al., 2016).

3.1. LSM Hyperparameter Tuning

Initializing LSM with appropriate hyperparameters is an important step to construct a model that produces useful high-dimensional representations. Since the input-to-liquid and recurrent-liquid connectivity matrices of the LSM are fixed *a priori* during training, how these connections are initialized dictates the liquid dynamics. We choose the hyperparameters K (governing the input-to-liquid connectivity matrix) and C (governing the recurrent-liquid connectivity matrices) empirically based on three observations: (1) stable spiking activity of the liquid, (2) eigenvalue analysis of the recurrent connectivity matrices, and (3) development of liquid-excitatory neuron membrane potential.

Spiking activity of the liquid is said to be stable if every finite stream of inputs results in a finite period of response. Sustained activity indicates that small input noise can perturb the liquid state and lead to chaotic activity that is no longer dependent on the input stimuli. It is impractical to analyze the stability of the liquid for all possible input streams within a finite time. We investigate the liquid stability by feeding in random input stimuli and sampling the excitatory neuronal spike counts at regular time intervals over the LSM-simulation period for different values of K and C . We separately adjust these hyperparameters for each learning task using random representations of the environment based on the following experimental steps. We begin by first selecting the hyper-parameter K , which indicates the number of pre-synaptic inputs to each neuron in the liquid. K is initialized to a small number ($=1$ in our experiments) while C is set to zero. We gradually increase K until the liquid neurons are sufficiently excited to determine the K that leads to optimally sparse spiking activity. The same optimal value of K can then be used for liquid of any size since each neuron still receives similar degree of excitation from the inputs and spikes sufficiently. Using the optimal value of K , we increase C until the desired eigenvalue spectrum and spiking neuronal dynamics (with respect to the evolution of the membrane potential over time) are obtained as explained in the following paragraph.

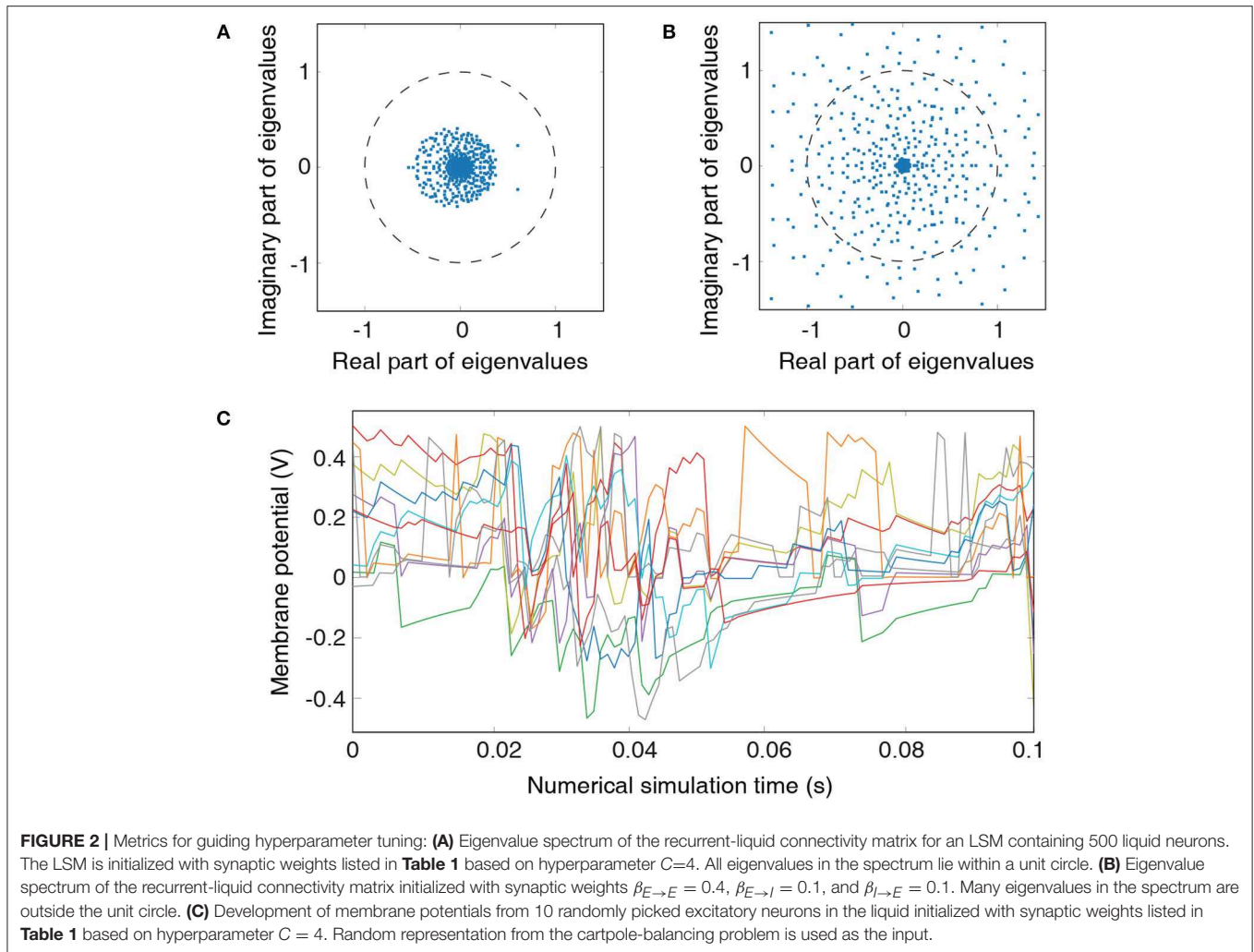
Analyzing the eigenvalue spectrum of the recurrent connectivity matrix is a common tool to assess the stability of the liquid. Each eigenvalue in the spectrum represents an individual mode of the liquid. Real part indicates decay rate of the mode while the imaginary part corresponds to the frequency of the mode (Rajan et al., 2010). Liquid spiking activity remains stable as long as all eigenvalues remain within the unit circle. However, this condition is not easily met for realistic recurrent-liquid connections with random synaptic weight initialization (Rajan and Abbott, 2006). We constrain the recurrent weights (hyperparameter β) such that each neuron receives balanced excitatory and inhibitory synaptic currents as

