International Journal of Neural Systems, Vol. 28, No. 2 (2018) 1750015 (20 pages)

DOI: 10.1142/S0129065717500150

World Scientific
www.worldscientific.com

A Scalable Weight-Free Learning Algorithm for Regulatory Control of Cell Activity in Spiking Neuronal Networks

Xu Zhang* and Greg Foderaro
Mechanical Engineering and
Materials Science, Duke University
Box 90300 Hudson Hall
Durham, NC, US
**xz70@duke.edu*

Craig Henriquez
Biomedical Engineering
Duke University
Box 90281 Hudson Hall
Durham, 27708, US

Silvia Ferrari
Sibley School of Mechanical
and Aerospace Engineering
Cornell University, 105 Upson Hall
Ithaca, New York, 14853, US

Accepted 9 December 2016
Published Online 8 March 2017

Recent developments in neural stimulation and recording technologies are providing scientists with the ability of recording and controlling the activity of individual neurons *in vitro* or *in vivo*, with very high spatial and temporal resolution. Tools such as optogenetics, for example, are having a significant impact in the neuroscience field by delivering optical firing control with the precision and spatiotemporal resolution required for investigating information processing and plasticity in biological brains. While a number of training algorithms have been developed to date for spiking neural network (SNN) models of biological neuronal circuits, exiting methods rely on learning rules that adjust the synaptic strengths (or weights) directly, in order to obtain the desired network-level (or functional-level) performance. As such, they are not applicable to modifying plasticity in biological neuronal circuits, in which synaptic strengths only change as a result of pre- and post-synaptic neuron firings or biological mechanisms beyond our control. This paper presents a weight-free training algorithm that relies solely on adjusting the spatiotemporal delivery of neuron firings in order to optimize the network performance. The proposed weight-free algorithm does not require any knowledge of the SNN model or its plasticity mechanisms. As a result, this training approach is potentially realizable *in vitro* or *in vivo* via neural stimulation and recording technologies, such as optogenetics and multielectrode arrays, and could be utilized to control plasticity at multiple scales of biological neuronal circuits. The approach is demonstrated by training SNNs with hundreds of units to control a virtual insect navigating in an unknown environment.

Keywords: Spiking neural networks; optogenetics; neural control; spike timing-dependent plasticity; neuromorphic; reinforcement learning.

1. Introduction

Recent developments in neural stimulation and recording technologies are revolutionizing the field of neuroscience, providing scientists with the ability of recording and controlling the activity of individual neurons in the brain of living animals, with very high spatial and temporal resolution.[1] Thanks to methods such as optogenetics, which deliver controlled cell firings to live neurons *in vitro* or *in vivo*, it is now becoming possible to determine which regions of the brain are primarily responsible for encoding particular stimuli and behaviors. Despite this remarkable progress, the relationship between biophysical models of synaptic plasticity and circuit-level learning, also known as functional plasticity, remains unknown. This important gap has been recently identified as one of the outstanding challenges in reverse engineering of the brain.[2]

Many spiking neural network (SNN) learning algorithms have been proposed to model and replicate both the synaptic plasticity and circuit-level learning capabilities observed in biological neuronal networks.[3–9] Experimental evidence has shown that learning in the brain is accompanied by changes in synaptic efficacy referred to as *synaptic plasticity*.[10] Developing learning rules for altering synaptic efficacy, commonly referred to as synaptic strength or *weight*, has since been the focus of both artificial neural networks (ANN) and SNN learning algorithms to date. Along the same lines, SNN learning algorithms inspired by biological mechanisms, such as spike timing dependent plasticity (STDP),[11–13] have recently been proposed to modify synaptic weights according to a learning rule model based on STDP or Hebbian plasticity, so as to optimize the network performance.[14–19] Other SNN learning algorithms include Spike-Prop[20,21] and ReSuMe,[22] which use classical backpropagation and Widrow–Hoff learning rules in combination with STDP to adapt the synaptic weights so as to produce a desired SNN response. When trained by these approaches, computational SNN have been shown to be very effective at solving decision and control problems in a number of applications, including delay learning, memory, and pattern classification.[23–26]

Despite their effectiveness, none of the computational SNN learning algorithms to date have been implemented or validated experimentally on biological neurons *in vitro* or *in vivo*. Such experimentation could help develop plausible models linking synaptic-level and functional-level plasticity in the brain, as well as find many potential neuroscience applications by closing the loop around the recording and the control of neuron firings. Existing SNN learning algorithms, however, are difficult to implement and test experimentally because they utilize learning rules for manipulating synaptic weights, while the strength of biological synapses is not easily modified in live neuronal networks. Experimental methods for regulating synaptic strengths in biological neurons, for example via alteration of intracellular proteins or neurotransmitters such as AMPA receptors,[27] do not lend themselves to the implementation of parallel and frequent weight changes, followed by the observation of network performance, as typically dictated by SNN learning algorithms.

To overcome this fundamental hurdle, the authors have proposed a new *weight-free* NN learning paradigm in which the learning rule regulates the spatiotemporal pattern of cell firings (or spike trains) to achieve a desired network-level response by indirectly modulating synaptic plasticity.[28,29] Because this learning paradigm does not rely on manipulating synaptic strengths, in principle its learning rule can be implemented experimentally by delivering the neural stimulation patterns determined by the algorithm to biological neurons using optogenetics or intracellular stimulation. As a first step, this new learning paradigm was demonstrated by showing that the synaptic strengths of a few neurons could be accurately controlled by optimizing a radial basis function (RBF) spike model using an analytical steepest-gradient descent method.[29] As a second step, the method was extended to networks with up to 10 neurons by introducing an unconstrained numerical minimization algorithm for determining the centers of the RBF model, such that the timings of the cell firings could be optimized.[28] The latter approach was also shown effective at training memristor-based neuromorphic computer chips that aim to replicate the functionalities of biological circuitry.[30,31] Because they are biologically inspired, these neuromorphic chips are characterized by STDP-like mechanisms that only adjust CMOS synaptic strengths by virtue of controllable applied voltages analogous to neuron firings. Therefore, they too are amenable to a learning paradigm that

seeks to regulate the spatiotemporal pattern of cell firings (or spike trains) in lieu of the synaptic strengths.

Both biological and neuromorphic circuits typically involve much larger numbers of units than have been previously considered by the authors. Thus, this paper presents a new perturbative learning approach for scaling the weight-free learning paradigm up to networks with hundreds of neurons. The proposed approach is scalable because it does not rely on computing the spike model gradients analytically or numerically but rather on inferring the sign of the gradient by perturbation methods, effectively amounting to a weight-free resilient backpropagation[32] algorithm for SNNs.

Even in the simplest organisms, brain circuits are characterized by hundreds of neurons responsible for integrating diverse stimuli and for controlling multiple functionalities.[33] Because of their size, these neuronal structures are believed to provide robustness, redundancy, and reconfigurability, characteristics that are also desirable in neuromorphic circuits and engineering applications of computational SNNs. Recent studies on living insects have established that multiple forms of sensory inputs, such as visual, tactile, and olfactory information are integrated within the central complex (CX) circuit, to control and adapt movements to surrounding environments.[33,34]

Inspired by these experimental studies in biology, this paper seeks to demonstrate the weight-free learning algorithm on a virtual simulation of a walking insect in a similar arena, with multiple sensory stimuli, and sensorimotor control provided by an SNN model. The simulation results show that when the SNN model is trained by the proposed weight-free learning algorithm, the insect is capable to navigate new and complex terrains efficiently and robustly, based only on the inputs from simulated olfactory and tacticle receptors. Besides validating the effectiveness of the learning algorithm, these results demonstrate that the proposed weight-free approach enables a more direct and effective transfer of biological findings to synthetic systems. Also, because the size of the SNN trained in this paper is comparable to that of the insect CX, the weight-free algorithm could potentially be realized *in vivo* in living insects, to investigate and control plasticity from the synaptic level to the functional level, via electrical or optogenetics stimulation.

2. SNN Model and Architecture

SNNs are computational models of neuronal networks motivated by biological studies demonstrating that spike patterns are an essential component of information processing in the brain. It has been hypothesized that SNNs have evolved in nature because they are flexible or reconfigurable, tolerant to noise or *robust*, require low power consumption, and can encode complex temporal and spatial inputs efficiently as correlated spike sequences known as *spike trains*.[35,36] Because of these potential advantages, SNNs are implemented as neuromorphic computational platforms in software or hardware for applications such as classification,[37] computational neuroscience,[38–40] and neurorobotics.[41]

The two crucial considerations involved in choosing the SNN model are its range of neurocomputational behaviors and its computational efficiency.[42] As can be expected, the implementation efficiency typically increases with the number of features and behaviors that can be accurately reproduced,[42] such that each model offers a tradeoff between these competing objectives. The computational neuron model that is most biophysically accurate is the well-known Hodgkin–Huxley (HH) model.[43] Due to its extremely low computational efficiency, however, using the HH model to simulate large networks of neurons can be computationally prohibitive.[42] The simplest model of spiking neuron is the leaky integrate-and-fire (LIF). While it can only reproduce the dynamics of a Class-1 excitable neurons that fire tonic spikes, LIF has the advantages that it displays the highest computational efficiency and is amenable to mathematical analysis. Moreover, the LIF model was found to accurately reproduce the firing dynamics of CMOS neurons.[44] Therefore, it is adopted in this paper to simulate all SNN architectures, and is reviewed in the appendix for completeness.

Motivated by both computational NNs and biological neuronal networks for sensorimotor control, such as the insect CX,[33] the following three regions or circuits are identified in the SNN model: input neurons ($\mathcal{I}$), output neurons ($\mathcal{O}$), and computational

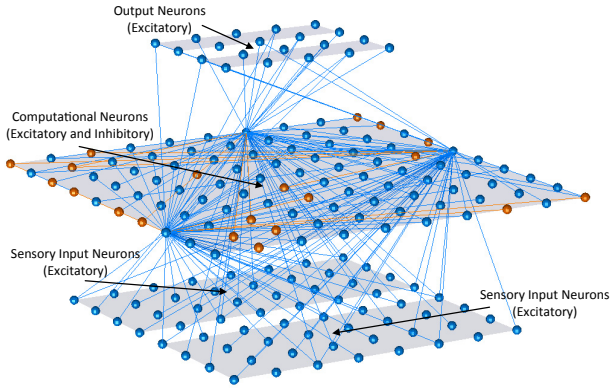


Fig. 1. Computational SNN architecture where synaptic connections are illustrated for three neurons randomly selected in each layer.

or *hidden* neurons ($\mathcal{H}$). Input neurons receive external information, such as sensory inputs, by virtue of the synaptic currents of structures, such as neuropils, that provide, for example, visual, olfactory, or tactile inputs. In this paper, sensory inputs are assumed to be already rate coded at the input-neuron level in a manner proportional to the stimulus (Sec. 4). The output neurons provide the network response which, in neuromorphic systems, may be decoded and used to command the actuators and, in biological systems, may consist of synaptic current inputs to motoneurons, such as for example central pattern generators (CPGs), that actuate muscles and limbs. Neurons that are only connected to input or output neurons, or to each other, are referred to as hidden neurons, as illustrated in Fig. 1. Assuming the availability of optogenetics or other firing control tool, this paper simulates a subset of input neurons that are light sensitive and can be made to fire on command.[45]

The goal of the proposed weight-free algorithm is to determine the firing sequences to be delivered to the input neurons ($\mathcal{I}$), optically or via neural stimulation, such that the dynamic network response to all possible inputs is optimized. By stimulating the training neurons and/or receiving sensory inputs, the stimulus current I_{stim} in Eq. (A.1) is modified, thereby altering the membrane potential. When two or more neurons are connected by synapses, the membrane potential of the postsynaptic neuron, governed by (A.1), also depends on the synaptic current from the presynaptic neurons given by,

$$I_{\text{syn}}(t) = g_{\text{syn}}(t)^*[V_{\text{post}}(t) - E_{\text{syn}}], \tag{1}$$

where I_{syn} is the synaptic current from the presynaptic neurons, $g_{\text{syn}}(t)$ is the synaptic conductance, V_{post} is the membrane potential of the postsynaptic neuron, and E_{syn} is the synaptic reversal potential. Therefore, stimulating neurons in $\mathcal{I}$ ultimately causes other neurons in the SNN to fire at later times, once their membrane potentials reach V_{th} via synaptic current inputs.

