IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 9, 2020, accepted May 30, 2020, date of publication June 10, 2020, date of current version June 25, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3001296*

# SPSNN: *n*th Order Sequence-Predicting Spiking Neural Network

**DOHUN KIM[1,2], VLADIMIR KORNIJCUK[3], CHEOL SEONG HWANG[1,2], AND DOO SEOK JEONG[3], (Member, IEEE)**

[1]Department of Material Science and Engineering, Seoul National University, Seoul 08826, South Korea
[2]Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, South Korea
[3]Division of Materials Science and Engineering, Hanyang University, Seoul 04763, South Korea

Corresponding author: Doo Seok Jeong (dooseokj@hanyang.ac.kr)

This work was supported by the National Research Foundation of Korea under Grant NRF-2019R1C1C1009810.

**ABSTRACT** We introduce a means of harnessing spiking neural networks (SNNs) with rich dynamics as a dynamic hypothesis to learn complex sequences. The proposed SNN is referred to as *n*th order sequence-predicting SNN (*n*-SPSNN), which is capable of single-step prediction and sequence-to-sequence prediction, i.e., associative recall. As a key to these capabilities, we propose a new learning algorithm, named the learning by backpropagating action potential (LbAP) algorithm, which features (i) postsynaptic event-driven learning, (ii) access to topological and temporal local data only, (iii) competition-induced weight normalization effect, and (iv) fast learning. Most importantly, the LbAP algorithm offers a unified learning framework over the entire SPSNN based on local data only. The learning capacity of the SPSNN is mainly dictated by the number of hidden neurons $h$; its prediction accuracy reaches its maximum value ($\sim$1) when the hidden neuron number $h$ is larger than twice training sequence length $l$, i.e., $h \geq 2l$. Another advantage is its high tolerance to errors in input encoding compared to the state-of-the-art sequence learning networks, namely long short-term memory (LSTM) and gated recurrent unit (GRU). Additionally, its efficiency in learning is approximately 100 times that of LSTM and GRU when measured in terms of the number of synaptic operations until successful training, which corresponds to multiply-accumulate operations for LSTM and GRU. This high efficiency arises from the higher learning rate of the SPSNN, which is attributed to the LbAP algorithm. The code is available on-line (https://github.com/galactico7/SPSNN).

**INDEX TERMS** Sequence-predicting spiking neural network, event-driven learning algorithm of locality, sequence learning, single-step prediction, associative recall.

## I. INTRODUCTION

Spiking neural network (SNN) is a dynamic hypothesis with diverse temporal kernels to express neuronal behaviors in response to synaptic transmission [1]–[3]. The central nervous system (CNS) is based on the SNN, and the SNN has therefore been analyzed theoretically to understand the working principles of the CNS. Apart from the SNN's physiological plausibility, its feasible applications to deep learning as a hypothesis have attracted considerable research attention from various fields [4]–[9]. The effort to realize an SNN using integrated circuits—which has continued over the last three decades—paves the way for the data- and energy-efficient acceleration of deep learning. This has been emerging as an important goal of neuromorphic engineering [5], [6]. In this case, the main challenge lies in the learning algorithm; a universal learning algorithm, similar to backpropagation algorithms for deep neural network (DNN), is still missing. There exist several methods to optimize synaptic weights in an SNN. They usually map the weights of backpropagation-trained DNNs onto SNNs [10], [11]. However, to leverage the SNN in neuromorphic hardware, the learning needs to be based on an event-driven algorithm of locality [5], [6], [12]. For instance, the event-driven random backpropagation (eRBP) algorithm satisfies this requirement considering that (i) the ad hoc update is driven by a presynaptic spike and (ii) only local variables are used to evaluate the change in weight [13]. Static-domain data, e.g., images, are suited to the eRBP algorithm for training SNNs.

Considering the rich dynamics of SNN, learning with dynamic-domain data perhaps harnesses the full capability of SNNs [3], [12]. Dynamic-domain data include time-series

The associate editor coordinating the review of this manuscript and approving it for publication was Arianna Dulizia.

data, which embody periodic discrete data points in a time domain. In a framework of deep learning, the recurrent neural network (RNN) and its variations, e.g., long short-term memory (LSTM) [14] and gated recurrent unit (GRU) [15], are known to have an excellent capability to learn time-series data. Unfortunately, there is a lack of both SNN architecture for learning time-series data as well as a learning algorithm for the architecture, which performs sequence-prediction tasks, e.g., single-step prediction and sequence-to-sequence prediction (also known as associative recall), with accuracy comparable to that of LSTM and GRU.

In this regard, we propose an SNN architecture for temporal sequence learning, named *n*th order sequence-predicting spiking neural network (*n*-SPSNN). The indispensable working memory for the prediction is realized using synaptic chains. To train the *n*-SPSNN, we propose an event-driven learning algorithm of locality, referred to as learning by backpropagation action potential (LbAP) algorithm. The LbAP algorithm was inspired by physiological observations of backpropagating action potential (bAP) boosts intervening in homosynaptic plasticity [16]. Note that the weight is only updated upon postsynaptic events in contrast to other event-driven algorithms such as spike timing-dependent plasticity (STDP) rule (updates upon both presynaptic and postsynaptic events) and eRBP algorithm (updates upon presynaptic events only). The locality allows the LbAP algorithm to be suitable for memory-efficient implementation in digital neuromorphic hardware, particularly, multi-core neuromorphic processors [6], [12], [17], [18].

The primary contributions of our work are as follows.

- We introduce a novel learning algorithm (LbAP algorithm) for SNNs, which is suitable for neuromorphic hardware.
- We introduce an SNN architecture for sequence predictions by deploying working memories.
- We identify the sequence-learning capability of the proposed architecture and learning algorithm using the Nottingham dataset and random sequences. The results highlight high efficiency in learning and excellent tolerance to errors in sequence encoding.
- The code for the LbAP algorithm and SNN implementation is available on-line (https://github.com/galactico7/SPSNN).

Section II (Related work) addresses several relevant previous works including associative recalls of spike sequences in recurrent SNNs and sequence learning in RNNs. Section III is dedicated to detailed explanations of the *n*-SPSNN (Section III.A), LbAP algorithm (Section III.B), and application of the LbAP algorithm to the *n*-SPSNN (Section III.C). Experimental results follow in Section IV. Section IV.A explains the single-step prediction and associative recall (sequence-to-sequence) capacity of the *n*-SPSNN with different hyper-parameter settings. The performance robustness of the *n*-SPSNN to variability in a sequence is addressed in Section IV.C. Section IV.D addresses the learning efficiency of *n*-SPSNN in terms of learning speed and number of synaptic operations (SynOps) until the completion of learning.

## II. RELATED WORK

Associative memory or recall has long been the subject of research interest in sequence learning, which is categorized as sequence-to-sequence learning. The Hopfield network is a seminal network to this end; the recurrent network is given the capability to memorize patterns over the neurons, and a piece of a pattern can activate the whole pattern [19]. However, the network still lacks dynamics, and hence, the sequence information of the pattern is ignored. By contrast, RNNs are dynamic hypotheses that can learn sequences and make single-step predictions. This capability is given by the feedback connection that bases the current prediction on the network activity at the previous time step [20]. However, training the network through time causes several critical issues such as vanishing gradient and exploding gradient problems [21]. Variations of RNN, e.g., LSTM [14] and GRU [15], cope with these issues due to a constant error-flow through internal units. A backpropagation algorithm is commonly used to train these networks. The good performance of these networks comes at the cost of large computational power [22]. In particular, their demands for computational power are highlighted by comparison with the *n*-SPSNN, which will be addressed in Section IV.D. Their low tolerance to errors in sequence encoding is another disadvantage.

The sequence-learning neural network proposed by Wang and Yuwono [23] employs short-term memory networks as sub-networks, which play a similar role to the working memory of the *n*-SPSNN in sequence learning. The short-term memory network considers memory decay with time so that a discount factor applies to the contributions of previous elements to the present element prediction. Each subsequence of previous elements is mapped onto a single detector in an injective manner, given strong lateral inhibition among the detectors, which is also similar to the proposed *n*-SPSNN architecture. However, the neural network proposed by Wang and Yuwono consists of binary neurons, i.e., McCulloch–Pitts neurons, and the inhomogeneous learning rule applying to the network fails to provide a unified framework of learning. The learning rule needs to access global data (all weights values) to normalize the weight under update, which differs from our LbAP algorithm. Unfortunately, the sequence-learning capacity of the network is unavailable.