previously discussed in subsection 2.1. This results in eigenvalues that lie within the unit circle as illustrated in **Figure 2A**. In order to emphasize the importance of LSM initialization, we also show the eigenvalue spectrum of the recurrent-liquid connectivity matrix when the weights are not properly initialized as shown in **Figure 2B** where many eigenvalues are outside the unit circle. Finally, we also use the development of the excitatory neuronal membrane potential to guide hyperparameter tuning. The hyperparameters C and β are chosen to ensure that membrane potential exhibits balanced fluctuation as illustrated in **Figure 2C** that plots the membrane potential of 10 randomly picked neurons in the liquid. Note that these steps to find K and C are based on empirical observations. We chose values of K and C to be 3 and 4 for cartpole and Pacman experiment, respectively, which ensures stable liquid spiking activity while enabling the liquid to exhibit fading memory of the past inputs.

3.2. Learning to Balance a Cartpole

Cartpole-balancing is a classic control problem wherein the agent has to balance a pole attached to a wheeled cart that can move freely on a rail of certain length as shown in **Figure 3A**. The agent can exert a unit force on the cart either to the left or right side for balancing the pole and keeping the cart within the rail. The environment state is characterized by cart position, cart velocity, angle of the pole, and angular velocity of the pole, which are designated by the tuple $(\chi, \dot{\chi}, \varphi, \dot{\varphi})$. The environment returns a unit reward every time-step and concludes after 200 time-steps if the pole does not fall or the cart does not go out of the rail. Because the game is played for a finite time period, we constrain $(\chi, \dot{\chi}, \varphi, \dot{\varphi})$ to be within the range specified by $(\pm 2.5, \pm 0.5, \pm 0.28, \pm 0.88)$ for efficiently mapping the real-valued state inputs to spike trains feeding into the LSM. Each real-valued state input is mapped to 10 input neurons which have firing rates proportional to one-hot encoding of the input value representing 10 distinct levels within the corresponding range.

We model the agent using an LSM containing 150 liquid neurons, 32 hidden neurons in the fully-connected layer between the liquid and output layer, and two output neurons. The maximum firing rate for the input neurons representing the environment state is set to 100 Hz and each input is presented for 100 ms. The LSM is trained for 10^5 time-steps, which are equally divided into 100 training epochs containing 1,000 time-steps per epoch. After each epoch, the LSM is evaluated for 1,000 time-steps with the probability of choosing a random action ϵ set to 0.05. Note that the LSM is evaluated for 1,000 time-steps (multiple gameplays) even though single gameplay lasts a maximum of only 200 time-steps as mentioned in the previous paragraph. We use the accumulated reward averaged over multiple gameplays as the true indicator of the LSM (agent) performance to account for the randomness in action-selection as a result of the ϵ -greedy policy. We train the LSM initialized with 10 different random seeds and obtain median accumulated reward as shown in **Figure 3B**. Note that the maximum possible accumulated reward per gameplay is 200 since each gameplay lasts at most 200 time-steps. Increase in median accumulated reward over epochs indicates that the LSM learnt to balance the cartpole using the dynamically



evolving high-dimensional liquid states. The ability of the liquid to provide rich high-dimensional input representations can be attributed to the careful initialization of the connectivity matrices and weights (explained in subsection 2.1), which ensures balance between the excitatory and inhibitory currents to the liquid neurons and preserves fading memory of past liquid activity. However, the median accumulated reward after 100 training epochs saturates around 125 and does not reach the maximum value of 200. We hypothesize that the game score saturation comes from the quantized representation of the environment state, and demonstrate in the following experiment with Pacman that the LSM can learn optimally given a better state representation. Finally, in order to emphasize the importance of LSM initialization, we also show the median accumulated reward per training epoch for training in which the LSM is initialized to have few synaptic connections. **Figure 3C** indicates that the median accumulated reward is around 90 when the LSM initialization is suboptimal.

To visualize the learnt action-value function guiding action selection, we compare Q-values produced by the LSM during

evaluation in three different scenarios depicted in **Figure 3D**. Note that each Q-value represents how good is the corresponding action for a given environment state. In scenario 1 (see **Figure 3D-1**) that corresponds to the beginning of the gameplay wherein the pole is almost balanced, the value of both the actions are identical. This implies that either action (moving the cart left or right) will lead to a similar outcome. In scenario 2 (see **Figure 3D-2**) wherein the pole is unbalanced to the left side, the difference between the predicted Q values increases. Specifically, the Q value for applying a unit force on the right side of the cart is higher, which causes the cart to move to the left. Pushing the cart to the left in turn causes the pole to swing back right toward the balanced position. Similarly, in scenario 3 (see **Figure 3D-3**) wherein the pole is unbalanced to the right side, the Q value is higher for applying a unit force on the left side of the cart, which causes the cart to move right and enables the pole to swing left toward the balanced position. This visually demonstrates the ability of the LSM (agent) to successfully balance the pole by pushing the cart appropriately to the left or right based on the learnt Q values.

3.3. Learning to Balance a Cartpole Without Complete State Information

In this sub-section, we demonstrate the capability of the LSM to learn without complete state information, thereby validating its ability to perform temporal integration across different RL game steps enabled by the sparse random recurrent connections. Specifically, we modify the previous cartpole-balancing task such that the agent only receives the cart position and angle of the pole, designated by tuple (χ, φ) , as an input while the velocity information is ignored. The objective is to determine if the decaying memory of the past cart position and pole angle retained by the liquid, as a result of the recurrent-liquid connectivity, enables the LSM to make better decisions without the velocity information. We clip (χ, φ) to be within the range specified by $(\pm 2.5, \pm 0.28)$ similar to the previous experiment; however, each real-valued state input is mapped to only 1 input neuron whose firing rate is proportional to the normalized state value. A positive state input causes the corresponding neuron to fire unit positive spikes. On the other hand, if the state input is negative, the input neuron fires unit negative spikes at a rate proportional to the absolute value of the input as described in Sengupta et al. (2019). We initialize the input-to-liquid weights

from a uniform distribution between -0.4 and 0.4 to achieve balanced input excitation in the presence of both positive and negative spikes. Other connection weights are initialized from a uniform distribution as shown in **Table 4**.