A biologically plausible model of synaptic connections is constructed by sampling uniformly and at random the distribution,[46]

$$p_{i,j} = C\ \exp\left[-\left(\frac{D(i,j)}{\lambda}\right)^2\right], \tag{2}$$

where for excitatory neurons $C = 0.8$, and for inhibitory neurons $C = 0.2$. The function $D(i,j)$ represents the Euclidean distance between neurons i and j in the neural circuit. The connection parameter λ is adjustable and, as λ approaches zero, so does the number of connections. The value $\lambda = 5$ is chosen for all SNN models in this paper. In particular, connections are formed according to the above distributions within each SNN region, and pairwise between adjacent regions, as schematized in Fig. 1. As in many artificial and biological networks, the hidden neurons provide separation between input and output neurons such that, although the network is fully connected and recurrent, there are no synaptic connections between input and output neurons. Once synaptic connections are established probabilistically from Eq. (2), each synaptic conductance is modeled using the alpha function,[47]

$$g_{\text{syn}}(t) = \bar{g}_{\text{syn}} h(e^{-\frac{t}{\tau_{\text{delay}}}} - e^{-\frac{t}{\tau_{\text{rise}}}}), \tag{3}$$

where the normalization factor is defined as,

$$h \triangleq \left(-e^{-\frac{t_{\text{peak}}}{\tau_{\text{rise}}}} + e^{-\frac{t_{\text{peak}}}{\tau_{\text{decay}}}}\right)^{-1} \tag{4}$$

to ensure that the amplitude equals $\bar{g}_{\text{syn}}$, and such that the conductance peaks at time:

$$t_{\text{peak}} = \frac{\tau_{\text{decay}}\ \tau_{\text{rise}}}{(\tau_{\text{decay}} - \tau_{\text{rise}})} \ln\left(\frac{\tau_{\text{decay}}}{\tau_{\text{rise}}}\right). \tag{5}$$

Importantly, the SNN learning algorithm has no control over the synaptic strengths and, thus, all synapses change over time only by virtue of the STDP rule (reviewed in the appendix). As observed in biological neurons,[48,49] if the presynaptic neuron fires before the postsynaptic neuron, the synapse is strengthened, and if it fires after, the synapse is

weakened. The pair-based STDP rule can be numerically implemented in the LIF-SNN using two local variables, x_j and y_i, representing low-pass filtered versions of the presynaptic spike train and the postsynaptic spike train, respectively.[47] Let us consider the synapse between neuron j and neuron i. Suppose each spike from presynaptic neuron j contributes to trace x_j at the synapse,

$$\frac{dx_j}{dt} = -\frac{x_j}{\tau_x} + \sum_{t_j^f} \delta(t - t_j^f), \tag{6}$$

where t_j^f denotes the firing times of the presynaptic neuron, and δ is the Dirac function. Then, the trace x_j is increased by one at t_j^f and subsequently decays with time constant τ_x. Similarly, each spike from postsynaptic neuron i contributes to a trace y_i according to,

$$\frac{dy_i}{dt} = -\frac{y_i}{\tau_y} + \sum_{t_i^f} \delta(t - t_i^f), \tag{7}$$

where t_i^f denotes the firing times of the postsynaptic neuron. When a presynaptic spike occurs, the weight decreases proportionally to the value of the postsynaptic trace y_i, while if a postsynaptic spike occurs, a potentiation of the weight is induced.

Firing rate coding is used for conversions between spike trains and continuous-time signals.[47] Rate coding computes the mean firing frequency of a chosen set of K neurons over a time window, $[t - t_r, t]$, as follows,

$$f(t_r) = \frac{1}{K} \sum_{i}^{K} z_i(t_r), \tag{8}$$

where $z_i(t_r)$ denotes the number of spikes of neuron i during $[t - t_r, t]$. By this approach, the continuous-time signal or function $f(t_r)$ can be encoded in the firing sequences (or spike trains) of a population of K neurons, where K can vary from one to many units. Population decoding and encoding is adopted in this paper, because it is believed to be more robust as well as more biologically plausible than single-unit decoding and encoding. Also, it is routinely utilized in biological neural systems as a useful indicator of neural activity in sensorimotor circuits and individual cells.[50,51] Finally, the SNN model described in this section is simulated using the open-source software Neural Circuit SIMulator (CSIM),[52] and the parameters provided in the appendix.

3. Weight-free SNN Learning Algorithm

In order to be applicable to biological and neuromorphic circuits, the weight-free learning algorithm presented in this section assumes that no knowledge of the SNN neuron model, connectivity, or synaptic strengths is available. It is assumed, however, that the SNN input, output, hidden, and training neurons, described in the previous section can be stimulated or recorded from with high spatiotemporal precision. In Sec. 5, noisy sensory inputs and currents are introduced to investigate the SNN robustness to stimulation errors by which the stimulus delivered induces multiple nearby neurons to fire.

In the proposed algorithm, SNN learning and synaptic plasticity are achieved through the application of training stimuli (e.g. optical or electrical pulses) delivered to pairs of training neurons at precise times determined via perturbative reinforcement learning. By controlling the firing of selected training neurons, the algorithm indirectly modifies the synaptic strengths according to internal synaptic plasticity mechanisms, such as STDP. However, unlike existing methods,[11–13] prior knowledge or models of these mechanisms are never utilized by the weight-free learning algorithm presented in this paper. The firing times of the training stimuli are determined solely from the error between the observed SNN response and the *desired* SNN response, by minimizing it in batch mode.

Whether optical or electrical, a training stimulus to an input neuron $i \in \mathcal{I}$ can be modeled as a square pulse function,

$$s_i(t) = w \sum_{l=1}^{M} \left[H\left(t - c_{i,l} + \frac{\beta}{2}\right) - H\left(t - c_{i,l} - \frac{\beta}{2}\right) \right], \tag{9}$$

where $c_{i,l}$ represents the temporal center of the lth square pulse, M is the total number of square pulses, w is the pulse amplitude, β is the pulse duration, and $H(\cdot)$ is the Heaviside function. Through simulation or experimentation, the parameters of the pulse function are chosen such that each pulse will reliably induce neuron i to spike once and only once. In this case, they are chosen as $w = 7 \times 10^{-7}$ (Amps) and $\beta = 0.004$ (sec) based on the SNN simulation described in Sec. 2. The pulses delivered to a

pair of neighboring neurons $(i,\ j)$, where $i, j \in \mathcal{I}$, are offset by a parameter b_k that varies with the training epoch, indexed by k, such that the choice $c_{j,l} = c_{i,l} + b_k$ minimizes the network error over time.

Let m denote the number of training cases available for which the desired SNN response to a given sensory stimulus is known. If $\mathbf{p}_\imath$ $(\imath = 1, \ldots, m)$ denotes a vector of firing rates corresponding to a known sensory stimulus, where each firing rate may be defined with respect to a population (or subset) of input neurons $(\mathcal{I})$, then the desired SNN response can be denoted by a corresponding vector $\mathbf{u}_\imath^*$ of firing rates that represent the desired response for one or more subsets of output neurons $(\mathcal{O})$. Then, for a training database $\mathcal{D} = \{(\mathbf{p}_1, \mathbf{u}_1^*), \ldots, (\mathbf{p}_m, \mathbf{u}_m^*)\}$, the SNN error at the kth epoch is defined as,

$$e_k = \frac{1}{m}\sqrt{\sum_{\imath}^{m}[\mathbf{u}_\imath^* - \mathbf{u}_\imath(k)]^T[\mathbf{u}_\imath^* - \mathbf{u}_\imath(k)]}, \quad (10)$$

where $\mathbf{u}_\imath(k)$ is the decoded SNN output at epoch k, obtained by applying Eq. (8) to the firings of the set of output neurons $(\mathcal{O})$, in response to input $\mathbf{p}_\imath$.

Because of the scale and the complexity of the SNN, its response to a sensory input $\mathbf{p}_\imath$ needs to be determined experimentally. Moreover, to circumvent computing the error gradient analytically or numerically, the sign of the error change brought about by the stimulus in Eq. (9) is also determined experimentally, via the following perturbation technique. Every iteration of the training algorithm, conducted over time (t), is comprised of a testing phase followed by a training phase. The *testing phase* consists of determining the sign of the error change over $\mathcal{D}$, as brought about by training stimuli in the form of Eq. (9). The *training phase* consists of delivering additional training stimuli also in the form of Eq. (9), but such that the synaptic strengths are changed in the direction of minimum SNN error.

During the testing phase, the SNN error in Eq. (10) is evaluated before and after synaptic strength perturbations. These perturbations are accomplished by delivering the square pulses in Eq. (9) with $b_k = b_0$ for any k, and $b_0 = \pm 0.002$, so as to induce small perturbations in the synaptic strengths. At each epoch (k), the pair of input neurons chosen to receive a pair of training stimuli, say (i, j), $i, j \in \mathcal{I}$, is chosen from an ordered list, $\mathcal{N}$, containing all possible neuron pairs in $\mathcal{I}$, such that from the binomial coefficient, $|\mathcal{N}| = \frac{N!}{2!(N-2)!}$, where $|\cdot|$ denotes the cardinality of a set, $N = |\mathcal{N}|$, and ! denotes the factorial of a non-negative integer. The training stimuli are delivered in the absence of sensory stimuli. Separately, before and after delivering the training stimuli, the SNN error in Eq. (10) is evaluated by delivering each (sensory stimulus) firing rate $\mathbf{p}_\imath \in \mathcal{D}$ to the corresponding subsets of input neurons $(\mathcal{I})$. This is accomplished by applying pulses with the desired frequency for a duration of $t_e = 0.04$, chosen such that the signal can propagate through the SNN and produce a response in the output neurons $(\mathcal{O})$ that can be reliably decoded as output $\mathbf{u}_\imath$. Let $e_{k,1}$ and $e_{k,2}$ denote the SNN errors, computed from Eq. (10), before and after the training stimuli are delivered, respectively. Then, the sign of the error change, $\Delta e_k \triangleq e_{k,2} - e_{k,1}$, reflects the SNN error sensitivity to the weight perturbations induced by the training stimuli in Eq. (9).