SNNs with recurrent connections are considered as hypotheses for associative recall [24]–[26]. Unlike the associative recall in the above-mentioned class, the associative recall in SNNs concerns not only a sequence of spiking neurons but also their precise spike times. This is because SNNs are endowed with the capacity to learn dynamic-domain data given their rich dynamics. Pfister *et al.* considered an SNN with stochastic spiking neurons and proposed a method to train the SNN to lead the postsynaptic neurons to fire spikes at desired times [27]. This work was further extended to more complicated SNNs with visible and hidden neurons,

where the hidden neurons receive supervision signals and represent a sequence [24]. In this work, the objective function for weight optimization was defined as the difference in a distribution function between a desired and actual spike time. The proposed learning algorithm for the visible neurons is found to be equivalent to the voltage-based STDP rule [28] in support of the physiological fidelity of the proposed learning algorithm. However, a different learning algorithm is used for the hidden neurons, which fails to provide a unified learning framework. Unfortunately, systematic analyses on the performance (learning capacity, efficiency, error-tolerance, etc.) of the recurrent SNN are unavailable.

Gardner and Grüning modified the learning rule by Pfister *et al.* [27] to train an SNN of deterministic neurons, referred to as FILT [29]. In the FILT rule, the synaptic weight is adjusted to reduce the difference in spike filtering between a desired and actual spike train. In a similar framework, several learning algorithms have been proposed to produce a desired spike train, such as remote supervised method [25], chronotron [30], spike pattern association neuron [31], and precise-spike-driven synaptic plasticity [32]. These learning rules are capable of training a neuron to generate the desired spike train in response to the input spike pattern. However, success in associative recall using these learning rules has not been demonstrated.

An STDP rule in conjunction with heterosynaptic depression enables a recurrent SNN to form synaptic chains, each of which represents a sequence that is recalled associatively [26]. Such chains are formed at random; however, the number of chains tends to decrease with the strength of global inhibition. Additionally, the network can copy an applied sequential input during training and reproduce the input subsequently. However, a critical downside is the necessary access to global data, such as total synaptic weights (for the heterosynaptic depression) and activity state variables (for global inhibition), which is inconsistent with the attributes of an ideal learning rule embedded in neuromorphic hardware.

The hierarchical temporal memory (HTM) adopts and simplifies the physiological observation that, in a pyramidal cell, a delay in postsynaptic potential is proportional to the distance between the dendritic spine and soma [33], [34]. The HTM network consists of a set of columns; a combination of such columns represents an element in a sequence. A sequence is learned such that a synaptic chain (with the same length as the sequence) representing the sequence is formed in the network. Therefore, learning complex sequences is inefficient.

## III. SEQUENCE-PREDICTING SPIKING NEURAL NETWORK AND LEARNING ALGORITHM

In this section, we detail our sequence-predicting framework. We first introduce a sequence prediction principle and the *n*-SPSNN architecture in Section III.A. Next, we describe the LbAP algorithm, show how the LbAP algorithm captures the temporal distance in Section III.B. Finally, we apply the LbAP algorithm to the *n*-SPSNN for sequence prediction and associative recall, which is detailed in Section III.C.

### A. SEQUENCE PREDICTION PRINCIPLE AND NETWORK ARCHITECTURE

We define a sequence of $l$ elements as $(x_1, x_2, \ldots, x_l)$, where each element is chosen from a set of $m$ symbols $S$ $(=\{s_1, s_2, \ldots, s_m\})$, i.e., $x_i \in \{s_1, s_2, \ldots, s_m\}$. $x_i$ and $x_{i+1}$ are separated by $\Delta t_e$ in time. Each element is represented by an $m$-long one-hot vector. The $n$-SPSNN learning a sequence is illustrated in Fig. 1(a). The network is given $m$ parallel sub-networks, each with a synaptic chain of $n$ neurons. One sub-network is dedicated to one of $m$ symbols only. The parallel $m$ sub-networks are in full connection with a hidden layer loaded with $h$ neurons. A weight matrix for the feedforward connections is defined as $\boldsymbol{w}_1$ $(\boldsymbol{w}_1 \in \mathbb{R}^{h\times nm}; w_1[i,j] \in [0, w_{max1}])$. The hidden neurons are fully connected with $m$ output neurons in an output layer. Each output neuron represents each of $m$ elements. Weight matrix $\boldsymbol{w}_2$ $(\boldsymbol{w}_2 \in \mathbb{R}^{m\times h}; w_2[i,j] \in [0, w_{max2}])$ defines the feedforward connections. Note that full lateral inhibition applies to both hidden and output layers. This $n$-SPSNN network is expressed as $m$-$(n \times m)$-$h$-$m$, considering the number of neurons in each layer.

The element on a given time step in a sequence is encoded as a one-hot vector. Each element of the vector is subsequently applied to the input neuron of the corresponding sub-network such that "1" indicates the presence of an input spike, whereas "0" indicates no spike (see Fig. 1(a)). Each spike is relayed over the synaptic chain in each sub-network. We consider an axonal delay $(\Delta t_{ax1})$ between neighboring neurons in a synaptic chain of $n$ neurons. Assuming that $\Delta t_{ax1} = \Delta t_e$, a spike on a given time step hops to the next neuron in $\Delta t_e$, and simultaneously, the next element in the sequence arrives at the input neurons. Therefore, a spike representing a particular element on a given time step can stay in the sub-network over the synaptic chain for $(n-1)$ $\Delta t_{ax1}$, serving as a working memory. Unless otherwise stated, the equalities $\Delta t_{ax1} = \Delta t_e$ and $\Delta t_e = 100$ ms hold. Note that the $n$-SPSNN robustly predicts a sequence with random variations in $\Delta t_e$ (i.e., $\Delta t_{ax1} \neq \Delta t_e$), which will be addressed in Section IV.D.

Fig. 1(b) schematizes this process for a 4-SPSNN ($n = 4$) trained with an arbitrary sequence of (B, A, C, D, B, B, C) ($l = 7$; $m = 4$; $S = \{A, B, C, D\}$). Sub-nets 1, 2, 3, and 4 represent elements A, B, C, and D, respectively. The table for each sub-network in Fig. 1(b) indicates a neuronal activity vector on each time step such that a non-zero element and "0" denote the presence of a spike and no spike, respectively. For instance, $000A_1$ for Sub-net 1 at $t_2$ means that Neuron 1 (N1) fires a spike, while the rest of the neurons (N2, N3, and N4) are quiet. Here, $A_1$ indicates the element corresponding to the sub-network (A) and the neuron index (subscript). The $n$-SPSNN begins sequence prediction when $n$ preceding elements are available. Thus, the 4-SPSNN in Fig. 1(b) begins the prediction at $t_4$ based on the component-wise sum of the

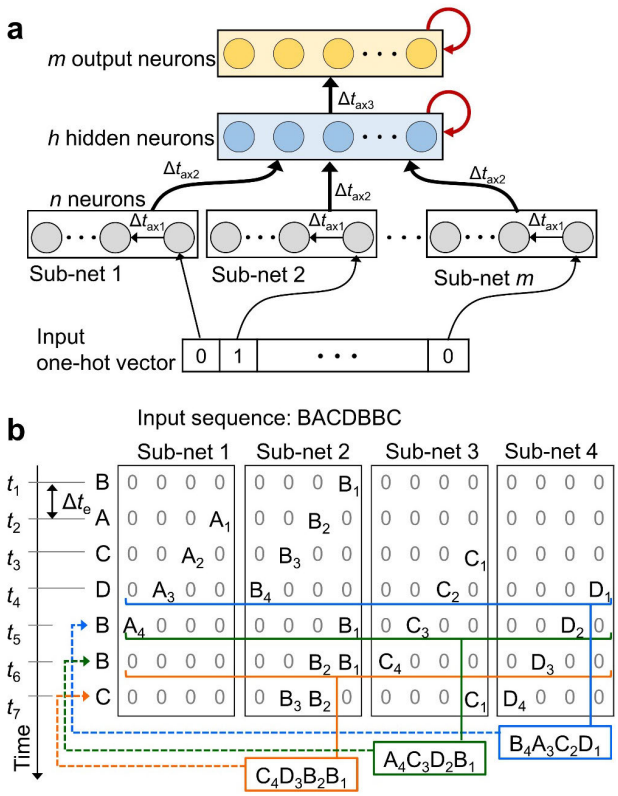



**FIGURE 1.** (a) Schematic of the *n*-SPSN ($m-(n \times m)-h-m$). The thick arrows indicate all-to-all connection, whereas the thin arrows indicate element-to-element connections. Lateral inhibition is indicated by red arrows. (b) Visualized single-step predictions for a sequence of (B, A, C, D, B, B, C).

four neuronal activity vectors ($B_4A_3C_2D_1$), meaning that N3 (Sub-net 1), N4 (Sub-net 2), N2 (Sub-net 3), and N1 (Sub-net 4) fire spikes at $t_4$ simultaneously. The prediction at $t_5$ is based on the vector ($A_4C_3D_2B_1$), meaning the simultaneous spiking of N4 (Sub-net 1), N1 (Sub-net 2), N3 (Sub-net 3), and N2 (Sub-net 4).