We model the agent using an LSM with 150 liquid neurons followed by a fully-connected layer with 32 hidden neurons and a final output layer with two neurons, which is similar to the architecture used for the previous cartpole-balancing experiment. Additional feedback connections between excitatory neurons that have a large delay of 20 ms are introduced to

TABLE 4 | Synaptic weight initialization parameters for learning to balance cartpole without complete state information.

Connection type	Weight
RECURRENT-LIQUID CONNECTIONS	
$E \rightarrow E$ with 1 ms delay	[0, 0.4]
$E \rightarrow E$ with 20 ms delay	[0, 0.4]
$E \rightarrow I$	[0, 0.4]
$I \rightarrow E$	[0, 0.4]
$I \rightarrow I$	[0, 0.01]

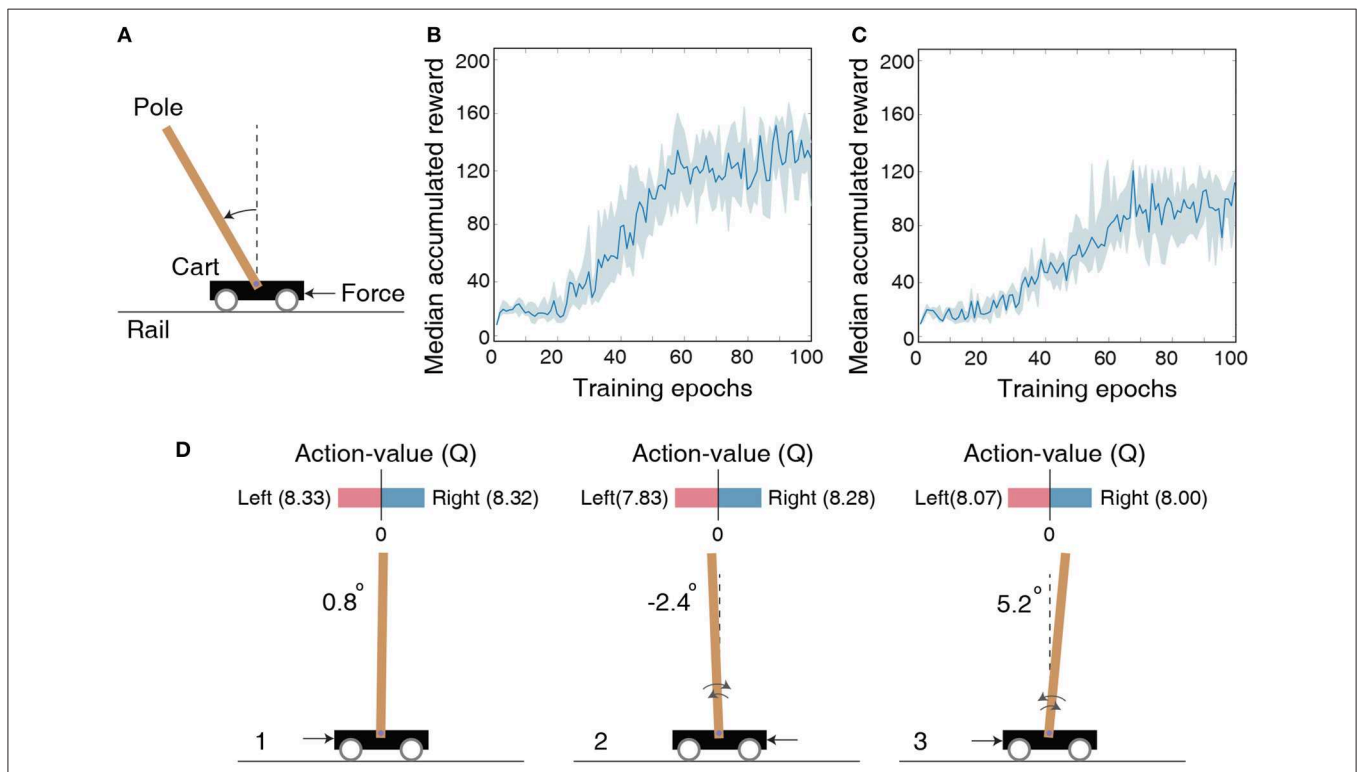


FIGURE 3 | **(A)** Illustration of the cartpole-balancing task wherein the agent has to balance a pole attached to a wheeled cart that moves freely on a rail of certain length. **(B)** The median accumulated reward per epoch provided by the LSM trained across 10 different random seeds for the cartpole-balancing task. Shaded region in the plot represents the 25-th to 75-th percentile of the accumulated reward over multiple random seeds. **(C)** The median accumulated reward per epoch from cartpole training across 10 different random seeds in which the LSM is initialized to have sparser connectivity between the liquid neurons compared to that used for the experiment in **(B)**. **(D)** Visualization of the learnt Q (action-value) function for the cartpole-balancing task at three different game-steps designated as 1, 2, and 3. Angle of the pole is written on the left side of each figure. Negative angle represents an unbalanced pole to the left and positive angle represents an unbalanced pole to the right. Black arrow corresponds to a unit force on the left or right side of the cart depending on which Q value is larger.