Based on the SNN-error sign changes during consecutive epochs, it is then possible to determine the parameters of the square-pulse training stimuli, defined in Eq. (9), such that the SNN error is minimized over time. This is accomplished during the training phase, when the training stimuli in Eq. (9) are designed using the offset parameter according to the following learning rule,

$$b_{k+1} = -\operatorname{sgn}(\Delta e_k) b_0 \quad (11)$$

such that $c_{j,l} = c_{i,l} + b_k$. In order to compensate for former episodes of synaptic plasticity,[53] the number of square pulses adopted in Eq. (9) is chosen according to the following rule,

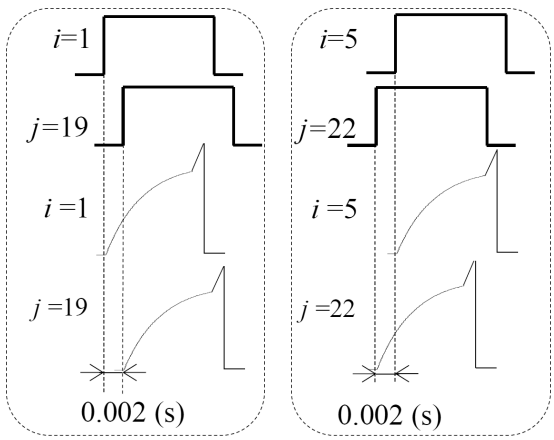
$$M_{k+1} = \begin{cases} 2M_k, & \text{if } b_0 > 0 \text{ and } \Delta e_k > 0, \\ \frac{1}{2}M_k, & \text{if } b_0 < 0 \text{ and } \Delta e_k > 0, \\ M_k, & \text{if } \Delta e_k < 0. \end{cases} \quad (12)$$

At each epoch (k), a new pair of neurons is selected to receive the training signals from the ordered list $\mathcal{N}$, and the process is repeated until the error decreases below a desired value or satisfies a desired stopping criterion.

The implementation of the two phases, including the above learning rules, is summarized in the weight-free SNN learning algorithm in Fig. 2. Figures 3 and 4 show that, without manipulating the synaptic weights, the training stimuli delivered

```
Initialize SNN
Set e_{k,2} = e_init > e_min for k = 0
Set k = 1
while e_{k-1,2} > e_min do
    Pick neurons i, j randomly
    Randomly initialize b_0
        s_1(t) = s_i(t) and s_2(t) = s_j(t + b_0)
        ℓ = 1
    while ℓ ≤ 2 do
        Stimulate SNN with p_1, ..., p_m
        Record and decode SNN outputs u_1, ..., u_m
        Calculate e_{k,ℓ} according to (10)
        if ℓ = 2 then
            if e_{k,2} > e_{k,1} then
                b_k = −sgn(e_{k−1,1} − e_{k−1,2}) b_0
                s_1(t) = s_i(t)
                s_2(t) = s_j(t + b_k)
            end
        end
        Stimulate neurons i and j using stimuli s_1(t) and s_2(t), respectively
        ℓ=ℓ + 1
    end
    k = k + 1
end
```

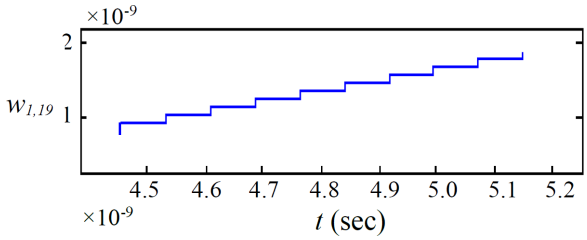
Fig. 2. Pseudocode of weight-free SNN learning algorithm.



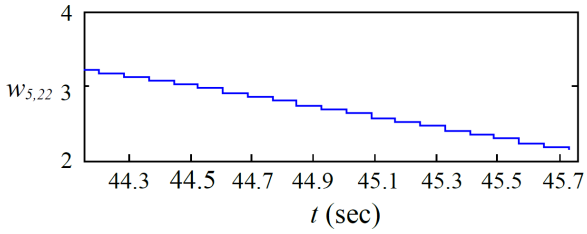
(a) (b)

Fig. 3. Training stimuli delivered by the weight-free algorithm in Fig. 2(a), and induced action potentials of two pairs of pre- and post-synaptic neurons $(1, 19)$ and $(5, 22)$ (b), causing the weight changes in Fig. 4.

by this algorithm (Fig. 3(a)) induce pre- and post-synaptic firings (Fig. 3(b)) that reliably cause synaptic strengths to change in the desired direction over time (Fig. 4) by virtue of the underlying plasticity



(a)



(b)

Fig. 4. Synaptic strength changes brought about by the weight-free algorithm's training stimuli in Fig. 3, where the pair of stimuli in Fig. 3(a) potentiates the synapse as shown in (a), and the pair of stimuli in Fig. 3(b) depresses the synapse as shown in (b).

mechanism. Because the weight-free algorithm does not control the synaptic strengths or the plasticity mechanism, it is potentially applicable to biological neuronal networks.

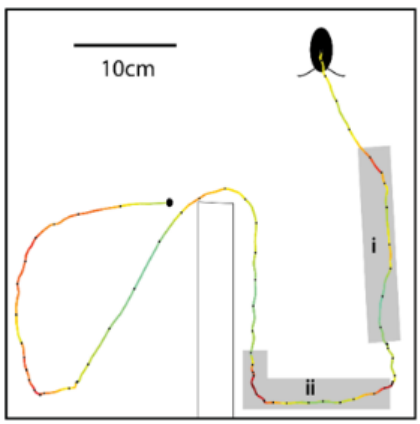
4. Application: Virtual Insect Control and Navigation

The brains of even the simplest of organisms have shown the remarkable ability to adapt and learn to solve complex problems necessary for survival. In simple organisms, such as the cockroach, physiological recordings have established that multiple forms of sensory inputs, including visual cues and tactile information from mechanosensors on the antenna, are integrated within the CX to control and adapt movements to surrounding environments.[34] These studies utilize methods for stimulating or recording from populations of brain neurons in freely behaving cockroaches using custom fabricated wire bundles inserted into the animal brain prior to releasing it into a controlled arena (Fig. 5(a)).

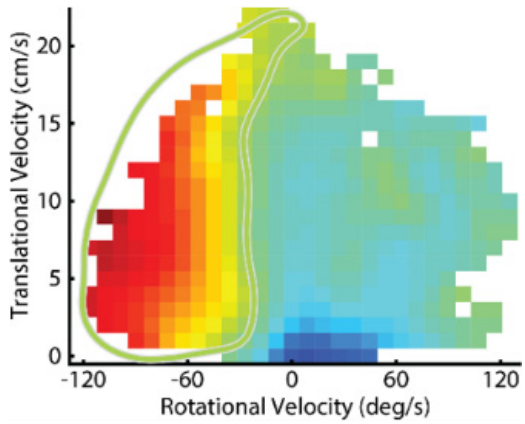
With the recording wires in place, the animal is presented with controlled stimuli, including various obstacles, and its motion is recorded with a video camera placed above the arena. These data are merged so that timing of the neural activity

relative to motion and stimuli is easily determined (Fig. 5(b)). Extracellular recording and lesion techniques link the CX to higher control of movement, and demonstrate that changes in the activity (e.g. firing rates) of individual units immediately precede changes in the firing rates of motoneurons responsible for locomotory behaviors such as walking speed, turning, and climbing.[33,54] The same wires can be used to stimulate the brain region to evoke altered behavior and plasticity.[55]

Even in simple organisms, the relationship between synaptic-level plasticity and higher-level learning and behavior remains unknown. Motivated by these experimental studies and the possibility of developing plausible models of sensorimotor mechanisms and plasticity, the proposed weight-free algorithm is used to train an idealized SNN model of insect CX (Fig. 1) to control the motion and navigation of a virtual animal in an unknown environment. After training, the virtual insect is placed in a region of complex topography, simulated using Matlab Virtual Reality Modeling Language (VRML),[56,57] as shown in Fig. 6. Using only the sensory stimuli from virtual antennas, the simulated insect must navigate the environment autonomously to reach a desired target by choosing a path of minimum distance and minimum elevation, so as to minimize its energy consumption.



(a)



(b)

Fig. 5. (Color online) Track of live insect in the arena color coded based on the activity level of a CX unit (a). Color-coded neural (spike) activity as a function of translational and rotational velocity over entire track (b), where warmer colors indicate higher firing frequency (taken from Ref. 55).

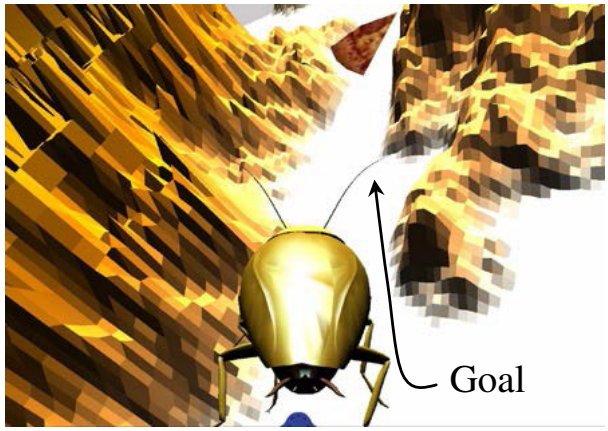


Fig. 6. Virtual insect navigating toward the goal in a complex terrain with hills of varying and unknown elevation (simulated in VRML[56,57]).

4.1. *Insect sensorimotor model*

A simple locomotion model is adopted by which the six-legged insect moves its three right legs all at the same speed v_R, and its left legs all at the same speed v_L. By this simplification, the insect can be modeled as a robot with two motors, each driving the speed of the right or left legs (like wheels). Then, the motion of the virtual insect in inertial frame $\mathcal{F}_\mathcal{W}$ can be modeled by the modified unicycle robot,

$$\begin{cases} \dot{x} = v\cos\theta, \\ \dot{y} = v\sin\theta, \\ v = (v_L + v_R)/2, \\ \dot{\theta} = (v_R - v_L)/L, \end{cases} \tag{13}$$

where x and y are the coordinates of the insect center of mass in $\mathcal{F}_\mathcal{W}$, v is the insect speed, θ is insect

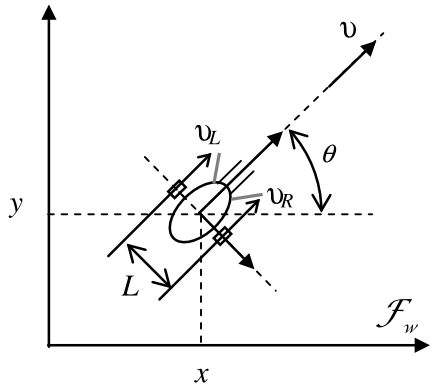


Fig. 7. Virtual insect kinematic model in inertial frame $\mathcal{F}_{\mathcal{W}}$.

heading angle, and L is the distance between the two motors, as illustrated in Fig. 7.