However, early prediction before seeing $n$ previous elements is made occasionally in real network operations, as will be addressed in Section IV.A.

Based on an $n$-long neuronal activity vector at $t_i$, the $n$-SPSNN should be able to predict the input element at $t_{i+1}$. For instance, $B_4A_3C_2D_1$ at $t_4$ in Fig. 1(b) outputs B as a predicted element at $t_5$ such that the output neuronal activity vector is 0100. The hidden layer in full connection with the parallel sub-networks and output layer associates the $n$-long neuronal activity vector with the desired output neuronal activity vector. Hidden neurons need to detect $n$ simultaneous spikes on a given time step and fire a spike accordingly, serving as a coincident detector. The spiking pattern of hidden neurons should be specific to a certain spatial pattern of spiking over the sub-networks. For instance, $B_4A_3C_2D_1$ at $t_4$ and $A_4C_3D_2B_1$ at $t_5$ should cause different spiking patterns to distinguish them. The lateral inhibition over the hidden neurons suppresses overlap between different spiking patterns. Eventually, the spiking pattern of hidden neurons activates the desired output neuron through the feedforward connections. The lateral inhibition over the output neurons ensures the clear separation of a desired output neuron from the others.

For associative recall (sequence-to-sequence prediction), the $n$-SPSNN for single-step prediction is modified to employ feedback connection from the output to the input layer. The single-step prediction in response to a set of $n$ preceding elements is fed into the input layer as an input. This feedback process continues onward until the end of the sequence.

The hidden and output neurons are expressed as a multi-compartment model in that the dendritic and somatic potentials are separately evaluated for each neuron. Both potentials are evaluated using a spike-response model (SRM) [2] (Appendix A). However, the dendrite is not allowed to fire spikes so that no refractory kernel applies to the dendritic potential evaluation. We consider axonal delays for the sub-networks-to-hidden layer and hidden layer-to-output layer feedforward connections, which are $\Delta t_{ax2}$ and $\Delta t_{ax3}$, respectively. They are fixed to 20 ms.

### B. LEARNING BY BACKPROPAGATING ACTION POTENTIAL (LbAP) ALGORITHM

To train the $n$-SPSNN, we propose a local learning algorithm called learning by backpropagating action potential (LbAP) algorithm. The LbAP algorithm was inspired by physiological observations of homosynaptic plasticity dictated by backpropagating action potentials (bAPs) [16], [35]. Upon spiking at a soma, the spike propagates to the dendritic spines; this is referred to as a bAP. The bAP amplitude decays over the dendrite. However, the initial amplitude is recovered if the dendritic potential exceeds a certain threshold, indicating a bAP boost. Otherwise, the amplitude keeps decaying out. The bAP, in turn, additively perturbs the dendritic potential, such that dendritic potential above the bAP-boost threshold undergoes a large increase in potential, whereas dendritic potential below the threshold undergoes a negligible increase in potential. The key to the direction of plasticity is the calcium influx such that a large (small) influx likely induces LTP (LTD) [36]–[38]. Importantly, the calcium influx tends to increase with membrane potential, and thus, a bAP boost likely induces LTP, while the failure of a bAP boost likely leads to LTD [16].

The LbAP algorithm simplifies the physiological observations as follows. First, the dendritic potential at the moment of bAP arrival directly determines the plasticity direction: if the potential is above the bAP-boost threshold, the synapse gains weight, and it loses weight otherwise. Second, a delay in backpropagation is ignored, so that the weights of all relevant synapses are updated simultaneously when the postsynaptic neuron fires a spike. Therefore, the LbAP algorithm is an event-driven local algorithm, ensuring incremental learning over a learning period. To be specific, the algorithm is a postsynaptic event-driven local algorithm because the weight is renewed only upon postsynaptic events in contrast to other event-driven algorithms such as STDP rule (presynaptic and

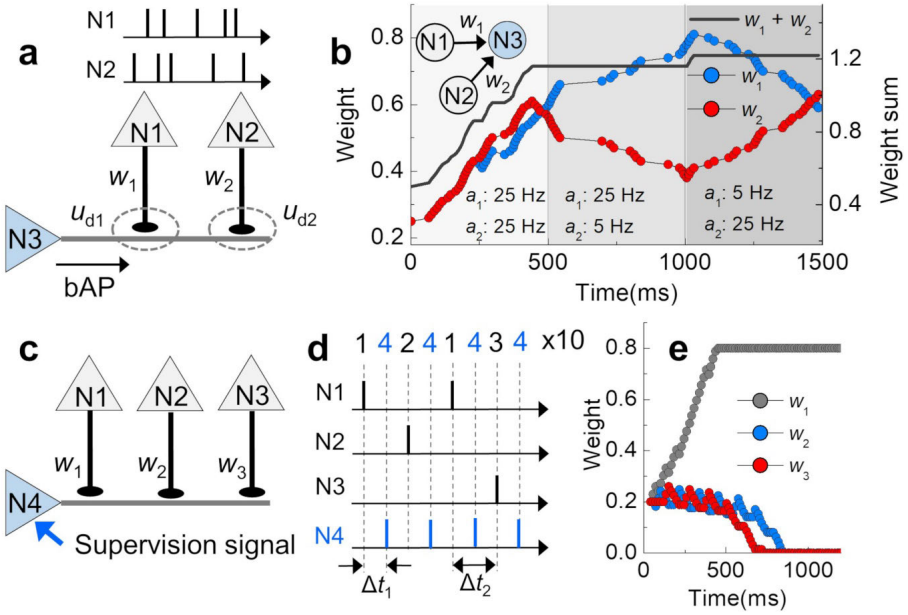



**FIGURE 2.** LbAP learning rule with rate and temporal codes. (a) Example of a neuronal configuration where the LbAP learning rule drives activity-dependent competition between the two presynaptic neurons N1 and N2, which share the same postsynaptic neuron N3. N1 and N2 emit Poisson spikes at activities of $a_1$ and $a_2$, respectively. (b) Evolution of weights $w_1$ and $w_2$ in response to $a_1$ and $a_2$, which differ for the three periods: 0–0.5 s, 0.5–1 s, and 1–1.5 s. The gray line denotes the sum of $w_1$ and $w_2$. (c) Example of a configuration where the LbAP learning rule drives competition between N1, N2, and N3, depending on the temporal correlation between a presynaptic and postsynaptic spike. A supervision signal applies to the postsynaptic neuron N4 to define the temporal correlation. We set $u_{d,th1}$, $u_{d,th2}$, and $w_{max}$ to 0, 1 mV, and 1, respectively. The firing threshold $u_{s,th}$ was fixed to 2.5 mV. (d) A spike sequence of (1, 4, 2, 4, 1, 4, 3, 4), where each number denotes the index of a neuron spiking at a given time. We set $\Delta t_1$ and $\Delta t_2$ to 20 ms and 30 ms, respectively. (e) Evolution of weights $w_1$, $w_2$, and $w_3$ in response to the spike sequence repeated ten times. Neuron N1 wins N2 and N3 because the unit sequence includes two 1-4 pairs, whereas both 2-4 and 3-4 pairs appear once. We set $u_{d,th1}$, $u_{d,th2}$, and $w_{max}$ to 50 $\mu$V, 1 mV, and 0.8, respectively. The neuronal parameters for both simulations are listed in Table 1.

postsynaptic event-driven algorithm) [39]–[41] and eRBP (presynaptic event-driven algorithm) [42].

The following equation describes the LbAP algorithm:

$$\Delta w = \left\{\alpha H\left(u_{\mathrm{d}} - u_{\mathrm{d,th}}\right) - \beta\Theta\left(u_{\mathrm{d}}\right)\right\}\delta\left(t - t_{\mathrm{post}}\right), \quad (1)$$

where $u_{\mathrm{d}}$, $u_{\mathrm{d,th}}$, and $H$ denote the dendritic potential at a given time, threshold for a bAP boost, and Heaviside step function, respectively. The LTP rate is determined by a positive constant $\alpha$. LTD is facilitated by a boxcar function $\Theta$ with $u_{\mathrm{d,th2}}$ and $u_{\mathrm{d,th1}}$ ($< u_{\mathrm{d,th2}}$):

$$\Theta = \begin{cases} 1 & \text{if } u_{\mathrm{d,th1}} < u_{\mathrm{d}} < u_{\mathrm{d,th2}}. \\ 0 & \text{otherwise.} \end{cases}$$

The parameter $u_{\mathrm{d,th1}}$ denotes a threshold for LTD. The LTD rate is determined by a positive constant $\beta$. The term $\delta\left(t - t_{\mathrm{post}}\right)$ ensures a postsynaptic event-driven weight update, where $t_{\mathrm{post}}$ refers to a postsynaptic event time. We employ weight boundaries (0 and $w_{\max}$) to avoid unlimited growth of weight and switch to inhibitory synapses. The LbAP algorithm is paraphrased in pseudocode, as follows:

**function** LbAP
  **for** j ∈ {postsynaptic spike} **do**
    **if** $u_{\mathrm{d}} > u_{\mathrm{d,th2}}$ **then** $w_{ji} \leftarrow w_{ji} + \alpha$
    **else if** $u_{\mathrm{d,th1}} < u_{\mathrm{d}} < u_{\mathrm{d,th2}}$ **then** $w_{ji} \leftarrow w_{ji} - \beta$
    **end if**
  **end for**
**end function**.