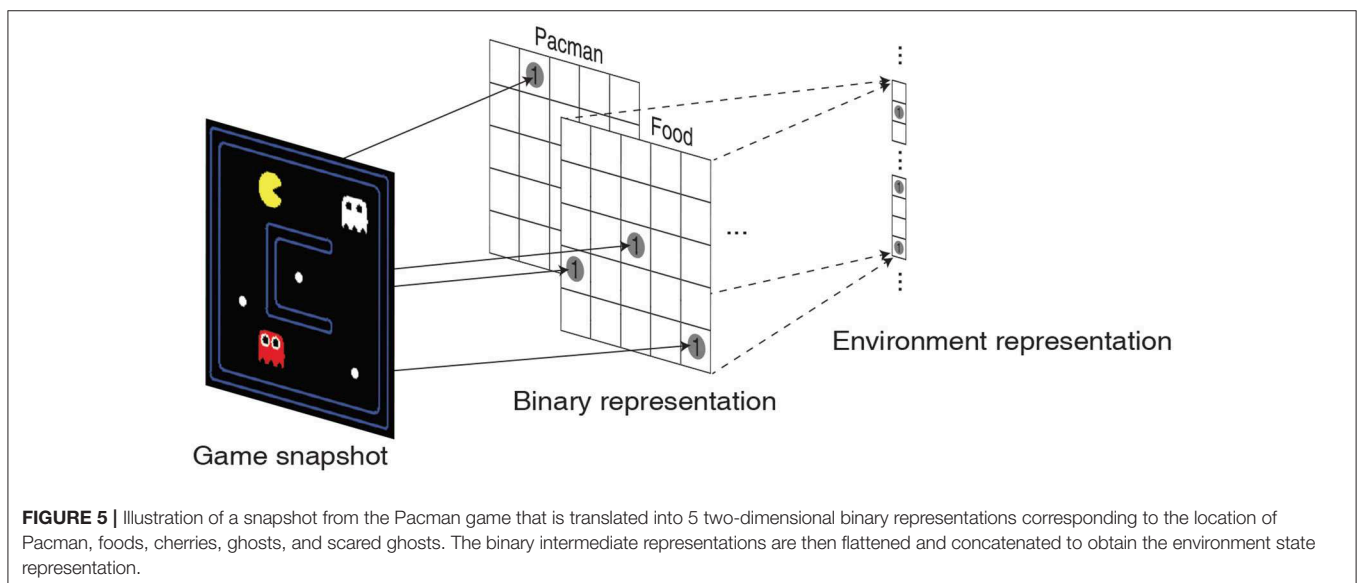
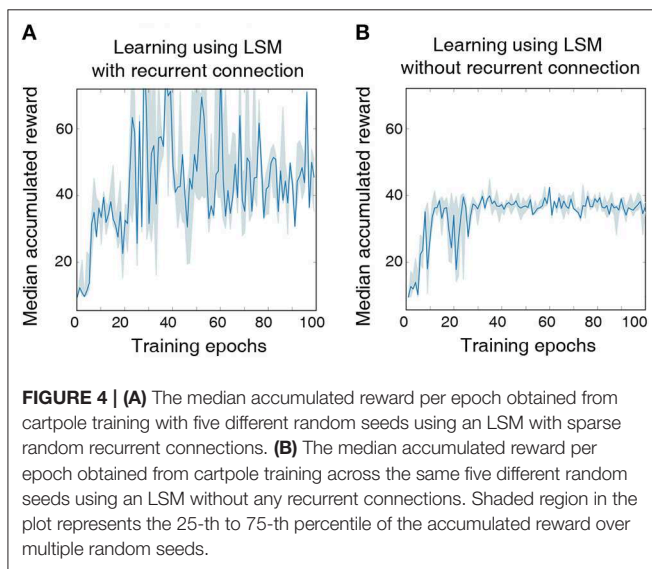
achieve long-term temporal integration over RL time-steps. In this experiment, we also reduced the LSM simulation time-steps to 20 ms from 100 ms used in the previous experiment to precisely validate the long-term temporal integration capability of the liquid. The LSM is trained for a total of 5×10^6 time-steps, which is sufficiently long to guarantee no further improvement in performance. Without complete state information, the LSM achieves best median accumulated reward of 70.93 over the last 10 epochs as illustrated in **Figure 4**, which is lower than that (125) attained with complete state information. However, the median accumulated reward of 70.93 achieved by the LSM based on incomplete state information is still higher than that (38.23) provided by the LSM without recurrent connections as shown in **Figure 4**. This indicates that the sparse recurrent connections provide useful information about the past input

since information about the cart velocity and angular velocity of the pole can be derived based on the current and past cart position and pole angle. We observe that LSM initialized based on some random seeds provide significantly better learning than others due to inherent stochasticity in the model, but we report the results based on the reward statistics obtained using runs from 5 different random seeds.

3.4. Learning to Play Pacman

In order to comprehensively validate the efficacy of the high-dimensional environment representations provided by the liquid, we train the LSM to play a game of Pacman (DeNero et al., 2010). The objective of the game is to control Pacman (yellow in color) to capture all the foods (represented by small white dots) in a grid without being eaten by the ghosts as illustrated in **Figure 5**. The ghosts always hunt the Pacman; however, cherry (represented by large white dots) make the ghosts temporarily scared of the Pacman and run away. The game environment returns unit reward whenever Pacman consumes food, cherry, or the scared ghost (white in color). The game environment also returns a unit reward and restarts when all foods are captured. We use the location of Pacman, food, cherry, ghost and scared ghost as the environment state representation. The location of each object is encoded as a two-dimensional binary array whose dimension matches with that of the Pacman grid as shown in **Figure 5**. The binary intermediate representations of all the objects are then concatenated and flattened into a single vector to be fed to the input layer of the LSM.