The right and left leg speeds, v_R and v_L, are determined by the firing times of the corresponding motoneurons, represented by t_R and t_L, respectively. The right and left motoneurons belong to the set of output neurons ($\mathcal{O}$) in the SNN model illustrated in Fig. 1. Then, using the Dirac delta function, $\delta(\cdot)$, the conversion between the motoneurons firings and the insect leg speed can be modeled as follows,

$$\dot{v}_{R,L} = -\frac{v_{R,L}}{\tau_{\text{motor}}} + \eta\ \delta(t - t_{R,L}), \tag{14}$$

where τ_{motor} is a time constant that results in a gradual decay of the motor speed following a spike by the corresponding motoneuron. As a result, the insect stops moving only after the motoneurons have not fired for a long time.

Sensory stimuli are simulated to provide the insect with information about the environment, particularly as it relates to obstacles and its target. As schematized in Fig. 8, the virtual insect has four exteroceptive sensors comprised of two "olfactory" sensors, to determine its distance from the target,

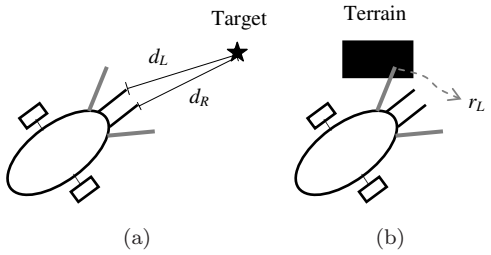


Fig. 8. Virtual insect olfactory (a) and tactile (b) sensory input models.

and two terrain sensors, to determine the elevation of the terrain nearby. These sensory inputs are rate-encoded into current stimuli that are delivered to corresponding populations of SNN input neurons (i.e. disjoint subsets of $\mathcal{I}$).

The sensor measurements are assumed to occur at the end points of the antennae, and the magnitudes of the current stimuli are governed by sensor models, as follows. For a fixed point target in $\mathcal{F}_{\mathcal{W}}$, let $d(\cdot)$ denote the Euclidean distance between the target and the position of the right or left olfactory sensors, denoted by coordinates (ξ_R, μ_R) or (ξ_L, μ_L), respectively. Then, the right and left olfactory inputs are modeled by,

$$g_{R,L}(\xi_{R,L}, \mu_{R,L}) = \alpha\{d(\xi_{R,L}, \mu_{R,L}) + \lambda[d(\xi_{R,L}, \mu_{R,L}) - d(\xi_{R,L}, \mu_{R,L})]\} \tag{15}$$

respectively, where $\alpha = 10^{-9}$ is a scaling factor, and $\lambda = 5$ is a constant parameter chosen to cause sensory stimuli to reliably produce realistic spike responses within a desired range of d_L and d_R. Because the position of the olfactory sensors depends on the position of the insect, the sensor stimuli are also implicit functions of the insect coordinates (x, y).

The terrain "tactile" sensory stimuli are simulated by considering left and right antennae that sense the elevation of the terrain at their given location. Let (χ_R, ζ_R) and (χ_L, ζ_L) denote the coordinates of the right and left tactile sensors, respectively. If $H(\chi, \zeta)$ denotes the elevation at (χ, ζ) in $\mathcal{F}_{\mathcal{W}}$, then the tactile inputs can be modeled as,

$$h_{R,L} = \left(\frac{\alpha\gamma}{H_{\max}}\right) \frac{H(\chi_{R,L}, \zeta_{R,L})}{[1 + H(\chi_{R,L}, \zeta_{R,L})]}, \tag{16}$$

where $\gamma = 0.1$ is a constant parameter chosen to bound the tactile inputs, and $H_{\max}$ is the maximum elevation value, such that when $H = H_{\max}$, the terrain is an insurmountable obstacle. Similarly to the olfactory inputs in Eq. (15), the tactile inputs in Eq. (16) are rate-encoded into current stimuli that are delivered to corresponding populations of input neurons ($\mathcal{I}$).

In order to investigate SNN robustness to errors, each of the four olfactory and tactile sensory inputs, g_R, g_L, h_R, and h_L, are corrupted by introducing additive noise. Let $n \sim U(0, 1)$ denote a scalar random variable sampled from a uniform distribution

with range [0, 1]. Then, the sensor noise is modeled as follows,

$$\begin{aligned} g_{R,L} &\leftarrow (\nu \cdot n) g_{R,L}, \\ h_{R,L} &\leftarrow (\nu \cdot n) g_{h,L}, \end{aligned} \tag{17}$$

where $0 \leq \nu \leq 1$ is the maximum noise amplitude, and the effect of noisy inputs can be investigated by replacing each sensory input with its noisy value given above.

4.2. *Insect CX model*

The virtual insect CX brain region, responsible for controlling the animal movements based on integrated sensory inputs, is simulated by means of the SNN model of the same scale, described in Sec. 2. In particular, the SNN input neurons ($\mathcal{I}$) receive all sensory input signals from olfactory and tactile stimuli, and the output neurons ($\mathcal{O}$) control the leg speeds. As schematized in Fig. 9, the input neurons are divided into four subsets that each receive the stimulus current based on the corresponding sensory inputs, computed by Eqs. (15) and (16). In each SNN layer, excitatory and inhibitory neurons are randomly generated with probability 0.8 and 0.2, respectively, according to previous findings by the authors.[58] Excitatory and inhibitory neurons can be simulated by utilizing positive and negative synaptic strengths, respectively. Then, based on the sensory stimuli and the subsequent activity of both input and hidden neurons, the output motoneurons control the speed of the right and left insect legs (Fig. 9).

Once the SNN model of the CX region is trained, the leg control must occur such that the insect reaches the target by minimizing distance and avoiding terrains with high elevation. Based on CX insect studies,[55] these motion objectives are formulated in terms of left (L) and right (R) motoneuron firing rates as follows,

$$\begin{pmatrix} f_L^* \\ f_R^* \end{pmatrix} = \begin{pmatrix} h_L & g_L \\ h_R & g_R \end{pmatrix} \begin{pmatrix} \kappa \\ \eta \end{pmatrix} \tag{18}$$

such that the desired output firing rates, f_L^* and f_R^*, depend on all four sensory inputs in Eqs. (15) and (16). The constant parameters are chosen such that $\kappa > \eta$, in order to prioritize terrain avoidance over reaching the target.

The weight-free learning algorithm is applied to the SNN CX model by developing a training database $\mathcal{D}$ comprised of $m = 6$ pairs of input and (desired) output firing rates. Each pair of firing rates $(\mathbf{p}_i, \mathbf{u}_i^*)$ encodes the sensory inputs and the (desired) motor outputs obtained from one of the six situations illustrated in Fig. 10, where the terrain is an obstacle with maximum elevation $H_{\max}$. In every case, the sensory input, $\mathbf{p}_i = [g_R\ g_L\ h_R\ h_L]^T$, and the desired motor output, $\mathbf{u}_i^* = [f_{R_i}^*\ f_{L_i}^*]^T$, are obtained from Eqs. (15)–(18). Then, during training, the SNN error in Eq. (10) can be computed by simulating the SNN response to one of the given sensory inputs in $\mathcal{D}$, and by comparing the decoded SNN output to the desired motor output, also in $\mathcal{D}$. The results of this weight-free SNN learning approach are presented in the next section.

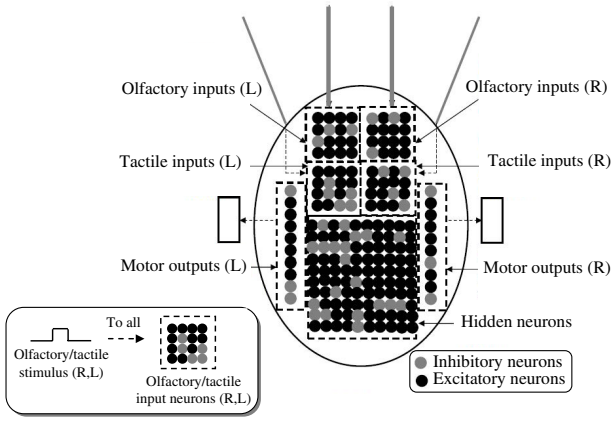


Fig. 9. Architecture of SNN model of insect CX, where *input* and *output* populations of neurons identified in the diagram receive sensory stimuli and produce motor control signals, respectively.

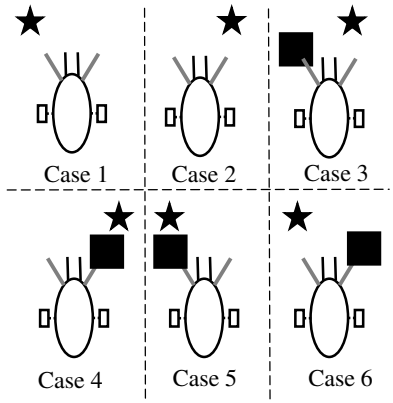


Fig. 10. Insect, terrain, and target locations for the six training cases used by the weight-free algorithm.

5. Simulation Results

The terrain used for the simulations is created via a cloud generated in Photoshop®, where the grayscale pixel value ranging from zero (white) to a maximum value $H_{\max} = 255$ (black) represents the elevation $H \in [0, H_{\max}]$. Three environments are considered: an obstacle-free arena, an S-maze, and an irregular terrain with various hills and narrow channels that are also visualized using a three-dimensional rendering in VRML. No prior knowledge of these environments is used in training the SNN model, and the virtual insect CX controls the insect motion based only on the sensory inputs described in Sec. 4.1. The SNN parameters, including the resting potential of 0.014 V, and firing threshold of 0.017 are summarized in the appendix, and adopted from experimental studies.[59]

Although, the insect can climb terrains of low elevation, flat areas are preferable in that climbing requires more energy expenditure. Initially, the SNN synaptic connections are assigned random weights and the naive insect has no control or navigation knowledge. After learning with the weight-free rule presented in Sec. 3, the virtual insect is placed in one of the three environments and navigates autonomously toward its target.

5.1. *Weight-free learning algorithm*

In order to circumvent computing the SNN error gradient (analytically or numerically), every iteration of the weight-free learning algorithm consists of a testing phase followed by a training phase. During the testing phase of the learning algorithm, olfactory/tacticle sensory stimuli are introduced by simulating one of the training cases in Fig. 10. In every case, the effect of small perturbations brought about by the training stimuli (Eq. (9), with $b_k = b_0$) on the SNN error is determined using Eq. (10). After the testing phase, a series of training stimuli designed according to Eqs. (11) and (12) are delivered to random pairs of input neurons, in order to induce synaptic plasticity such that the SNN error is minimized.