Notably, rate-based and spike (event)-based learning schemes merge in a unified framework based on the LbAP algorithm. Regarding rate-based learning, consider two independent presynaptic neurons (N1 and N2) firing Poisson spikes at $a_1$ and $a_2$ and a postsynaptic neuron (N3) firing to the presynaptic Poisson spikes (see Fig. 2(a)). Applying the LbAP algorithm to the two synapses results in rate-dependent changes in weights ($w_1$ and $w_2$) with $a_1$ and $a_2$ in an unsupervised manner, as plotted in Fig. 2(b). The first period ($a_1 = a_2 = 25$ Hz) explains a simultaneous increase in $w_1$ and $w_2$ due to the equally high firing rates. However, the different rates in the second period ($a_1 = 25$ Hz; $a_2 = 5$ Hz) bifurcate $w_1$ and $w_2$ such that N1 with the higher rate gains weight while N2 loses weight. Alternating the rates ($a_1 = 5$ Hz; $a_2 = 25$ Hz) in the third period reverses the directions of the weight changes. Note that this learning condition recalls the monocular deprivation experiment that backs the seminal Bienenstock–Cooper–Munro (BCM) rule [43], [44].

The result highlights rate-dependent learning (the higher the firing rate of a neuron, the more likely that the synapse with a postsynaptic neuron strengthens) in agreement with the Hebbian learning. However, unlike the basic Hebb's rule, competition between the two presynaptic neurons is induced even without explicit weight normalization, as identified by the constant sum of the weights in the second and third periods in Fig. 2(b). This feature highlights the key advantage of the LbAP algorithm, which enables weight normalization without access to global data unlike other normalization algorithms, e.g., heterosynaptic depression [26], Oja rule [45], and subtractive normalization [1].

The LbAP algorithm captures the temporal configuration of individual presynaptic and postsynaptic spikes. A presynaptic spike closely preceding a postsynaptic spike likely boosts a bAP at the dendritic spine, yielding LTP. Additionally, using the LbAP algorithm, a pair of presynaptic and postsynaptic neurons that most frequently fire spikes in close succession (a presynaptic spike preceding a postsynaptic spike) is distinguished from the other pairs. For instance, consider a toy network of three presynaptic neurons (N1–N3) and a postsynaptic neuron (N4) in Fig. 2c. One spike at a time is elicited from one of the three presynaptic neurons following a given sequence (1, 4, 2, 4, 1, 4, 3, 4) repeated 10 times (Fig. 2(d)). Events from N1, N2, N3, and N4 are denoted by 1, 2, 3, and 4, respectively. A supervision signal (external current) is applied to N4 to manipulate the temporal configuration of pre and postsynaptic spikes. In the sequence $(1, 4, 2, 4, 1, 4, 3, 4)\times 10$, N1 is most frequently paired with N4 (20 times) so that the synapse between N1 and N4 gains weight, whereas the other synapses undergo LTD, as shown in Fig. 2(e). The temporal order of spikes (a presynaptic spike preceding a postsynaptic spike in close succession) likely indicates the causality between the presynaptic postsynaptic events because a cause should precede its effect. However, the opposite order likely undermines the causality. Therefore, this example identifies the LbAP algorithm as an identifier of statistical causality between individual spikes, highlighting its suitability for spike-based learning.

### C. TRAINING METHOD AND CAPABILITY EVALUATION IN DETAIL

The $n$-SPSNN ($m$-($n \times m$)-$h$-$m$) was trained for a single-step prediction, given the $n$ previous elements in a sequence. As training data, we employed $l$-long random sequence data ($x_1, x_2, \ldots, x_l$), where $x_i$ was randomly chosen from set $S$ ($= \{s_1, s_2, \ldots, s_m\}$) with equal probability. Note that $l$ and $m$ are measures of complexity in the training data. Each element in the sequence was sampled every $\Delta t_e$ and subsequently encoded as a one-hot vector. Responding to the "1" in the one-hot vector, the input neuron in the corresponding sub-network in Fig. 1 fires periodic spikes at $a_0$ (=50 Hz). A supervised learning framework was used to train the $n$-SPSNN as a whole; the actual element on the present time step was considered as the correct response to the $n$ previous elements. Accordingly, the weights $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ were ad hoc updated every time step. The correct element was encoded as a one-hot vector (supervision signal) and applied to the output layer in sync with the $n$th input element of the $n$ previous elements. The supervision signal was a train of periodic current pulses at $a_0$; each pulse sufficed to evoke a spike from the neuron. Thus, periodic spikes at $a_0$ were elicited from the output neuron, which drove the update of $\boldsymbol{w}_2$. Unsupervised learning trained the weight matrix $\boldsymbol{w}_1$ because a desired spiking pattern of hidden neurons was unknown unlike training the weight matrix $\boldsymbol{w}_2$. Nevertheless, both unsupervised and supervised learning were performed within a unified framework based on the LbAP algorithm. The weight matrix $\boldsymbol{w}_1$ was loaded with random values ($0 < w_{ij} < w_{\max 1}$) initially. To avoid unwanted preset connections to the output neurons, the weight matrix $\boldsymbol{w}_2$ was loaded with constant values (0.2). Note that the lateral inhibition weights for both hidden and output layers were invariant through learning. The $n$-SPSNN was trained with the same sequence data repeatedly until the saturation of single-step prediction accuracy. The parameters in Table 1 were used for the simulation results, unless otherwise stated.

**TABLE 1. Parameters for $n$-SPSNN.**

| Symbol | Explanation | Value |
|---|---|---|
| $u_{\text{sth}}$ | Threshold for spikes | 10 mV |
| $u_{\text{d,th1}}$ | Threshold for LTD | 0.05 mV |
| $u_{\text{d,th2}}$ | Threshold for a bAP boost | 1 mV |
| $u^s_{reset}$ | Maximum hyper-polarized potential | 10 mV |
| $u^s_r$ | Rest potential at soma | 0 |
| $t^s_s$ | Postsynaptic current time constant at the soma | 15 ms |
| $t^s_m$ | Postsynaptic potential time constant at the soma | 20 ms |
| $t^d_s$ | Postsynaptic current time constant at the dendrite | 15 ms |
| $t^d_m$ | Postsynaptic potential time constant at the dendrite | 20 ms |
| $\epsilon_0$ | Pre-exponential factor | 0.0243 |
| $\kappa_0$ | Pre-exponential factor | 0.162 |
| $I^{ext}$ | An externally injected current | 1 mA |
| $w_{\max 1}$ | Maximum weight for sub-networks-to-hidden layer | 0.25 |
| $w_{\max 2}$ | Maximum weight for hidden layer-to-output layer | 0.75 |
| $\alpha$ | Synaptic permanence increment | 0.03 |
| $\beta$ | Synaptic permanence decrement | 0.03 |
| $a_0$ | Input activity | 50 Hz |
| $\Delta t_{\text{ax1}}$ | Axonal delays in synaptic chain | 100 ms |
| $\Delta t_{\text{ax2}}$ | Axonal delays for the sub-networks-to-hidden layer | 20 ms |
| $\Delta t_{\text{ax3}}$ | Axonal delays for the hidden layer-to-output layer | 20 ms |

A single-step prediction result was determined from output neuronal spikes in a time step. Training generally hinders output spikes from multiple neurons in a given time step, and hence, the index of a single active neuron was encoded as a one-hot vector of a predicted element. Otherwise, the

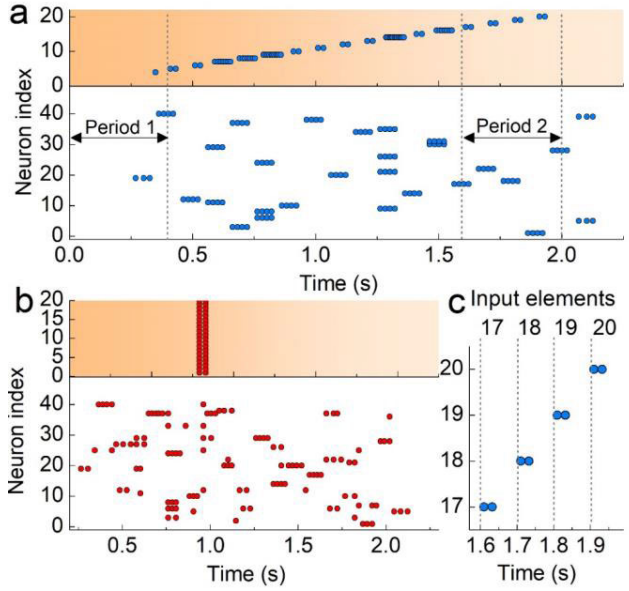



**FIGURE 3.** Spiking sequence before and after learning. (a) Spiking sequence of a 20-(4 × 20)-40-20 SPSNN for output neurons (upper panel) and hidden neurons (lower panel) in response to an input sequence of (1, 2, 3, . . . , 20), which is identical to training data. The spiking sequence of output neurons became associated with the training sequence in contrast to the untrained SPSNN shown in (b). (c) Spiking behavior of output neurons in Period 2 in (a), highlighting the capability of single-step predictions.

neuron index of the largest activity was considered to output a predicted element. The accuracy of single-step prediction was evaluated by applying the training sequence to the *n*-SPSNN without a supervision signal and by comparing the actual output with the correct output. For instance, if an *n*-SPSNN trained with an *l*-long sequence makes correct predictions *x* times, its single-step prediction accuracy is $x/(l - n)$, where *n* is in the denominator because the first *n* elements are ignored considering the working principle of the *n*-SPSNN.