The LSM configurations and game settings used for Pacman experiments are summarized in **Table 5**, where each game setting has different degree of complexity with regards to the Pacman grid size and the number of foods, ghosts, and cherries. In the first experiment, we use a 7×7 grid with three foods for Pacman to capture and a single ghost to prevent it from achieving its objective. Thus, the maximum possible accumulated reward at the end of a successful game is 4. **Figure 6A** shows that the



median accumulated reward gradually increases with the number of training epochs and converges closer to the maximum possible reward, thereby validating the capability of the liquid to provide useful high-dimensional representation of the environment state necessary for efficient training of the readout weights using the presented methodology. Interestingly, in the second experiment using a larger 7×17 grid, we find that the median reward converges to 12, which is greater than the number of foods available in the grid. This indicates that the LSM does not only learn to capture all the foods; in addition, it also learns to capture the cherry and the scared ghosts, leading to further increase the accumulated reward since consuming the scared ghost results in a unit immediate reward. In the final experiment, we train the LSM to control Pacman in 17×19 grid with sparsely dispersed foods. We find that larger grid requires more exploration and training

steps for the agent to perform well and achieve the maximum possible reward, resulting in a learning curve that is less steep compared to that obtained for smaller grid sizes in the earlier experiments as shown in **Figure 6C**.

Finally, we plot the average of Q-values produced by the LSM as the Pacman navigates the grid to visualize the correspondence between the learnt Q-values and the environment state. As discussed in subsection 2.2, each Q-value produced by the LSM provides a measure of how good is a particular action for a give environment state. The Q-value averaged over the set of all possible actions (known as the state-value function) thus indicates the value of being in a certain state. **Figure 6D** illustrates the state-value function while playing the Pacman game in a 7×17 grid. The predicted state-value starts at a relatively high level because the foods are abundant in the grid and the ghosts are far away from the Pacman (see **Figure 6D-1**). The state-value gradually decreases as the Pacman navigates through the grid and gets closer to the ghosts. The predicted state-value then shoots up after the Pacman consumes cherry and makes the ghosts temporarily consumable (see **Figure 6D-2**), leading to potential additional reward. The predicted state-value drops after the ghosts are reborn (see **Figure 6D-3**). Finally, we observe a slight increase in the state-value toward the end of the game when the Pacman is closer to the last food after it consumes a cherry (see **Figure 6D**). It is interesting to note

TABLE 5 | LSM configuration and game settings for different Pacman experiments reported in this work.

Grid size	Ghost	Food	Cherry	Training steps	Liquid neurons	Hidden neurons
7×7	1	3	0	5×10^5	500	128
7×17	2	6	2	5×10^5	2,000	512
17×19	1	6	0	3×10^6	3,000	512