Examples of sensory and training stimuli delivered during the testing and training phases are shown in Fig. 11 for training Case 2. The SNN model in this example is characterized by the architecture in Table 4, with 184 neurons and random initial weights. As illustrated in Fig. 10, in Case 2 there are

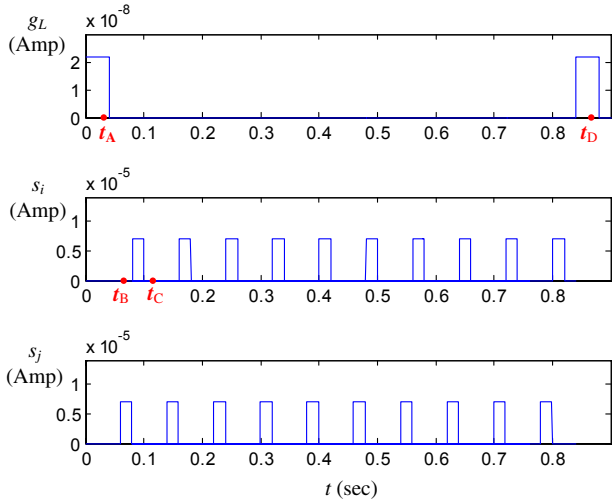


Fig. 11. Time history of left olfactory input, g_L, and training stimuli, s_i and s_j, delivered by the algorithm to two randomly chosen neurons i and j, respectively, for training Case 2. The response of all neurons at sample moments in time, t_A, t_B, t_C, and t_D, is plotted in Fig. 12.

no obstacles nearby and the target is to the right of the insect, thus the distance between the target and the left olfactory sensor is greater than that between the target and the right olfactory sensor. As a result, the stimulus is encoded as an olfactory sensory input g_L, plotted in Fig. 11, and delivered around time t_A to the olfactory input neurons in the virtual CX, located in the bottom left of the input layer (Fig. 1). Because the initial SNN synaptic weights are random, the insect motoneurons do not fire as a result of this sensory input, as demonstrated by the membrane potentials at t_A and t_B (Fig. 12). In order to perturb the synaptic strengths and infer the error-gradient sign change, at t_C two sequences of training stimuli (with $b_k = b_0$), plotted in Fig. 11, are delivered to a pair of input neurons during a time window $[0.05,\ 0.85]$. Upon termination of the training stimuli, at t_D, the olfactory sensory input g_L is delivered again to the SNN input layer revealing the new motoneuron activity shown in Fig. 12, which would translate into the insect turning right. Since in this case, the SNN response is close to the desired firing rate in $\mathcal{D}$, the algorithm moves on to a new iteration.

When the SNN response to a sensory input is significantly different from the desired one, testing is followed by another training phase that changes the offset parameters and number of pulses so as to minimize the SNN error. This situation is illustrated by

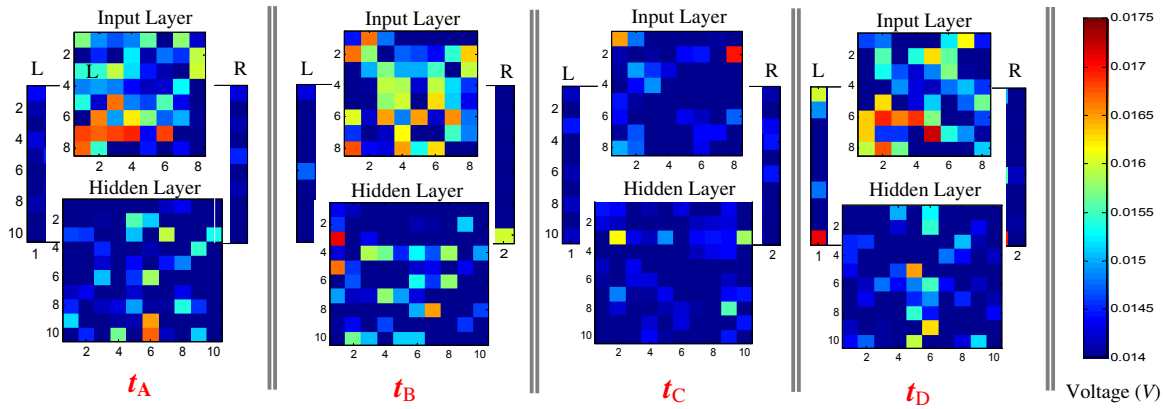


Fig. 12. SNN membrane potentials during testing and training phases, plotted at four sample times, t_A, t_B, t_C, and t_D, when subject to the sensory and training stimuli in Fig. 11.

another example in which the SNN model is tested and trained using Case 1, where there are no obstacles nearby but the target is to the left of the insect (Fig. 10). The neural activity in this example is illustrated using the raster plot in Fig. 13, where the insect CX is subject to an olfactory sensor input g_R and the corresponding activity of the right motoneurons results in a firing frequency f_R (solid line) that is very different from the desired value for Case 1, denoted by f_R^* (dashed line). In this case, the training phase consists of delivering training stimuli with

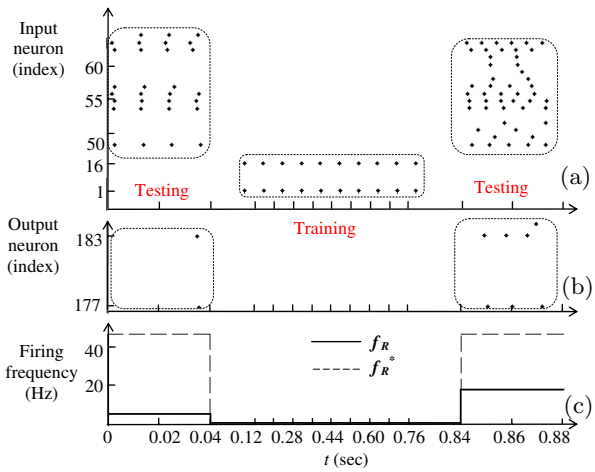


Fig. 13. Spike response during testing and training phases (a) for training Case 1, where the three phases are labeled by dashed squares, and, prior to training, the SNN firing frequency of the right motoneurons (b), f_R, differs from the desired firing frequency, f_R^* (c). After delivering the training stimuli (a), the firing frequency is improved (c).

$M_k = 10$ pulses to neurons $i = 1$ and $j = 16$, during a time window spanning $t_E = 0.4$ to $t_F = 0.84$, designed to minimize the SNN error in Eq. (10). Subsequently, when the same olfactory sensor input g_R is delivered again to the SNN input layer, it can be seen that the synaptic plasticity induced during the training phase has altered the SNN output response, such that its firing frequency f_R is closer to the desired frequency f_R^*.

The test-and-train routine is repeated iteratively over time, selecting a new pair of neurons at every epoch (k), such that the SNN error (Eq. (10)) decreases incrementally over time, as demonstrated by the training blue in Fig. 14. Because the gradient is never computed and the synaptic strengths are perturbed by delivering stimuli to randomly selected neurons, the error does not decrease monotonically. Also, training is conducted in batch mode, such that the SNN error is computed for all training cases in the database $\mathcal{D}$. As a result, the algorithm may slow down in the presence of local minima and experience large fluctuations in the error changes, as also incurred by classical resilient backpropagation in artificial NNs.[32] Ultimately, the algorithm terminates when the SNN error meets desired stopping criteria, e.g. is below a desired value $e_{\min}$, or ceases to decrease beyond a desired threshold over many consecutive epochs.

Because the computation required by this weight-free algorithm scales linearly with the number of hidden neurons ($\mathcal{I}$), the approach is applicable to SNNs with up to hundreds of neurons. Furthermore, the effectiveness of the algorithm in modifying the

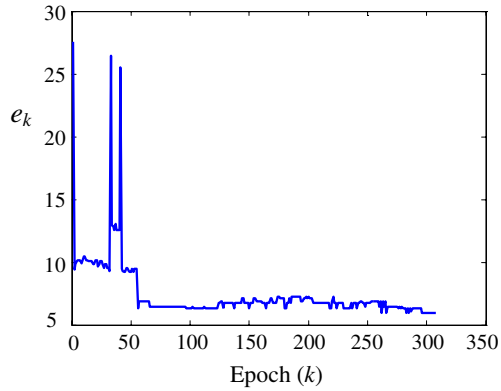


Fig. 14. Training blue with SNN error history.

Table 1. SNN output firing frequency comparison.

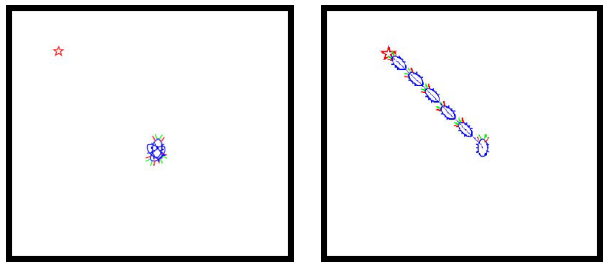
Case (No.)	Initial (Hz) f_L	Initial (Hz) f_R	Final (Hz) f_L	Final (Hz) f_R	Desired (Hz) f_L^*	Desired (Hz) f_R^*
1	0	5	22.5	41	20	46.7
2	0	12.5	40	15	46.7	20
3	7.5	15.0	80	25	100	20
4	2.5	27.5	35	60	46.7	73
5	10	12.5	60	40	73	46.7
6	0	32.5	30	90	20	100

high-level SNN response, without directly manipulating the synaptic strengths, can be illustrated by the results summarized in Table 1. These results show that the output firing frequencies of the final (trained) SNN model are very close to the desired firing frequencies for all six training cases in the database $\mathcal{D}$, even though the network is initialized poorly by the random approach, as indicated by its initial firing frequencies.

5.2. *Insect navigation*

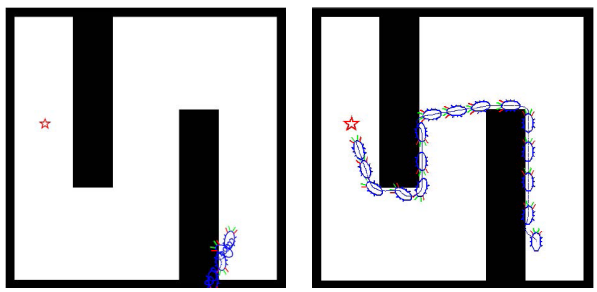
After learning, the virtual insect is placed in the three environments illustrated in Figs. 15–17, and compared to a naive insect controlled by the initial SNN model, with random synaptic strengths. It can be seen that, unlike the naive version, the trained SNN is capable of integrating information regarding the target location and terrain conditions and control the legs of the virtual insect such that it avoids elevated terrain, and reaches the target represented by a star. In particular, when placed in the obstacle-free environment, where only target information is relevant, the virtual insect navigates to the target using the path of shortest distance (Fig. 15(b)), while the naive insect rotates in place (Fig. 15(a)).

Similarly to the arena used for biological experiments in Fig. 5, the S-Maze involves a target and two partial walls that the trained insect is capable of navigating around efficiently to find the target on the other side (Fig. 16(b)). Instead, the naive insect moves in circles locally, responding to tactile



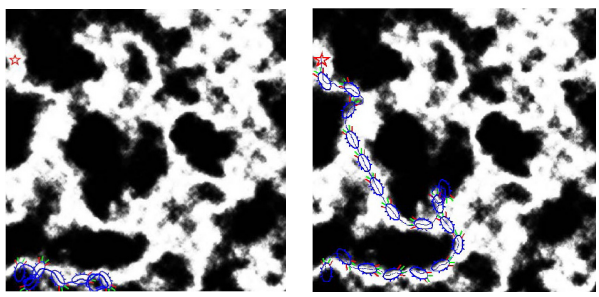
(a) (b)

Fig. 15. Comparison of insect trajectories for naive (random synaptic strengths) (a) and trained (b) SNNs in an obstacle-free arena.