## IV. RESULTS

### A. SEQUENCE-PREDICTION CAPACITY

Fig. 3 compares the spiking pattern of a fully trained 20-(4 × 20)-40-20 SPSNN with that of an untrained SPSNN. We used a sequence of (1, 2, 3, . . . , 20) ($l = 20$; $m = 20$; $S = \{1, 2, \ldots, 20\}$), where each element was sampled every $\Delta t_e$ (=100 ms). Fig. 3(a) shows the response of the fully trained SPSNN to the training sequence, identifying the capability of single-step predictions, unlike the untrained SPSNN in Fig. 3(b). The output spikes are delayed for one time-step because of the EPSC integration rate of the used neuron model. The delay is shown in Fig. 3(c), where the present input element and the output spikes responding to the element on the previous time step are present on the same time step. The first *n*th elements in the training sequence cannot be predicted correctly because the *n*-SPSNN needs *n* previous elements to predict the following element. Nevertheless, this 4-SPSNN can predict the fourth element based on the first three elements for this specific learning as in Period 1 shown in Fig. 3(a).

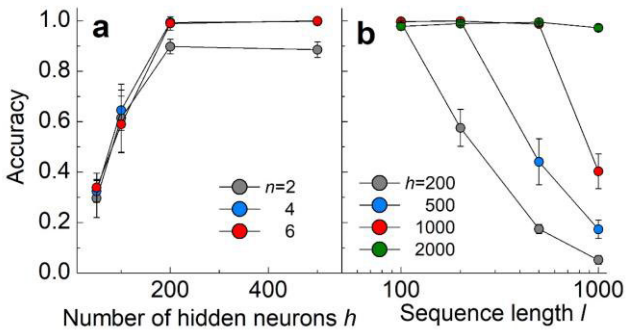



**FIGURE 4.** The single-step prediction accuracy. (a) Single-step prediction capability of a 20-($n$ × 20)-$h$-20 SPSNN with respect to the number of hidden neurons $h$ for three different $n$ values (2, 4, and 6). The SPSNN was trained using a random sequence ($l = 100$; $m = 20$). Each accuracy value was evaluated from ten trials; each trial includes a training period with a different random sequence and subsequent accuracy evaluation period. (b) Accuracy of a 20-(4 × 20)-$h$-20 SPSNN with varying training sequence length $l$ ($m = 20$) for different $h$ values.

**TABLE 2.** Single-step prediction Accuracy with respect to the number of hidden neurons $h$ for three different $n$ values (2, 4, and 6).

| Sequence length $l$ | Network | Accuracy |
|---|---|---|
| 100 | 20-(2×20)-50-20 | 0.295 ± 0.0758 |
| 100 | 20-(2×20)-100-20 | 0.614 ± 0.135 |
| 100 | 20-(2×20)-200-20 | 0.898 ± 0.0292 |
| 100 | 20-(2×20)-500-20 | 0.886 ± 0.0307 |
| 100 | 20-(4×20)-50-20 | 0.324 ± 0.0427 |
| 100 | 20-(4×20)-100-20 | 0.646 ± 0.0798 |
| 100 | 20-(4×20)-200-20 | 0.994 ± 0.00728 |
| 100 | 20-(4×20)-500-20 | 0.998 ± 0.00439 |
| 100 | 20-(6×20)-50-20 | 0.338 ± 0.0578 |
| 100 | 20-(6×20)-100-20 | 0.589 ± 0.112 |
| 100 | 20-(6×20)-200-20 | 0.989 ± 0.0270 |
| 100 | 20-(6×20)-500-20 | 1 |

To identify the sequence-prediction capacity of the proposed *n*-SPSNN, we analyzed the prediction accuracy of an $m$-($n \times m$)-$h$-$m$ SPSNN by varying the number of hidden neurons ($h$) and the length of a training sequence. Fig. 4(a) shows the measured single-step prediction accuracy with respect to $h$ for 2-, 4-, and 6-SPSNNs trained with random sequences ($l = 100$; $m = 20$). The data are provided in Table 2. The accuracy tends to increase with the number of hidden neurons until its saturation with approximately 200 hidden neurons. The accuracy for the 4-, and 6-SPSNNs reaches approximately 0.99, whereas the maximum accuracy for the 2-SPSNN is approximately 0.89. This result indicates that the number of hidden neurons is a key parameter for single-prediction capacity. Considering the negligible difference in maximum accuracy between 4- and 6-SPSNNs, $n$ is fixed to 4 hereafter. For the 2-SPSNN, the bAP-boost threshold of a hidden neuron $u^h_{\mathrm{d,th2}}$ was set to 0.5 mV (cf. a $u^h_{\mathrm{d,th2}}$ of 1 mV in Table 1) because two simultaneous spikes from the sub-networks fail to elevate the dendritic potential of hidden neurons above 1 mV.

The optimal number of hidden neurons offers the maximum accuracy (∼1) at minimal SynOps. Fig. 4 reveals the rule of thumb that $h$ ($\geq 2l$) leads to an accuracy of

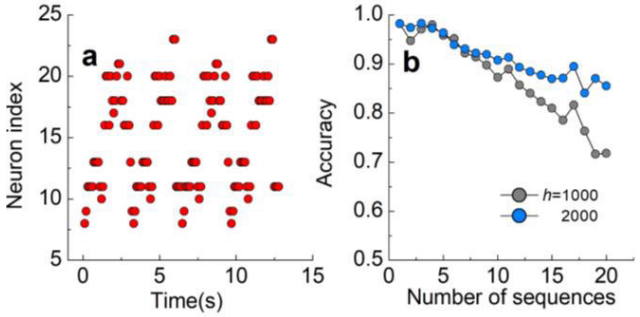



**FIGURE 5.** Single-step prediction accuracy on the Nottingham dataset. (a) Spiking sequence of input neurons in response to a one-hot encoded Nottingham tune. (b) Prediction-accuracy with the number of learned songs for a 26-(6 × 26)-*h*-26 SPSNN for different *h* values ($h = 1000$, and 2000).

**TABLE 3.** The single-step prediction accuracy with varying training sequence length $l$ ($m = 20$) for different $h$ values.

| Sequence length $l$ | Network | Accuracy |
|---|---|---|
| 100 | 20-(4×20)-200-20 | 0.994 ± 0.00728 |
| 100 | 20-(4×20)-500-20 | 0.998 ± 0.00439 |
| 100 | 20-(4×20)-1000-20 | 0.997 ± 0.00503 |
| 100 | 20-(4×20)-2000-20 | 0.978 ± 0.0143 |
| 200 | 20-(4×20)-200-20 | 0.576 ± 0.0733 |
| 200 | 20-(4×20)-500-20 | 0.999 ± 0.00215 |
| 200 | 20-(4×20)-1000-20 | 0.999 ± 0.00215 |
| 200 | 20-(4×20)-2000-20 | 0.989 ± 0.00627 |
| 500 | 20-(4×20)-200-20 | 0.174 ± 0.0184 |
| 500 | 20-(4×20)-500-20 | 0.441 ± 0.0913 |
| 500 | 20-(4×20)-1000-20 | 0.987 ± 0.00512 |
| 500 | 20-(4×20)-2000-20 | 0.995 ± 0.00343 |
| 1000 | 20-(4×20)-200-20 | 0.0519 ± 0.0168 |
| 1000 | 20-(4×20)-500-20 | 0.174 ± 0.0368 |
| 1000 | 20-(4×20)-1000-20 | 0.403 ± 0.0689 |
| 1000 | 20-(4×20)-2000-20 | 0.972 ± 0.0131 |

approximately unity; therefore, $h$ ($=2l$) appears to be the optimal number. This rule of thumb is underpinned by Fig. 4(b), which shows the prediction accuracy of 20-(4 × 20)-*h*-20 SPSNNs ($h = 200$, 500, 1000, and 2000) with respect to sequence length ($l = 100$, 200, 500, and 1000; $m = 20$). The rule that $h$ ($\geq 2l$) leads to the maximum accuracy ($\sim 1$) holds for the data in Fig. 4(b). The data in Fig. 4(b) are provided in Table 3.