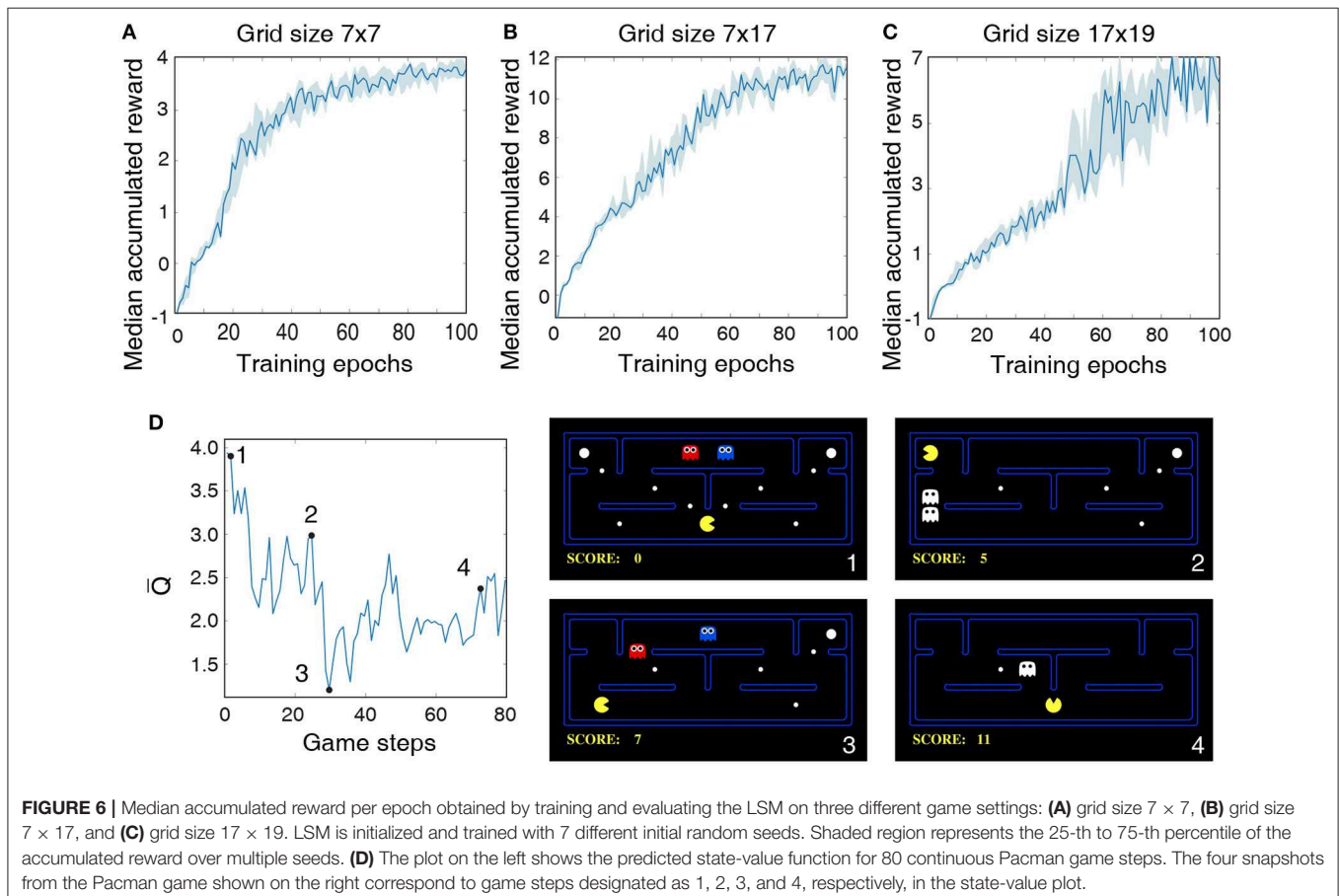


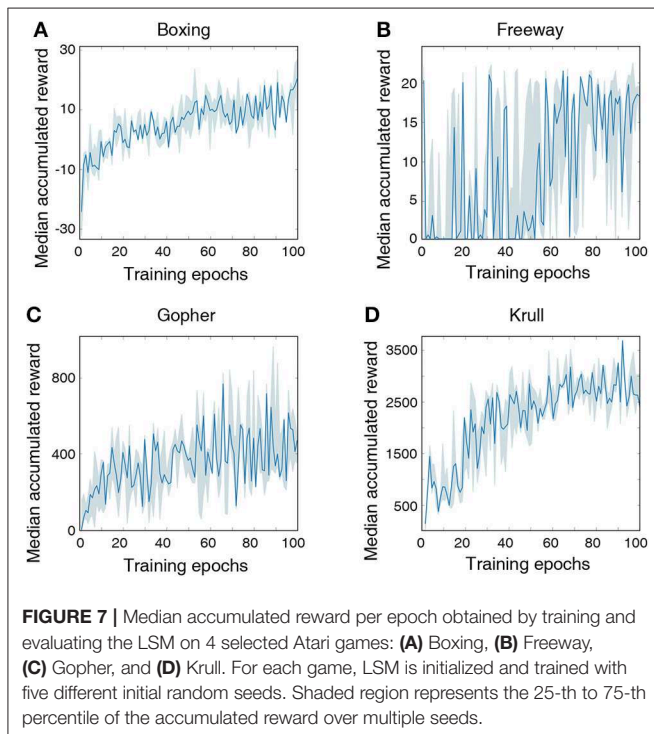
FIGURE 6 | Median accumulated reward per epoch obtained by training and evaluating the LSM on three different game settings: **(A)** grid size 7×7 , **(B)** grid size 7×17 , and **(C)** grid size 17×19 . LSM is initialized and trained with 7 different initial random seeds. Shaded region represents the 25-th to 75-th percentile of the accumulated reward over multiple seeds. **(D)** The plot on the left shows the predicted state-value function for 80 continuous Pacman game steps. The four snapshots from the Pacman game shown on the right correspond to game steps designated as 1, 2, 3, and 4, respectively, in the state-value plot.

that although the scenario in **Figure 6D-4** is similar to that in **Figure 6D-2**, the state-value is smaller since the expected accumulated reward at this step is at most 3 assuming that the Pacman can capture both the scared ghost and the last food. On the other hand, in the environment state shown in **Figure 6D-2**, the expected accumulated reward is >3 since 4 foods and 2 scared ghosts are available for the Pacman to capture.

3.5. Learning to Play Atari Games

Finally, we train the LSM using the presented methodology to play Atari games (Brockman et al., 2016), which are widely used to benchmark deep reinforcement learning networks. We arbitrarily select 4 games for evaluation, namely, Boxing, Gopher, Freeway, and Krull. We use the RAM of the Atari machine, which stores 128 bytes of information about an Atari game, as a representation of the environment (Brockman et al., 2016). During training, we modified the reward structure of the game by clipping all positive immediate rewards to +1 and all negative immediate rewards to -1 . However, we do not clip the immediate reward during testing and measure the actual accumulated reward following Mnih et al. (2015). For all selected Atari games, we model the agent using an LSM containing 500 liquid neurons and 128 hidden neurons. Number of output neurons varies for each game as the number of possible actions is different. The maximum Poisson firing rate for the input neurons is set to 100 Hz and each input is presented for 100ms. The LSM is trained for 5×10^3 steps.

Figure 7 illustrates that the LSM successfully learnt to play the Atari games without any prior knowledge of the rules,



leading to gradually increasing accumulated reward with the number of training epochs. We compare the median accumulated reward provided by the LSM to the average accumulated reward obtained from playing with random actions for 1×10^5 steps. Note that the median accumulated reward used for comparison is the highest reward achieved during the evaluation phase over the last 10 training epochs. **Table 6** shows that the median accumulated reward offered by the LSM is higher than the average accumulated reward obtained with random actions for all the four Atari games, which demonstrates the capability of the LSM to learn successful strategies in complex RL tasks. In fact, the median accumulated reward on Boxing and Krull reach the same level as human players reported in Mnih et al. (2015). However, we observe that the median accumulated reward on Freeway and Gopher are much lower than that of the human players. In order to identify the cause of poor learning, we trained all selected games using a deep learning network consisting of two convolutional and two fully-connected layers, and compared the median accumulated reward with that provided by the LSM. The architecture of the deep learning network used for different games is listed in **Table 7**. **Table 6** shows that the deep learning network trained with end-to-end error backpropagation using the Q-learning algorithm achieves better than human-level performance on Boxing and Krull while yielding lower rewards on Freeway and Gopher. Hence, the inferior performance of the LSM on Freeway and Gopher can be attributed to the nature (or complexity) of the respective games. However, the deep learning network yields superior performance compared to that provided by the LSM on all selected Atari games. We believe that the gap in the LSM performance compared to that obtained using the deep learning network stems from the inability of a randomly initialized LSM to extract complex input representations and game strategies. On the computation perspective, training a deep learning network incurs higher cost due to additional trainable parameters and the need for carrying out end-to-end error backpropagation. Simpler models like LSMs with lower training complexity offer a possible alternative for efficient training and inference in edge devices, such as self-flying drones that

TABLE 6 | Median accumulated reward for each game is chosen from the highest median accumulated reward over the last 10 training epochs across five different initial random seeds.