(a) (b)

Fig. 16. Comparison of insect trajectories for naive (random synaptic strengths) (a) and trained (b) SNNs in an S-maze.



(a) (b)

Fig. 17. Comparison of insect trajectories for naive (random synaptic strengths) (a) and trained (b) SNNs in a complex terrain of variable elevation.

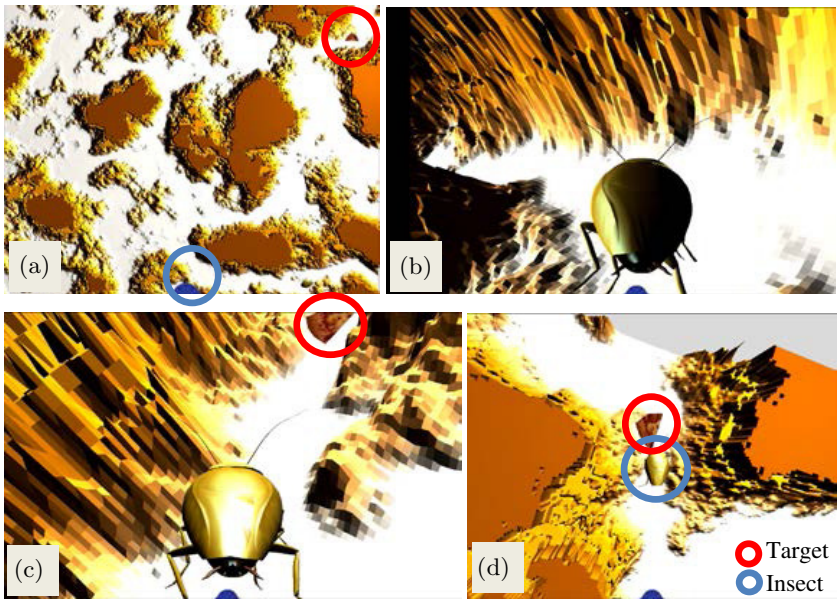


Fig. 18. Insect behavior in a complex terrain (a), visualized in VRML, illustrating the ability of the trained SNN to navigate through narrow passages (b)–(c), ultimately reaching the target (d).

information, but is unable to move past the first wall (Fig 16(a)). The irregular terrain characterized by hills and narrow channels represents a fairly complex landscape that would cause several path planning algorithms to remain stuck in dead ends or take long and inefficient detours. Instead, the trained SNN is capable to navigate efficiently and autonomously toward the goal, avoiding regions of high elevation and exploiting useful canyons between them (Fig. 17(b)). Although the naive insect can respond to stimuli from contact with obstacles, it is unable to reach the target (Fig. 17(a)).

When the simulation results are visualized in VRML (Fig. 18), the virtual insect also appears to display a realistic behavior (see insect video[60]). Besides validating the effectiveness of the proposed weight-free learning algorithm, these results also demonstrate that this spike-based formalism enables more direct and effective transfer of biological findings from animal experiments to synthetic platforms.

5.3. *Scalability and robustness*

The scalability and robustness of the SNN weight-free learning algorithm are investigated by considering the effects of noisy sensory inputs, modeled by Eq. (17), on three SNN architectures comprised of 11, 14, 184 neurons, and 819 neurons, described in Tables 2–5. Numerical tests show that, for noise-free sensory inputs, the weight-free learning algorithm can successfully train all four SNN architectures to properly control the virtual insect such that it can successfully and efficiently navigate all three environments. As an example, the root mean square error or training blue for the SNN with 819 neurons is plotted

Table 2. SNN architecture with 11 neurons.

Neuron Type	Excitatory	Inhibitory	Total
Input sensor neurons	8	0	8
Hidden neurons	1	0	1
Output motor neurons	2	0	2

Table 3. SNN architecture with 14 neurons.

Neuron Type	Excitatory	Inhibitory	Total
Input sensor neurons	6	2	8
Hidden neurons	4	0	4
Output motor neurons	2	0	2

Table 4. SNN architecture with 184 neurons.

Neuron Type	Excitatory	Inhibitory	Total
Input sensor neurons	49	15	64
Hidden neurons	80	20	100
Output motor neurons	14	6	20

Table 5. SNN architecture with 819 neurons.

Neuron Type	Excitatory	Inhibitory	Total
Input sensor neurons	124	20	144
Hidden neurons	524	101	625
Output motor neurons	44	6	50

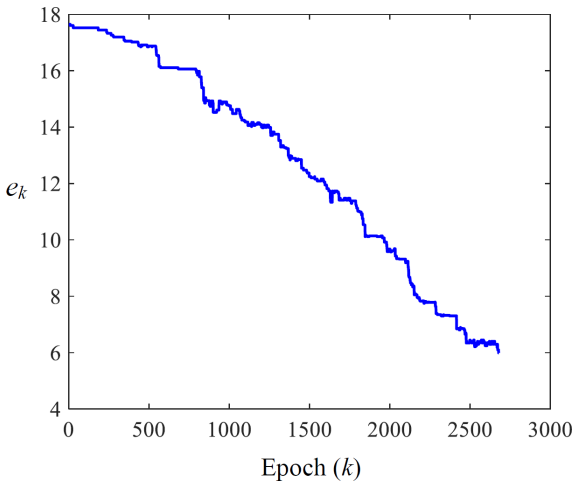


Fig. 19. Training blue for SNN with 819 neurons.

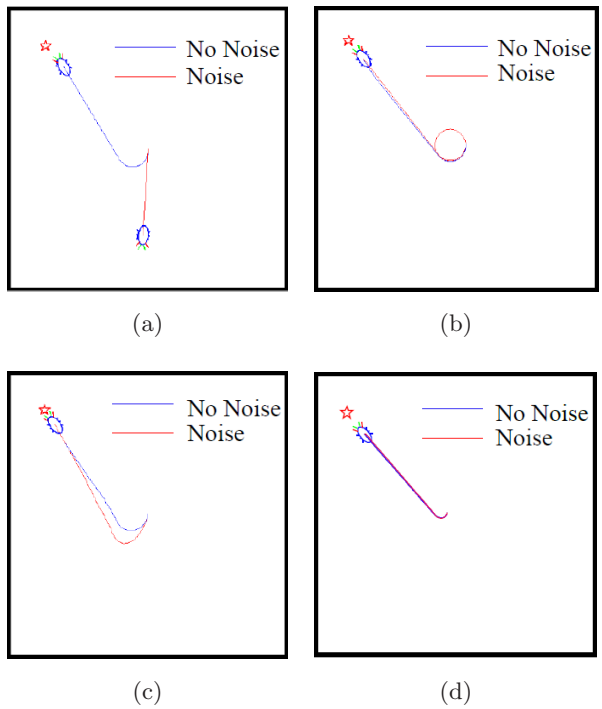


Fig. 20. Trained insect trajectories with or without sensory input noise for SNN with 11 neurons (a), 14 neurons (b), 184 neurons (c) and 819 neurons (d) when $\nu = 0.5$.

in Fig. 19. It can be seen that the error converges steadily to its minimum even for a large network size.

In the presence of noise, the performance robustness improves as the size of the SNN is increased. Figure 20 shows the trajectories of the trained virtual insect controlled by the three SNNs in an obstacle-free environment, with and without sensor noise. It can be seen that the 11-neuron SNN is unable to navigate to the target in the presence of the sensor noise with $\nu = 0.5$. In contrast, the 14-neuron SNN is capable to control the insect such that it reaches the target, but the insect first spins in a circle and only afterwards finds the shortest path. When compared to the 14-neuron SNN, the 184-neuron SNN shows a better performance, controlling the insect such that it corrects its orientation by turning around and then navigating to the target along the shortest path. The 819-neuron SNN shows the best performance of all four networks, because the noise has almost no impact on the insect trajectory (Fig. 20(d)). This is confirmed by plotting the insect distance from the target, as shown in Fig. 21 (where the final time is chosen heuristically). It can be seen that the largest (819-neuron) SNN performs better than all other SNNs, with or without sensor noise, and that in the presence of noise (with $\nu = 0.5$) the insect performance improves even more significantly with the SNN size.

To further understand SNN robustness to noisy inputs, the insect performance is analyzed as a function of noise amplitude, $\nu \in [0,\ 1]$. By varying ν in Eq. (17), random noise is produced with a magnitude up to $100 \cdot \nu$ (%) of the original sensor input ($g_{R,L}$ or $h_{R,L}$). Let success be defined by the ability to reach the target in a desired time window, in this case chosen as 5.6 (s) based on the initial target distance and maximum insect speed. If the insect is unable to navigate to the target in this time window or leaves the arena, failure is declared.

By plotting the successes and failures as a function of noise amplitude (ν) in Fig. 22, the SNN robustness to noisy sensor inputs is established in the limit. It can be seen that robustness increases with the size of the SNN, as the 819-neuron SNN is effective up to $\nu = 0.8$, while the 11-neuron SNN is only effective up to $\nu = 0.2$. This is attributed to the presence of a larger hidden layer capable of compensating for input errors. Surprisingly, the 14-neuron SNN can cope with noise up to $\nu = 0.5$, indicating

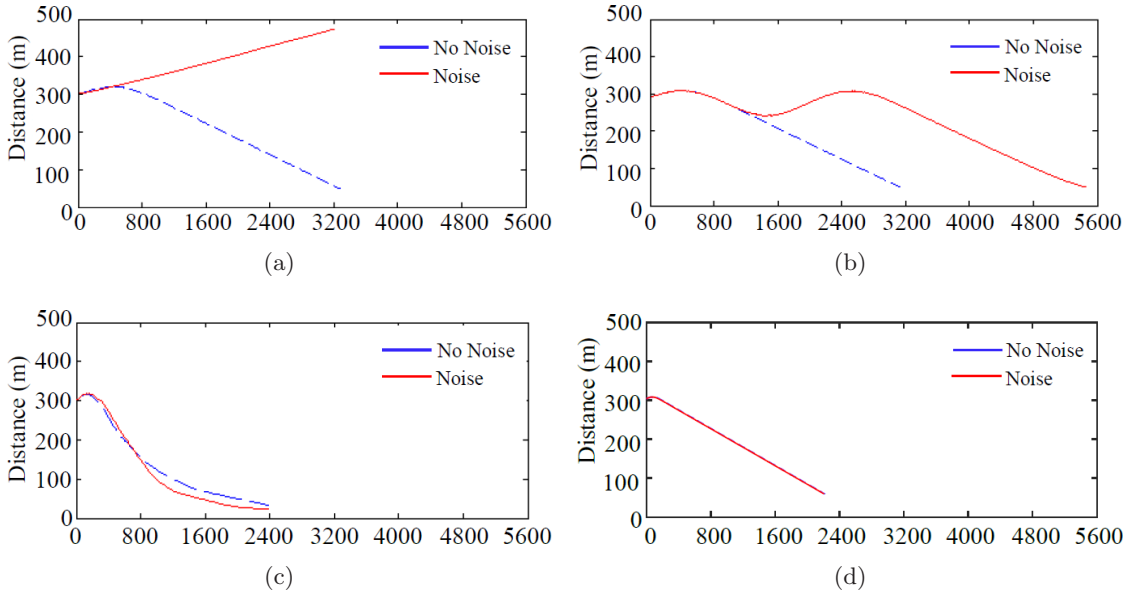


Fig. 21. Time history of insect distance from the target in the presence of sensor noise, when $\nu = 0.5$, using a trained SNN with 11 neurons (a), 14 neurons (b), 184 neurons (c), and 819 neurons (d).