We trained a 6-SPSNN on the Nottingham dataset (1200 British and American folk tunes). For each tune, we used its monophonic melody only, which was discretized as 26 notes according to pitch height. The note on a given time step was encoded as a one-hot vector and input into the input layer (26 neurons). The time bin size was set to 100 ms, so that each tune was subject to periodic sampling every $\Delta t_e$ ($=100$ ms). This preprocessing yielded training sequences ($62 \leq l \leq 192$; $m = 26$). The response of the 26 input neurons to a random tune is shown in Fig. 5(a).

To evaluate the sequence-prediction capacity, we trained a 26-(6 × 26)-*h*-26 SPSNN on the tunes (randomly sampled from the dataset and preprocessed as explained) by varying the number of sampled tunes. Fig. 5(b) shows the prediction accuracy of two SPSNNs ($h = 1000$ and 2000) with the number of sampled tunes. For a single tune, the accuracy of both cases for one sequence is above 0.98. However, it decreases with the number of the trained sequences. For 20 sequences, the accuracy reaches approximately 0.86 for $h = 2000$, whereas that for $h = 1000$ is approximately 0.72.

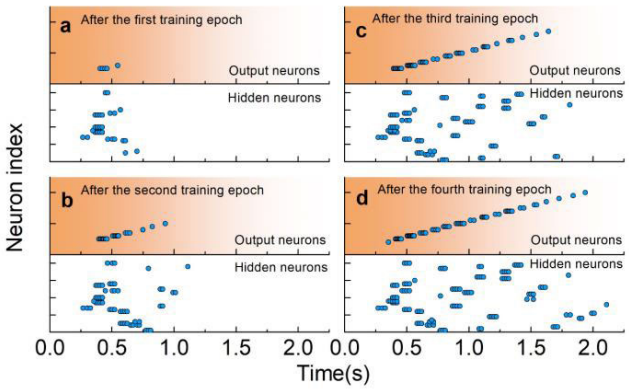



**FIGURE 6.** Associative recall. Associative recall capability of a 20-(4 × 20)-40-20 SPSNN after the (a) first, (b) second, (c) third, and (d) fourth training epoch. The training sequence was (1, 2, 3, …, 20) ($l = 20$; $m = 20$; $\Delta t_e = 100$ ms). For each case, the associative memory was triggered by the initial four elements (1, 2, 3, 4) of the total sequence.

### B. ASSOCIATIVE RECALL (SEQUENCE-TO-SEQUENCE PREDICTION)

The high accuracy of single-step prediction of the *n*-SPSNN offers the basis for associative recall (sequence-to-sequence prediction). For associative recall, the *n*-SPSNN architecture is modified such that feedback from the output to the input layer is employed to pass the prediction result on to the input. An advantage is that the output result (one-hot vector) can be applied to the input layer without additional encoding. To identify associative recall capability, we repeatedly trained a 20-(4 × 20)-40-20 SPSNN without feedback using the sequence (1, 2, 3, …, 20) ($l = 20$; $m = 20$). An associative recall test with the feedback followed every training epoch; associative recall was triggered by applying the first four elements of the sequence. Fig. 6 shows the progress of associative recall with the repetition of training. The 20-(4 × 20)-40-20 SPSNN eventually succeeds in recalling the whole sequence after repeating training four times.

### C. ROBUSTNESS OF LEARNING AND INFERENCE TO VARIABILITY IN SEQUENCE

Considering that real-world sequences include many imperfections, e.g., typo and noise, sequence-learning hypotheses need to make correct predictions despite the presence of imperfections. In this regard, the robustness of the *n*-SPSNN to errors in input encoding was examined. A 4-SPSNN [20-(4 × 20)-200-20] was trained using a random sequence ($l = 100$; $m = 20$), and its single-step prediction accuracy was measured with a test sequence that is identical to the training sequence but with a few different elements from the

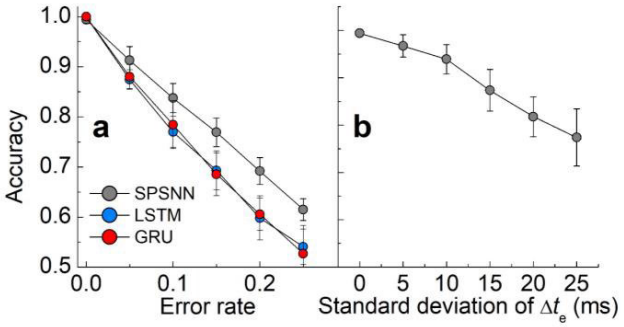



**FIGURE 7.** Tolerance of the SPSNN to errors in input sequences. (a) Robustness of single-step prediction for a 20-(4 × 20)-200-20 SPSNN to variability in elements in an input sequence ($l = 100$; $m = 20$) in comparison with LSTM and GRU. (b) Degradation of prediction accuracy for the same SPSNN with respect to variability in input sampling period ($\Delta t_e$).

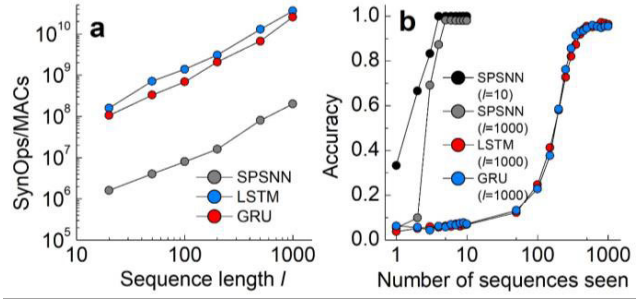



**FIGURE 8.** Efficiency in learning. (a) Number of SynOps for a 20-(4 × 20)-2000-20 SPSNN until a prediction accuracy of 0.97 with respect to sequence length ($20 \le l \le 1000$; $m = 20$). LSTM and GRU are compared with the SPSNN in terms of the number MAC operations required to reach the same prediction accuracy (0.97). (b) Single-step prediction accuracy evolution for a 20-(4 × 20)-2000-20 SPSNN, LSTM, and GRU with the number of training iterations. They were trained using random sequences ($l = 1000$; $m = 20$). For comparison, the same data for a 20-(4 × 20)-20-20 SPSNN trained using a random sequence ($l = 10$; $m = 20$) are co-plotted.

training sequence. They were chosen randomly. The different elements indicate errors in input encoding; their number $x$ defines the error rate as $x/l$. We evaluated the average prediction accuracy for a given error rate in the range 0–0.25 on 20 trials. Fig. 7(a) shows a linear decrease in accuracy with error rate, reaching approximately 0.62 at the maximum error rate (0.25). The results are compared with the error-tolerance of an LSTM and GRU, which are state-of-the-art sequence learning hypotheses. The LSTM and GRU used for this comparative study are elaborated in Appendix B. Similar to the 4-SPSNN, the LSTM and GRU undergo the degradation of prediction accuracy with error rate. However, their degradation rates are faster than that of the 4-SPSNN, insomuch as the accuracy for the LSTM and GRU reaches approximately 0.54 and 0.53, respectively, with an error rate of 0.25 (Fig. 7(a)). This comparison ensures a large tolerance of encoding error for the $n$-SPSNN trained with the LbAP algorithm compared to the state-of-the-art sequence learning hypotheses.

Prediction-robustness to variability in input-encoding delay is the key to the application to asynchronous neuromorphic hardware. To identify this robustness, a 20-(4 × 20)-200-20 SPSNN was trained using a sequence ($l = 100$; $m = 20$) with constant $\Delta t_e$ (=100 ms), and its single-step prediction accuracy was investigated with the same sequence but with randomly varying $\Delta t_e$ over the sequence. The delay in input-encoding $\Delta t_e^{'}$ was sampled from a Gaussian distribution function, which is centered at $\Delta t_e$ (=100 ms) with a standard deviation of $\sigma$, i.e., $\Delta t_e^{'} \sim N(\Delta t_e, \sigma)$. The delay was sampled for every interval over the test sequence. The standard deviation $\sigma$ is a measure of the variability in input-encoding delay. The measured prediction accuracy with the standard deviation is shown in Fig. 7(b). The accuracy tends to decrease with the standard deviation because the difference in input-encoding delay between the training and test sequences becomes larger with the standard deviation. Nevertheless, an accuracy of approximately 0.77 is maintained even with a standard deviation of 25 ms (25% of the center value). All data in Fig. 7 are provided in Table 4.