Game	Learning with LSM	Random actions	Human player	Learning with deep network
Boxing	20.2	0.8	4.3	68.2
Freeway	19.8	0.0	29.6	21.6
Gopher	611.1	279.3	2,321	1,443
Krull	3,686	1,590	2,395	4,672

Columns 1 and 4 report median accumulated rewards from learning with LSM and deep network, respectively. Average accumulated reward in column 2 is obtained from playing with random actions for 1×10^5 steps, which is a sufficiently large number for the average accumulated reward to be stable. Accumulated reward from human players reported in Mnih et al. (2015) is listed in column 3 for every game.

TABLE 7 | Convolutional deep learning network architecture used in Atari experiments.

Layer	Output features	Kernal size	Stride	Padding
One-dimensional convolutional	4	4	2	1
One-dimensional convolutional	16	4	2	1
Fully connected	128			
Fully connected	3 for Freeway			
	8 for Gopher			
	18 for Boxing and Krull			

operate under computational resource constraints and limited power budget.

4. DISCUSSION

LSM, an important class of biologically plausible recurrent SNNs, has thus far been primarily demonstrated for pattern (speech/image) recognition (Bellec et al., 2018; Srinivasan et al., 2018), gesture recognition (Chrol-Cannon and Jin, 2015; Panda and Srinivasa, 2018), and sequence generation tasks (Nicola and Clopath, 2017; Panda and Roy, 2017; Bellec et al., 2019) using standard datasets. To the best of our knowledge, our work is the first demonstration of LSMs, trained using Q-learning based methodology, for complex RL tasks like Pacman and Atari games commonly used to evaluate deep reinforcement learning networks. The benefits of the proposed LSM-based RL framework over the state-of-the-art deep learning models are 2-fold. First, LSM entails fewer trainable parameters as a result of using fixed input-to-liquid and recurrent-liquid synaptic connections. However, this requires careful initialization of the respective matrices for efficient training of the liquid-to-readout weights as experimentally validated in section 3. We note that the performance of LSMs could be further improved by training the recurrent weights using localized Spike Timing Dependent Plasticity (STDP) based learning rules (Bi and Poo, 1998; Song et al., 2000; Diehl and Cook, 2015) as demonstrated in Panda and Roy (2017) or biologically inspired variants of backpropagation-through-time (Bellec et al., 2018, 2019). Second, LSMs can be efficiently implemented on event-driven neuromorphic hardware like IBM *TrueNorth* (Merolla et al., 2014) or Intel *Loihi* (Davies et al., 2018), leading to potentially much improved energy efficiency while achieving comparable performance to deep learning models on the chosen benchmark tasks. Note that the readout layer in the presented LSM needs to be implemented outside the neuromorphic fabric since they are composed of artificial rate-based neurons that are typically not supported in neuromorphic hardware realizations. Alternatively, readout layer composed of spiking neurons could be used that can be trained using

spike-based error backpropagation algorithms (Lee et al., 2016, 2018; Panda and Roy, 2016; Jin et al., 2018; Wu et al., 2018; Bellec et al., 2019). Future works could also explore STDP-based reinforcement learning rules (Pfister et al., 2006; Farries and Fairhall, 2007; Florian, 2007; Legenstein et al., 2008) to render the training algorithm amenable for neuromorphic hardware implementations.

5. CONCLUSION

Liquid State Machine (LSM) is a bio-inspired recurrent spiking neural network composed of an input layer sparsely connected to a randomly interlinked liquid of spiking neurons for the real-time processing of spatio-temporal inputs. In this work, we proposed LSMs, trained using Q-learning based methodology, for solving complex Reinforcement Learning (RL) tasks like playing Pacman and Atari that have been hitherto benchmarked for deep reinforcement learning networks. We presented initialization strategies for the fixed input-to-liquid and recurrent-liquid connectivity matrices and weights to enable the liquid to produce useful high-dimensional representation of the environment based on the current and past input states necessary for efficient training of the liquid-to-readout weights. We demonstrated the significance of the sparse recurrent connections, which enables the liquid to retain decaying memory of the past input representations and perform temporal integration across RL time-steps, by training it using partial input state information that yielded higher accumulated reward than that provided by a liquid without recurrent connections. Our experiments on the Pacman game showed that the LSM learns the optimal strategies for different game settings and grid sizes. Our analyses on a subset of Atari games indicated that the LSM achieves comparable score to that reported for human players in existing works.

DATA AVAILABILITY

Publicly available datasets were analyzed in this study. This data can be found here: <https://github.com/openai/gym>.

AUTHOR CONTRIBUTIONS

GS and WP wrote the paper. WP performed the simulations. All authors helped with developing the concepts, conceiving the experiments, and writing the paper.

FUNDING

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, by the Semiconductor Research Corporation, the National Science Foundation, Intel Corporation, the DoD Vannevar Bush Fellowship, and by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