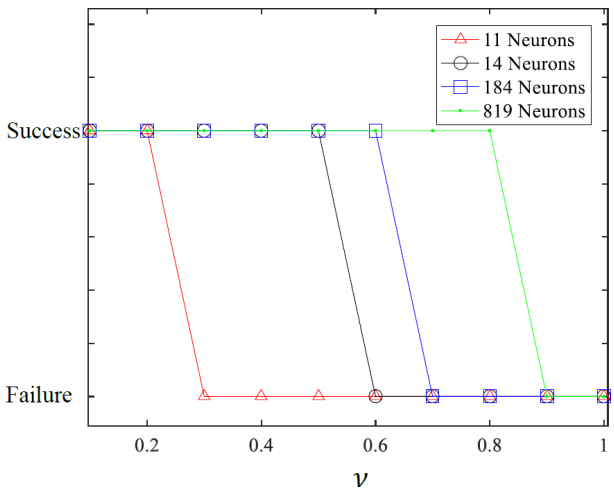


Fig. 22. Trained SNN effectiveness as a function of sensor noise level, for different numbers of neurons.

that its limiting performance is acceptable up to a very significant noise amplitude. However, as shown in Figs. 20 and 21, the insect trajectories in this case are far less efficient than the trajectories obtained with the 819-neuron SNN. Thus, a large SNN can not only provide robust performance in the limit, but also robust *optimal* performance as may be desirable in many biological and engineering systems.

The robustness of the weight-free learning algorithm to errors in stimulus delivery is tested by modifying the ideal stimulus s_i, defined in Eq. (9), to a square pulse with a Gaussian error, as follows. Consider a neuron j in a neighborhood of neuron i, where the scale of the neighborhood is represented by the variance of a Gaussian distribution, σ. Then, when the weight-free training algorithm delivers stimulus s_i (in Eq. (9)) to neuron i, any neuron j in its neighborhood is also stimulated with an amplitude,

$$w_j = w \sum_i \exp\left[-\frac{D(i,j)}{2\sigma^2}\right] \tag{19}$$

that decreases exponentially with the distance between neurons, $D(i,j)$, using the stimulus,

$$s_j(t) = w_j \sum_{l=1}^{M} \left[H\left(t - c_{i,l} + \frac{\beta}{2}\right) - H\left(t - c_{i,l} - \frac{\beta}{2}\right)\right]$$

which is delivered to all neighboring neurons. In this paper, the value $\sigma = 0.8$ is found to adequately represent the error found in neuronal cultures, causing multiple nearby neurons to fire, as shown in Fig. 23 for a neuron i with coordinates $(X, Y) = (2, 2)$. When the weight-free algorithm in Fig. 2 is implemented with the stimulus error in Eq. (19), the network performance can still be optimized reliably, as shown by the training blue in Fig. 24.

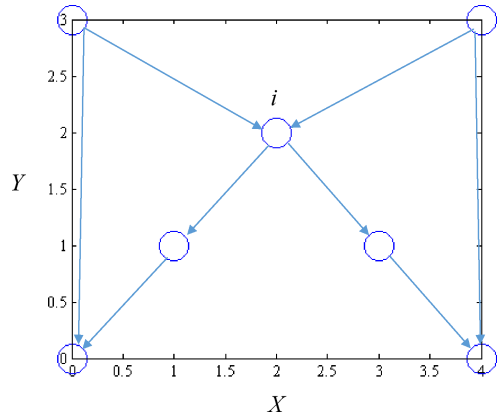


Fig. 23. Seven neurons influenced by a training stimulus with Gaussian error, as shown in Eq. (19), intended for neuron i at $(X, Y) = (2, 2)$.

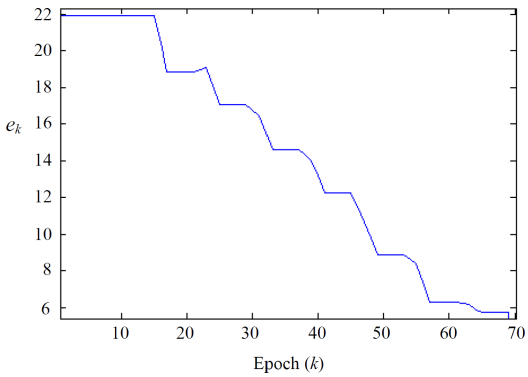


Fig. 24. Training blue in the presence of training stimulus error for the network in Fig. 23, with $\sigma = 0.8$.

6. Conclusions

This paper presents a weight-free SNN learning algorithm capable of manipulating plasticity at multiple scales of neural circuits. The advantage of a weight-free algorithm is that plasticity is altered by controlling cell activity and, thus, it is potentially realizable in live biological neuronal networks via optogenetics or intracellular stimulation. The idea of training neural networks by controlling cell activity, rather than by controlling the synaptic weights according to a learning rule, was recently demonstrated by the authors on small-scale SNNs. This idea is motivated by techniques, such as optogenetics, that are having a significant impact in the neuroscience field by delivering optical firing control with the precision and spatiotemporal resolution required for investigating information processing and plasticity in biological brains.

Even in the simplest organisms, however, brain circuits are characterized by hundreds of neurons responsible for integrating diverse stimuli and for controlling multiple functionalities. Because of their size, these large neuronal structures are believed to provide robustness and reconfigurability, characteristics that are also desirable in neuromorphic circuits and engineering applications of computational SNNs. This paper presents a new perturbative learning algorithm for scaling the weight-free paradigm up to networks with over 800 neurons. The proposed approach is scalable because it does not rely on gradient computations but, instead, delivers training stimuli based on consecutive changes in the sign of the gradient, similarly to resilient backpropagation.

The results in this paper show that the weight-free method is feasible for training SNNs on a larger scale than previously shown in the literature, and without any knowledge of their connectivity or synaptic strengths. The spike-based method by perturbation reduces the computational load and displays improved convergence, also maintaining a high level of accuracy and reliability in the chosen application of virtual insect control and navigation. This application is motivated by experimental studies in biology which have recently revealed that changes in the activity (e.g. firing rates) of individual units immediately precede changes in the firing rates of motoneurons responsible for locomotory behaviors such as walking speed, turning, and climbing.[33,34] The same wires used for recording cell activity can be used to stimulate the brain region to evoke altered behavior and plasticity. Hence, the weight-free learning algorithm is demonstrated on a virtual simulation of the aforementioned insect experiments to demonstrate its effectiveness at inducing higher level learning, as well as to illustrate how it might some day be used for controlling plasticity in experiments on living insects. The simulation results show that the trained SNN sensorimotor controller is capable of integrating different sensory stimuli and accomplish desired behavioral goals efficiently and reliably. As expected, the robustness of the trained SNN is shown to improve significantly with network size, providing near optimal performance even in the presence of large sources of sensor noise.

Future work will explore the applicability of the proposed algorithm to biological neuronal networks, for example to investigate and alter plasticity in the

CX of living insects *in vivo*, or to train live neurons in slice, where Channel Rhodopsin2 neurons are amenable to controlled cell firings via optogenetics. These applications will require increased computational efficiency and may involve alternate mechanisms of synaptic plasticity and temporal coding. Also, future work will explore how stimulation accuracy and precision may influence the effectiveness of the training algorithm, and how connectivity and genetically modified regions may affect the propagation of synaptic plasticity and resulting functional plasticity.

Acknowledgments

This work was supported by the National Science Foundation, under ECCS Grant 0925407.stop

Appendix A. Neuron Model

The LIF membrane potential can be modeled by the differential equation,[4,5,47]

$$\tau_m \frac{dV(t)}{dt} = -V(t) + R_m[I_{\text{stim}}(t) + I_{\text{syn}}(t) + I_{\text{noise}}], \tag{A.1}$$

where I_{stim} is the stimulus current, e.g. from an external stimulus such as blue light or a controlled input voltage, I_{syn} is the synaptic current from the presynaptic neurons, and I_{noise} is a random current input modeled as a Gaussian variable with zero mean and known variance. The passive membrane time constant, $\tau_m = C_m \cdot R_m$, and the membrane resistance, R_m, are assumed constant. When the membrane potential reaches a threshold value, $V > V_{\text{thresh}}$, the neuron fires (spikes), and the membrane potential returns to a resting potential E_{rest}. The neuron model parameters are defined as follows, with units in square brackets:

C_m [F] : Membrane capacity.
R_m [Ohm]: Membrane resistance.
V_{thresh} [V]: Membrane voltage threshold.
V_{resting} [V]: Resting membrane voltage.
V_{init} [V]: Initial voltage, $V(0)$, at time $t = 0$.
T_{refract} [s]: Length of absolute refractory period.
I_{noise} [A]: Standard deviation of current noise added at each integration time constant.
The units for SNN variables and parameters are shown in Table A.1.

Table A.1. Parameters and units.

Parameter	Units
$t, t_L, t_R, T_{\text{refract}}, t_r, \beta, c_{i,l}, b_0, b_k$	s
$x, y, L, d_L, d_R, H_{\max}, D(a,b), \lambda$	mm
α	Amp/mm
$h_L, g_L, h_R, g_R, s(t), S_i, S_i^0, I_{\text{noise}}, \omega$	Amp
v, v_L, v_R, η	mm/s
$f, f_L^*, f_R^*, f_L, f_R, z_i, f_L^0, f_R^0$	Hz
C_m	F
R_m	Ohm
$V_{\text{thresh}}, V_{\text{resting}}, V_{\text{init}}$	V

Table A.2. Parameter values.

Parameter	Simulation value
C_m (F)	3e-8
R_m (Ohm)	1e6
V_{thresh} (V)	0.017
V_{resting} (V)	0.014
V_{init} (V)	0
T_{refract} (s)	0.002
I_{noise} (A)	5e-12

Appendix B. STDP Model

In the STDP mechanism, the synaptic plasticity between two neurons changes according to timing of the pre- and post-synaptic neuron spikes denoted by t_{pre} and t_{post}, respectively. The STDP rule for synaptic weight change is,

$$\Delta w = \begin{cases} A_+ \exp(-(t_{\text{post}} - t_{\text{pre}})/\tau_+), \\ A_- \exp((t_{\text{post}} - t_{\text{pre}})/\tau_-), \end{cases} \tag{A.2}$$

where $A_+ = 2e-10, A_- = -1e-10$ are the amplitudes of the STDP synapses, and $\tau_+ = 0.0148, \tau_- = 0.0338$ are the time constants of exponential decay of the positive learning window for STDP.