### D. LEARNING EFFICIENCY

Energy-efficient learning is an important attribute of a learning algorithm embedded in neuromorphic hardware [5], [12]. In this regard, a high learning rate is beneficial to energy-efficient learning, reducing the number of operations that significantly consume power. The SynOps is such an operation, which indicates a single update on a neuronal membrane potential upon an event. Therefore, the number of SynOps required for successful learning is a direct measure of energy-efficiency in learning. This quantity was evaluated for a 20-(4×20)-2000-20 SPSNN learning sequences of different lengths ($20 \le l \le 1000$; $m = 20$). Success in learning was defined by prediction accuracy above 0.97, and hence, the iterative training terminated when an accuracy of 0.97 was reached. The results are plotted in Fig. 8(a). The number of SynOps increases with the sequence length because a longer training sequence needs more ad hoc updates over the whole sequence, which inevitably increases SynOps.

We compared the required number of SynOps for successful learning with the required number of multiply-accumulate (MAC) operations for an LSTM and GRU. As for the SPSNN, both LSTM and GRU were trained using sequences of different lengths ($20 \le l \le 1000$; $m = 20$), and the training terminated when the single-step prediction accuracy reached 0.97. Details of the LSTM and GRU are provided in Appendix B. The evaluation results are co-plotted in Fig. 8(a), highlighting the efficient learning for the SPSNN with approximately two orders of magnitude fewer energy-consuming operations. The efficiency in learning is attributed to the fast learning rate facilitated by the LbAP algorithm, which is identified by monitoring the evolution of prediction accuracy with the number of training iterations (epochs). As shown in Fig. 8(b), a 20-(4 × 20)-2000-20 SPSNN trained using a random sequence ($l = 1000$; $m = 20$) achieves its maximum accuracy (∼0.98) in five training iterations, while the LSTM and GRU needs approximately two orders of magnitude more iterations. Moreover, the learning rate is

**TABLE 4. Tolerance to errors in input sequences and variability in input sampling period.**

| Error rate | Sequence length $l$ | Network: 20-(4×20)-200-20 SPSNN | Network: LSTM | Network: GRU |
|---|---|---|---|---|
| 0 | 100 | 0.994 ± 0.00728 | 1 | 1 |
| 0.05 | 100 | 0.913 ± 0.0275 | 0.874 ± 0.0196 | 0.880 ± 0.0240 |
| 0.1 | 100 | 0.838 ± 0.0294 | 0.770 ± 0.0306 | 0.784 ± 0.0479 |
| 0.15 | 100 | 0.769 ± 0.0287 | 0.693 ± 0.0383 | 0.685 ± 0.0427 |
| 0.2 | 100 | 0.692 ± 0.0267 | 0.598 ± 0.0441 | 0.606 ± 0.0323 |
| 0.25 | 100 | 0.615 ± 0.0218 | 0.541 ± 0.0353 | 0.527 ± 0.0562 |
| s.d. of $\Delta t_e$ (ms) | | | | |
| 0 | 100 | 0.994 ± 0.00728 | - | - |
| 5 | 100 | 0.967 ± 0.0235 | - | - |
| 10 | 100 | 0.939 ± 0.0307 | - | - |
| 15 | 100 | 0.873 ± 0.0440 | - | - |
| 20 | 100 | 0.818 ± 0.0421 | - | - |
| 25 | 100 | 0.774 ± 0.0605 | - | - |

independent of the network size (here, the number of hidden neurons) and sequence length $l$ as shown in the comparison with a 20-(4 × 20)-20-20 SPSNN trained using a random sequence ($l = 10$; $m = 20$) (Fig. 8(b)). The smaller network could learn the sequence with four iterative training steps, identifying a non-scaling learning rate with both network size and sequence length.

## V. CONCLUSION

We proposed an SNN architecture suitable for single-step prediction given $n$ previous elements in a training sequence, referred to as $n$-SPSNN. The key to the $n$th order sequence prediction is the sub-networks of synaptic chains that serve as working memory. This $n$-SPSNN architecture can learn sequences of various lengths using the LbAP algorithm as a unified learning framework. The LbAP algorithm is a postsynaptic event-driven learning algorithm of locality; each synapse involves a single local state variable (dendritic potential) so that memory usage is minimal. The competition between synapses with the same postsynaptic neuron is facilitated by the LbAP algorithm, which realizes effective weight normalization using local state variables only. The LbAP algorithm endows the $n$-SPSNN with the capabilities of single-step prediction and associative recall.

The sequence prediction robustness to variability in the test sequence element highlights its high tolerance to errors in input encoding, which is higher than the state-of-the-art sequence learning hypotheses LSTM and GRU. The $n$-SPSNN also offers the sequence prediction robustness to variability in intervals between neighboring elements, implying high tolerance to random changes in input-encoding delay. The efficiency in learning is another advantage of the $n$-SPSNN with the LbAP algorithm. The learning is completed in a few iterations. The iteration number necessary for success in learning hardly scales with the network size and sequence length; therefore, the LbAP algorithm can train large-scale SNNs in an energy- and time-efficient manner.

Nevertheless, the learning capacity of the $n$-SPSNN is limited mainly by (i) the use of one-hot coding for input (extremely sparse coding) and (ii) the limited number of hidden neurons $h$. The former limits the number of symbol representations for a given network setting. Therefore, dense coding is desired to improve the learning capacity of a given $n$-SPSNN, which we leave as a future work for the moment. Considering the latter, the optimal number of hidden neurons $h$ for successful learning scales with sequence length $l$ such that $h \approx 2l$. The number of learnable sequences with different lengths is also determined by this rule; the entire length of the concatenated sequences should satisfy this rule. Therefore, the network should be preset appropriately considering the complexity of the sequences that the $n$-SPSNN is trained on.

**TABLE 5. Acronyms.**

| Acronym | Explanation |
|---|---|
| bAPs | Backpropagating action potentials |
| eRBP | Event-driven random backpropagation |
| GRU | Gated recurrent unit |
| HTM | Hierarchical temporal memory |
| LbAP | Learning by backpropagating action potential |
| LSTM | Long short-term memory |
| MAC | Multiply-accumulate |
| $n$-SPSNN | $n$th order sequence-predicting SNN |
| SRM | Spike-response model |
| STDP | Spike timing-dependent plasticity |
| SynOps | Synaptic operations |

## APPENDIX

### A. MULTI-COMPARTMENT NEURON MODEL

Multi-compartment neurons were employed in the hidden and output layers in the $n$-SPSNN; each neuron is with a soma and multiple dendritic spines. Accordingly, somatic and dendritic potentials were evaluated separately. The somatic potential of neuron $i$ ($u_i^s$) was evaluated using the SRM [2] expressed as

$$u_i^s(t) = \eta\left(t - \hat{t}_i\right) + \sum_j w_{ij} \sum_f \epsilon(t - t_j^{(f)}) + \int_0^\infty \kappa(s) I_i^{ext}(t-s)ds, \quad (2)$$

where $\hat{t}_i$, $w_{ij}$, and $t_j^{(f)}$ denote the last spike time of neuron $i$, the weight of the synapse between neurons $j$ and $i$, and the $f^{\text{th}}$ spike time of neuron $j$, respectively. A refractory period and leaky integration of postsynaptic current are realized by the kernels $\eta$ and $\epsilon$, respectively. An externally injected current into neuron $i$ for supervised learning is denoted by $I_i^{ext}$.

The kernels are expressed as

$$\eta(t) = -(u^s_{reset} - u^s_r)\exp(-\frac{t}{t^s_m})\Theta(t), \tag{3}$$

$$\epsilon(t) = \epsilon_0\left[\exp\left(-\frac{t}{t^s_m}\right) - \exp(-\frac{t}{t^s_s})\right]\Theta(t), \tag{4}$$

$$\kappa(t) = \kappa_0 \exp\left(-\frac{t}{t^s_m}\right)\Theta(t), \tag{5}$$

where $u^s_{reset}$ and $u^s_r$ are the most hyperpolarized membrane potential (immediately after spiking) and the resting potential at the soma, respectively. At the soma, the postsynaptic current and potential decay exponentially with time constants of $t^s_s$ and $t^s_m$, respectively. The pre-exponential factors $\epsilon_0$ and $\kappa_0$ are positive constants. The somatic membrane potential exceeding a threshold for spiking fires a spike, and the potential is evaluated on the next time step with the updated $\hat{t}_i$.

The SRM applied to the dendritic potential evaluation. However, because no dendritic spikes are allowed, the first term on the right-hand side of (2) is ruled out. Furthermore, because supervision current pulses are applied to the soma only, the last term on the right-hand side of (2) is excluded. The same kernel in (4) was used but with the parameters $t^d_s$ and $t^d_m$ instead of $t^s_s$ and $t^s_m$. The replacement considers different responses of postsynaptic current and membrane potential to presynaptic spikes for a soma and dendritic spine, based on physiological observations [16], [46]. The neuronal parameters used in this study are listed in Table 1.