REFERENCES

- Amit, D. J. (1992). *Modeling Brain Function: The World of Attractor Neural Networks*. New York, NY: Cambridge University Press.
- Auer, P., Burgsteiner, H., and Maass, W. (2002). "Reducing communication for distributed learning in neural networks," in *International Conference on Artificial Neural Networks* (Madrid: Springer), 123–128.
- Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems 2018* (Quebec). Available online at: <http://papers.nips.cc/paper/7359-long-short-term-memory-and-learning-to-learn-in-networks-of-spiking-neurons>
- Bellec, G., Scherr, F., Hajek, E., Salaj, D., Legenstein, R., and Maass, W. (2019). Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv [Preprint] arXiv:1901.09049*. Available online at: <https://arxiv.org/abs/1901.09049>
- Berberich, N. (2017). *Implementation of a real-time liquid state machine on spinnaker for biomimetic robot control (Masterarbeit)*. Munich: TUM.
- Bi, G.-q., and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). Openai gym. *arXiv [Preprint] arXiv:1606.01540*. Available online at: <https://arxiv.org/abs/1606.01540>
- Chrol-Cannon, J., and Jin, Y. (2015). Learning structure of sensory inputs with synaptic plasticity leads to interference. *Front. Comput. Neurosci.* 9:103. doi: 10.3389/fncom.2015.00103
- Davies, M., Srinivasa, N., Lin, T.-H., China, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359
- Dayan, P., and Abbott, L. (2003). Theoretical neuroscience: computational and mathematical modeling of neural systems. *J. Cognit. Neurosci.* 15, 154–155. doi: 10.1162/089892903321107891
- DeNero, J., and Klein, D. (2010). "Teaching introductory artificial intelligence with pac-man," in *First AAAI Symposium on Educational Advances in Artificial Intelligence* (California), 1885–1889.
- Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099
- Douglas, R. J., Koch, C., Mahowald, M., Martin, K., and Suarez, H. H. (1995). Recurrent excitation in neocortical circuits. *Science* 269, 981–985. doi: 10.1126/science.7638624
- Farries, M. A., and Fairhall, A. L. (2007). Reinforcement learning with modulated spike timing-dependent synaptic plasticity. *J. Neurophysiol.* 98, 3648–3665. doi: 10.1152/jn.00364.2007
- Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Comput.* 19, 1468–1502. doi: 10.1162/neco.2007.19.6.1468
- Harris, K. D., and Mrsic-Flogel, T. D. (2013). Cortical connectivity and sensory coding. *Nature* 503:51. doi: 10.1038/nature12654
- Heeger, D. (2000). Poisson model of spike generation. *Handout* 5, 1–13. Available online at: <https://www.cns.nyu.edu/~david/handouts/poisson.pdf>
- Jiang, X., Shen, S., Cadwell, C. R., Berens, P., Sinz, F., Ecker, A. S., et al. (2015). Principles of connectivity among morphologically defined cell types in adult neocortex. *Science* 350:aac9462. doi: 10.1126/science.aac9462
- Jin, Y., Zhang, W., and Li, P. (2018). "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Montréal, QC: Curran Associates), 7005–7015.
- Joshi, P., and Maass, W. (2005). Movement generation with circuits of spiking neurons. *Neural Comput.* 17, 1715–1738. doi: 10.1162/0899766054026684
- Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435
- Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508
- Legenstein, R., Pecevski, D., and Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput. Biol.* 4:e1000180. doi: 10.1371/journal.pcbi.1000180
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations 2016* (San Juan, PR). Available online at: <https://iclr.cc/archive/www/doku.php%3Fid=iclr2016:main.html>
- Lukoševičius, M., and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.* 3, 127–149. doi: 10.1016/j.cosrev.2009.03.005
- Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955
- Maass, W., Natschläger, T., and Markram, H. (2003). "A model for real-time computation in generic neural microcircuits," in *Advances in Neural Information Processing Systems*, eds S. Becker, S. Thrun, and K. Obermayer (Vancouver, BC: Curran Associates), 229–236.
- Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning* (New York, NY), 1928–1937.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518:529. doi: 10.1038/nature14236
- Nair, V., and Hinton, G. E. (2010). "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Haifa), 807–814.
- Nicola, W., and Clopath, C. (2017). Supervised learning in spiking neural networks with force training. *Nat. Commun.* 8:2208. doi: 10.1038/s41467-017-01827-3
- Panda, P., and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 299–306.
- Panda, P., and Roy, K. (2017). Learning to generate sequences with combination of hebbian and non-hebbian plasticity in recurrent spiking neural networks. *Front. Neurosci.* 11:693. doi: 10.3389/fnins.2017.00693
- Panda, P., and Srinivasa, N. (2018). Learning to recognize actions from limited training examples using a recurrent spiking neural model. *Front. Neurosci.* 12:126. doi: 10.3389/fnins.2018.00126
- Pfister, J.-P., Toyozumi, T., Barber, D., and Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Comput.* 18, 1318–1348. doi: 10.1162/neco.2006.18.6.1318
- Rajan, K., and Abbott, L. (2006). Eigenvalue spectra of random matrices for neural networks. *Phys. Rev. Lett.* 97:188104. doi: 10.1103/PhysRevLett.97.188104
- Rajan, K., Abbott, L., and Sompolinsky, H. (2010). Stimulus-dependent suppression of chaos in recurrent neural networks. *Phys. Rev. E* 82:011903. doi: 10.1103/PhysRevE.82.011903
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323:533. doi: 10.1038/323533a0
- Savage, J. E. (1998). *Models of Computation*, Vol. 136. Reading, MA: Addison-Wesley.
- Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: Vgg and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484. doi: 10.1038/nature16961
- Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3:919. doi: 10.1038/78829
- Srinivasan, G., Panda, P., and Roy, K. (2018). Spilinc: spiking liquid-ensemble computing for unsupervised speech and image recognition. *Front. Neurosci.* 12:524. doi: 10.3389/fnins.2018.00524

- Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.
- Tieck, J. C. V., Pogančić, M. V., Kaiser, J., Roennau, A., Gewaltig, M.-O., and Dillmann, R. (2018). “Learning continuous muscle control for a multi-joint arm by extending proximal policy optimization with a liquid state machine,” in *International Conference on Artificial Neural Networks* (Rhodes: Springer), 211–221.
- Tieleman, T., and Hinton, G. (2012). Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.* 4, 26–31. Available online at: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- Verstraeten, D., Schrauwen, B., Stroobandt, D., and Van Campenhout, J. (2005). Isolated word recognition with the liquid state machine: a case study. *Inform. Process. Lett.* 95, 521–528. doi: 10.1016/j.ipl.2005.05.019
- Watkins, C. J., and Dayan, P. (1992). Q-learning. *Mach. Learn.* 8, 279–292. doi: 10.1023/A:1022676722315
- Watkins, C. J. C. H. (1989). *Learning from delayed rewards* (PhD thesis), King’s College, Cambridge, United Kingdom.
- Wehr, M., and Zador, A. M. (2003). Balanced inhibition underlies tuning and sharpens spike timing in auditory cortex. *Nature* 426:442. doi: 10.1038/nature02116
- Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Copyright © 2019 Ponghiran, Srinivasan and Roy. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.