References

1. K. Deisseroth, Optogenetics, *Nat. Methods* **8** (2011) 26–29.
2. National Academy of Engineering, NAE grand challenges for engineering (2015), Available at `http://www.engineeringchallenges.org/9109.aspx`.
3. S. Ferrari, B. Mehta, G. D. Muro, A. M. VanDongen and C. Henriquez, Biologically realizable reward-modulated hebbian training for spiking neural networks, in *Proc. Int. Joint Conf. Neural Networks*, Hong Kong (2008), pp. 1781–1787.
4. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *Int. J. Neural Syst.* **19**(4) (2009) 295–308.

5. Z. Wang, L. Guo and M. Adjouadi, A generalized leaky integrate-and-fire neuron model with fast implementation method, *Int. J. Neural Syst.* **24**(5) (2014) 1440004.
6. G. Zhang, H. Rong, F. Neri and M. J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, *Int. J. Neural Syst.* **24**(5) (2014) 1440006.
7. S. Shapero, M. Zhu, J. Hasler and C. Rozell, Optimal sparse approximation with integrate and fire neurons, *Int. J. Neural Syst.* **24**(5) (2014).
8. B. Strack, K. M. Jacobs and K. J. Cios, Simulating vertical and horizontal inhibition with short-term dynamics in a multi-column multi-layer model of neocortex, *Int. J. Neural Syst.* **24**(5) (2014) 1440002.
9. J. Friedrich, R. Urbanczik and W. Senn, Code-specific learning rules improve action selection by populations of spiking neurons, *Int. J. Neural Syst.* **24**(5) (2014) 1450002.
10. R. M. Douglas and G. V. Goddard, Long-term potentiation of the perforant path-granule cell synapse in the rat hippocampus, *Brain Res.* **86**(2) (1975) 205–215.
11. H. Markram, J. Lübke, M. Frotscher and B. Sakmann, Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps, *Science* **275**(5297) (1997) 213–215.
12. G.-Q. Bi and M.-M. Poo, Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type, *J. Neurosci.* **18**(24) (1998) 10464–10472.
13. S. Nobukawa and H. Nishimura, Enhancement of spike-timing-dependent plasticity in spiking neural systems with noise, *Int. J. Neural Syst.* **26**(5) (2015) 1550040.
14. C. Pennartz, Reinforcement learning by hebbian synapses with adaptive thresholds, *Neuroscience* **81**(2) (1997) 303–319.
15. R. Legenstein, S. M. Chase, A. B. Schwartz and W. Maass, A reward-modulated hebbian learning rule can explain experimentally observed network reorganization in a brain control task, *J. Neurosci.* **30**(25) (2010) 8400–8410.
16. S. Ferrari, B. Mehta, G. D. Muro, A. M. VanDongen and C. Henriquez, Biologically realizable reward-modulated hebbian training for spiking neural networks, *2008 IEEE Int. Joint Conf. Neural Networks* (*IJCNN*) (IEEE, 2008), pp. 1780–1786.
17. G. Foderaro, C. Henriquez and S. Ferrari, Indirect training of a spiking neural network for flight control via spike-timing-dependent synaptic plasticity, *2010 49th IEEE Conf. Decision and Control* (*CDC*) (IEEE, 2010), pp. 911–917.
18. R. V. Florian, Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity, *Neural Comput.* **19**(6) (2007) 1468–1502.
19. N. Kasabov, K. Dhoble, N. Nuntalid and G. Indiveri, Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition, *Neural Netw.* **41** (2013) 188–201.
20. S. B. Shrestha and Q. Song, Adaptive learning rate of spikeprop based on weight convergence analysis, *Neural Netw.* **63** (2015) 185–198.
21. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural Netw.* **22**(10) (2009) 1419–1431.
22. F. Ponulak, Resume-new supervised learning method for spiking neural networks, Institute of Control and Information Engineering, Poznan University of Technology (2005), Available at http://d1. cie. put. poznan. pl/˜ fp/research. html (2005).
23. R. V. Florian, The chronotron: A neuron that learns to fire temporally precise spike patterns, *PLOS ONE* **7**(8) (2012) e40233.
24. A. Taherkhani, A. Belatreche, Y. Li and L. P. Maguire, Dl-resume: A delay learning-based remote supervised method for spiking neurons, *IEEE Trans. Neural Netw. Learn. Syst.* **26**(12) (2015) 3137–3149.
25. A. Taherkhani, A. Belatreche, Y. Li and L. P. Maguire, Multi-DL-ReSuMe: Multiple neurons delay learning remote supervised method, *2015 Int. Joint Conf. Neural Netw.* (*IJCNN*) (IEEE, 2015), pp. 1–7.
26. A. Mohemmed, S. Schliebs, S. Matsuda and N. Kasabov, Training spiking neural networks to associate spatio-temporal input–output spike patterns, *Neurocomputing* **107** (2013) 3–10.
27. A. E.-D. El-Husseini, E. Schnell, S. Dakoji, N. Sweeney, Q. Zhou, O. Prange, C. Gauthier-Campbell, A. Aguilera-Moreno, R. A. Nicoll and D. S. Bredt, Synaptic strength regulated by palmitate cycling on psd-95, *Cell* **108**(6) (2002) 849–863.
28. X. Zhang, Z. Xu, C. Henriquez and S. Ferrari, Spike-based indirect training of a spiking neural network-controlled virtual insect, *2013 52th IEEE Annual Conf. Decision and Control* (*CDC*) (IEEE, 2013), pp. 6798–6805.
29. X. Zhang, G. Foderaro, C. Henriquez, A. VanDongen and S. Ferrari, A radial basis function spike model for indirect learning via integrate-and-fire sampling and reconstruction techniques, *Adv. Artif. Neural Syst.* **2012** (2012) 10.
30. D. Hu, X. Zhang, Z. Xu, S. Ferrari and P. Mazumder, Digital implementation of a spiking neural network (snn) capable of spike-timing-dependent plasticity (stdp) learning, *14th IEEE Int. Conf. Nanotechnology* (*IEEE-NANO*) (IEEE, 2014), pp. 873–876.
31. P. Mazumder, D. Hu, I. Ebong, X. Zhang, Z. Xu and S. Ferrari, Digital implementation of a virtual insect trained by spike-timing dependent plasticity, *Integr., VLSI J.* **54** (2016) 109–117.

32. M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: The rprop algorithm, *IEEE Int. Conf. Neural Networks, 1993* (IEEE, 1993), pp. 586–591.
33. R. Ritzmann and A. Büschges, Adaptive motor behavior in insects, *Curr. Opin. Neurobiol.* **17** (2007) 629–636.
34. R. E. Ritzmann, C. M. Harley, K. A. Daltorio, B. R. Tietz, A. J. Pollack, J. A. Bender, P. Guo, A. L. Horomanski, N. D. Kathman, C. Nieuwoudt *et al.*, Deciding which way to go: How do insects alter movements to negotiate barriers? *Front. Neurosci.* **6** (2012).
35. A. van Schaik, Building blocks for electronic spiking neural networks, *Neural Netw.* **14**(6) (2001) 617–628.
36. S. M. Bohte, H. L. Poutré and J. N. Kok, Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer rbf networks, *IEEE Trans. Neural Netw.* **13**(2) (2002) 426–435.
37. W. Maass, Networks of spiking neurons: the third generation of neural network models, *Neural Netw.* **10**(9) (1997) 1659–1671.
38. R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris Jr. *et al.*, Simulation of networks of spiking neurons: A review of tools and strategies, *J. Comput. Neurosci.* **23**(3) (2007) 349–398.
39. A. Destexhe, T. Bal, D. A. McCormick and T. J. Sejnowski, Ionic mechanisms underlying synchronized oscillations and propagating waves in a model of ferret thalamic slices, *J. Neurophysiol.* **76**(3) (1996) 2049–2070.
40. N. Kopell and G. B. Ermentrout, Coupled oscillators and the design of central pattern generators, *Math. Biosci.* **90**(1) (1988) 87–109.
41. J. L. Krichmar and G. M. Edelman, Brain-based devices for the study of nervous systems and the development of intelligent machines, *Artif. Life* **11**(1–2) (2005) 63–77.
42. E. M. Izhikevich *et al.*, Simple model of spiking neurons, *IEEE Trans. Neural Netw.* **14**(6) (2003) 1569–1572.
43. A. L. Hodgkin and A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. Physiol.* **117**(4) (1952) 500–544.
44. S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder and W. Lu, Nanoscale memristor device as synapse in neuromorphic systems, *Nano Lett.* **10**(4) (2010) 1297–1301.
45. M. R. Dranias, H. Ju, E. Rajaram and A. M. VanDongen, Short-term memory in networks of dissociated cortical neurons, *J. Neurosci.* **33**(5) (2013) 1940–1953.
46. W. Maass, T. Natschläger and H. Markram, Real-time computing without stable states: A new framework for neural computation based on perturbations, *Neural Comput.* **14**(11) (2002) 2531–2560.
47. W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity* (Cambridge University Press, Cambridge, UK, 2006).
48. W. Levy and O. Steward, Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus, *Neuroscience* **8**(4) (1983) 791–797.
49. C. D. Meliza and Y. Dan, Receptive-field modification in rat visual cortex induced by paired visual stimulation and single-cell spiking, *Neuron* **49**(2) (2006) 183–189.
50. V. B. Mountcastle, Modality and topographic properties of single neurons of cat's somatic sensory cortex, *J. Neurophysiol.* **20**(4) (1957) 408–434.
51. D. H. Hubel and T. N. Wiesel, Receptive fields of single neurons in the cat's striate cortex, *J. Physiol.* **148**(3) (1959) 574–591.
52. T. Natschläger, H. Markram and W. Maass, Computer models and analysis tools for neural microcircuits, in *Neuroscience Databases* (Springer, 2003), pp. 123–138.
53. N. Caporale and Y. Dan, Spike timing-dependent plasticity: A hebbian learning rule, *Annu. Rev. Neurosci.* **31** (2008) 25–46.
54. R. Kukillaya, J. Proctor and P. Holmes, Neuromechanical models for insect locomotion: Stability, maneuverability, and proprioceptive feedback, *CHAOS* **19**(026107) (2009).
55. J. P. Martin, P. Guo, L. Mu, C. M. Harley and R. E. Ritzmann, Central-complex control of movement in the freely walking cockroach, *Curr. Biol.* **25**(21) (2015) 2795–2803.
56. MathWorks, Virtual reality modeling language (vrml) (2016), Available at http://www.mathworks.com/help/sl3d/vrml.html.
57. A. Sharov, 3d insects (2016), Available at http://alexei.nfshost.com/3d/3dinsect.html.
58. V. Braitenberg and A. Schüz, *Cortex: Statistics and Geometry of Neuronal Connectivity* (Springer Science & Business Media, 2013).
59. H. Ju, M. R. Dranias, G. Banumurthy and A. M. VanDongen, Spatiotemporal memory is an intrinsic property of networks of dissociated cortical neurons, *J. Neurosci.* **35**(9) (2015) 4040–4051.
60. X. Zhang, SNN-controlled insect simulation in virtual reality modeling language (vrml) (2016), Available at https://youtu.be/J31QcxCNuiY.
61. Y. Zhang, Y. Wang, J. Jing and X. Wang, Sparse bayesian learning for obtaining sparsity of EEG frequency bands based feature vectors in motor imagery classification, *Int. J. Neural Syst.* **27**(2) (2017) 1650032.