### B. TRAINING RNN WITH LSTM AND GRU LAYER

For the LSTM and GRU experiment, we trained a two-layer neural network with a recurrent unit. The first layer is the LSTM or GRU layer with 40 units and the second layer is a dense layer with 20 output neurons. Training employed categorical cross-entropy as a loss function and the Adam optimizer with a learning rate of 0.001. To realize *n*th order prediction, *n*-long subsequences were taken as inputs and encoded as an *m*-long real-valued vector (0–1) using a real-valued dense distributed representation. The output was an *m*-long vector that indicates a predicted element given a subsequence including *n* preceding elements. During training, a desired output was encoded as a one-hot vector with which the weights were updated ad hoc, i.e., online learning.

### C. ACRONYMS

See Table 5.

## REFERENCES


[1] P. Dayan and L. F. Abbott, *Theoretical Neuroscience*. London, U.K.: MIT Press, 2001.

[2] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[3] D. S. Jeong, "Tutorial: Neuromorphic spiking neural networks for temporal learning," *J. Appl. Phys.*, vol. 124, no. 15, Oct. 2018, Art. no. 152002.

[4] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers Neurosci.*, vol. 12, p. 774, Oct. 2018.

[5] E. O. Neftci, "Data and power efficient intelligence with neuromorphic learning machines," *iScience*, vol. 5, pp. 52–68, Jul. 2018.

[6] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[7] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[8] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Netw.*, vol. 111, pp. 47–63, Mar. 2019.

[9] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers Neurosci.*, vol. 7, p. 178, Oct. 2013.

[10] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 7, pp. 3227–3235, Jul. 2018.

[11] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 41, pp. 11441–11446, Oct. 2016.

[12] V. Kornijcuk and D. S. Jeong, "Recent progress in real-time adaptable digital neuromorphic hardware," *Adv. Intell. Syst.*, vol. 1, no. 6, Oct. 2019, Art. no. 1900030.

[13] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers Neurosci.*, vol. 11, p. 324, Jun. 2017.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: http://arxiv.org/abs/1412.3555

[16] P. J. Sjöström and M. Häusser, "A cooperative switch determines the sign of synaptic plasticity in distal dendrites of neocortical pyramidal neurons," *Neuron*, vol. 51, no. 2, pp. 227–238, Jul. 2006.

[17] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[18] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.

[19] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, 1982.

[20] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.

[21] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[22] J. S.-D. Jasmine Collins and D. Sussillo, "Capacity and trainability in recurrent neural networks," presented at the 34th Int. Conf. Mach. Learn., Sydney, NSW, Australia, 2017.

[23] D. L. Wang and B. Yuwono, "Anticipation-based temporal pattern generation," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, no. 4, pp. 615–628, Apr. 1995.

[24] J. Brea, W. Senn, and J.-P. Pfister, "Matching recall and storage in sequence learning with spiking neural networks," *J. Neurosci.*, vol. 33, no. 23, pp. 9565–9575, Jun. 2013.

[25] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting," *Neural Comput.*, vol. 22, no. 2, pp. 467–510, Feb. 2010.

[26] I. R. Fiete, W. Senn, C. Z. H. Wang, and R. H. R. Hahnloser, "Spike-time-dependent plasticity and heterosynaptic competition organize networks to produce long scale-free sequences of neural activity," *Neuron*, vol. 65, no. 4, pp. 563–576, Feb. 2010.

[27] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning," *Neural Comput.*, vol. 18, no. 6, pp. 1318–1348, Jun. 2006.

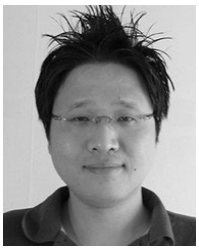
[28] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, "Connectivity reflects coding: A model of voltage-based STDP with homeostasis," *Nat. Neurosci.*, vol. 13, p. 344, Mar. 2010.

[29] B. Gardner and A. Grüning, "Supervised learning in spiking neural networks for precise temporal encoding," *PLoS ONE*, vol. 11, no. 8, Aug. 2016, Art. no. e0161335.
[30] R. V. Florian, "The chronotron: A neuron that learns to fire temporally precise spike patterns," *PLoS ONE*, vol. 7, no. 8, Aug. 2012, Art. no. e40233.
[31] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike pattern association neuron for learning spatio-temporal spike patterns," *Int. J. Neural Syst.*, vol. 22, no. 04, Aug. 2012, Art. no. 1250012.
[32] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns," *PLoS ONE*, vol. 8, no. 11, Nov. 2013, Art. no. e78318.
[33] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers Neural Circuits*, vol. 10, p. 23, Mar. 2016.
[34] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural Comput.*, vol. 28, no. 11, pp. 2474–2504, Nov. 2016.
[35] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, "Rate, timing, and cooperativity jointly determine cortical synaptic plasticity," *Neuron*, vol. 32, no. 6, pp. 1149–1164, Dec. 2001.
[36] J. Lisman, "A mechanism for the Hebb and the anti-Hebb processes underlying learning and memory," *Proc. Nat. Acad. Sci. USA*, vol. 86, no. 23, pp. 9574–9578, Dec. 1989.
[37] C. Hansel, A. Artola, and W. Singer, "Relation between dendritic $Ca^{2+}$ levels and the polarity of synaptic long-term modifications in rat visual cortex neurons," *Eur. J. Neurosci.*, vol. 9, no. 11, pp. 2309–2322, Nov. 1997.
[38] K. Cho, J. P. Aggleton, M. W. Brown, and Z. I. Bashir, "An experimental test of the role of postsynaptic calcium levels in determining synaptic strength using perirhinal cortex of rat," *J. Physiol.*, vol. 532, no. 2, pp. 459–466, Apr. 2001.
[39] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neurosci.*, vol. 3, no. 9, pp. 919–926, Sep. 2000.
[40] R. C. Froemke and Y. Dan, "Spike-timing-dependent synaptic modification induced by natural spike trains," *Nature*, vol. 416, no. 6879, pp. 433–438, Mar. 2002.
[41] E. M. Izhikevich and N. S. Desai, "Relating STDP to BCM," *Neural Comput.*, vol. 15, no. 7, pp. 1511–1523, Jul. 2003.
[42] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers Neurosci.*, vol. 7, p. 272, Jan. 2014.
[43] E. Bienenstock, L. Cooper, and P. Munro, "Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex," *J. Neurosci.*, vol. 2, no. 1, pp. 32–48, Jan. 1982.
[44] L. N. Cooper and M. F. Bear, "The BCM theory of synapse modification at 30: Interaction of theory with experiment," *Nature Rev. Neurosci.*, vol. 13, no. 11, pp. 798–810, Nov. 2012.
[45] E. Oja, "Simplified neuron model as a principal component analyzer," *J. Math. Biol.*, vol. 15, no. 3, pp. 267–273, Nov. 1982.
[46] J. C. Magee, "Dendritic integration of excitatory synaptic input," *Nature Rev. Neurosci.*, vol. 1, no. 3, pp. 181–190, Dec. 2000.

**DOHUN KIM** received the B.S. degree in materials science and engineering from Seoul National University, Seoul, South Korea, in 2016, where he is currently pursuing the Ph.D. degree in materials science and engineering. Since 2016, he has been focusing on learning algorithms for neuromorphic hardware implementation, especially for temporal learning.

**VLADIMIR KORNIJCUK** received the B.S. degree in telecommunication physics and electronics from Vilnius University, Vilnius, Lithuania, the M.S. degree in materials science and engineering from the Seoul National University of Science and Technology, Seoul, South Korea, and the Ph.D. degree in nano and information technology from the University of Science and Technology, Seoul. He is currently a Postdoctoral Scholar with Hanyang Univeristy, South Korea. His current research interest includes digital neuromorphic processor design.

**CHEOL SEONG HWANG** received the Ph.D. degree from Seoul National University, Seoul, South Korea, in 1993. Since 1998, he has been a Professor with the Department of Materials Science and Engineering, Seoul National University. His current research interests include high-k gate oxides, dynamic random access memory capacitors, new memory devices, including resistive RAM devices and ferroelectric materials and devices, energy storage capacitors, as well as neuromorphic computing.



**DOO SEOK JEONG** (Member, IEEE) received B.E. and M.E. degrees in materials science from Seoul National University, in 2002 and 2005, respectively, and the Ph.D. degree in materials science from RWTH Aachen, Germany, in 2008. He was with the Korea Institute of Science and Technology, from 2008 to 2018. He is currently an Associate Professor with Hanyang University, South Korea. His current research interests include spiking neural networks for sequence learning and future prediction, learning algorithms, spiking neural network design, and digital neuromorphic processor design.