Neurocomputing 380 (2020) 67–77

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Training multi-layer spiking neural networks using NormAD based spatio-temporal error backpropagation

Navin Anwani[a], Bipin Rajendran[b,*]

[a] Department of Electrical Engineering, Indian Institute of Technology Bombay, Mumbai 400076, India
[b] Department of Electrical and Computer Engineering, New Jersey Institute of Technology, NJ 07102, USA

ARTICLE INFO

Article history:
Received 14 December 2018
Revised 19 July 2019
Accepted 31 October 2019
Available online 6 November 2019

Communicated by Dr. Chenchen Liu

Keywords:
Supervised learning
Spiking neuron
Normalized approximate descent
Leaky integrate-and-fire
Multi-layer SNN
Spatio-temporal error backpropagation
NormAD
XOR problem

ABSTRACT

Spiking neural networks (SNNs) have garnered a great amount of interest for supervised and unsupervised learning applications. This paper deals with the problem of training multi-layer feedforward SNNs. The non-linear integrate-and-fire dynamics employed by spiking neurons make it difficult to train SNNs to generate desired spike trains in response to a given input. To tackle this, first the problem of training a multi-layer SNN is formulated as an optimization problem such that its objective function is based on the deviation in membrane potential rather than the spike arrival instants. Then, an optimization method named Normalized Approximate Descent (NormAD), hand-crafted for such non-convex optimization problems, is employed to derive the iterative synaptic weight update rule. Next, it is reformulated to efficiently train multi-layer SNNs, and is shown to be effectively performing spatio-temporal error backpropagation. The learning rule is validated by training 2-layer SNNs to solve a spike based formulation of the XOR problem as well as training 3-layer SNNs for generic spike based training problems. Thus, the new algorithm is a key step towards building deep spiking neural networks capable of efficient event-triggered learning.

1. Introduction

The human brain assimilates multi-modal sensory data and uses it to *learn* and perform complex cognitive tasks such as pattern detection, recognition, and completion. This ability is attributed to the dynamics of approximately 10^{11} neurons interconnected through a network of 10^{15} synapses in the human brain. This has motivated the study of neural networks in the brain and attempts to mimic their learning and information processing capabilities to create *smart learning machines*. Neurons, the fundamental information processing units in brain, communicate with each other by transmitting action potentials or spikes through their synapses. The process of learning in the brain emerges from synaptic plasticity viz., modification of strength of synapses triggered by spiking activity of corresponding neurons.

Spiking neurons are the third generation of artificial neuron models which closely mimic the dynamics of biological neurons. Unlike previous generations, both inputs and the output of a spiking neuron are signals in time. Specifically, these signals are point processes of spikes in the membrane potential of the neuron, also called a spike train. Spiking neural networks (SNNs) are computationally more powerful than previous generations of artificial neural networks as they incorporate temporal dimension to the information representation and processing capabilities of neural networks [1–3]. Owing to the incorporation of temporal dimension, SNNs naturally lend themselves for processing of signals in time such as audio, video, speech, etc. Information can be encoded in spike trains using temporal codes, rate codes, or population codes [4–6]. Temporal encoding uses exact spike arrival time for information representation and has far more representational capacity than rate code or population code [7]. However, one of the major hurdles in developing temporal encoding based applications of SNNs is the lack of efficient learning algorithms to train them with desired accuracy.

In recent years, there has been significant progress in the development of neuromorphic computing chips, which are specialized hardware implementations that emulate SNN dynamics inspired by the parallel, event-driven operation of the brain. Some notable examples are the TrueNorth chip from IBM [8], the Zeroth processor from Qualcomm [9], and the Loihi chip from Intel [10]. Hence, a breakthrough in learning algorithms for SNNs is apt and timely, to complement the progress of neuromorphic computing hardware.

The present success of deep learning based methods can be traced back to the breakthroughs in learning algorithms for second

* Corresponding author.
E-mail addresses: anwaninavin@gmail.com (N. Anwani), bipin@njit.edu (B. Rajendran).

https://doi.org/10.1016/j.neucom.2019.10.104
0925-2312/

generation artificial neural networks (ANNs) [11]. As we will discuss in Section 2, there has been work on learning algorithms for SNNs in the recent past, but those methods have not found wide acceptance as they suffer from computational inefficiencies and/or the lack of reliable and fast convergence. One of the main reasons for unsatisfactory performance of algorithms developed so far is that those efforts have been centered around adapting high-level concepts from learning algorithms for ANNs or from neuroscience and porting them to SNNs. In this work, we utilize properties specific to spiking neurons in order to develop a supervised learning algorithm for temporal encoding applications with spike-induced weight updates.

A supervised learning algorithm named *NormAD*, for single layer SNNs was proposed in [12]. For a spike domain training problem, it was demonstrated to converge at least an order of magnitude faster than the previous state-of-the-art. Recognizing the importance of multi-layer SNNs for supervised learning, in this paper we extend the idea to derive NormAD based supervised learning rule for multi-layer feedforward spiking neural networks. It is a spike-domain analogue of the error backpropagation rule commonly used for ANNs and can be interpreted to be a realization of spatio-temporal error backpropagation. The derivation comprises of first formulating the training problem for a multi-layer feedforward SNN as a non-convex optimization problem. Next, the Normalized Approximate Descent based optimization, introduced in [12], is employed to obtain an iterative weight adaptation rule. The new learning rule is successfully validated by employing it to train 2-layer feedforward SNNs for a spike domain formulation of the XOR problem and 3-layer feedforward SNNs for general spike domain training problems.

This paper is organized as follows. We begin with a summary of learning methods for SNNs documented in literature in Section 2. Section 3 provides a brief introduction to spiking neurons and the mathematical model of Leaky Integrate-and-Fire (LIF) neuron, also setting the notations we use later in the paper. Supervised learning problem for feedforward spiking neural networks is discussed in Section 4, starting with the description of a generic training problem for SNNs. Next we present a brief mathematical description of a feedforward SNN with one hidden layer and formulate the corresponding training problem as an optimization problem. Then Normalized Approximate Descent based optimization is employed to derive the spatio-temporal error backpropagation rule in Section 5. Simulation experiments to demonstrate the performance of the new learning rule for some exemplary supervised training problems are discussed in Section 6. Section 7 concludes the development with a discussion on directions for future research that can leverage the algorithm developed here towards the goal of realizing event-triggered deep spiking neural networks.

2. Related work

One of the earliest attempts to demonstrate supervised learning with spiking neurons is the SpikeProp algorithm [13]. However, it is restricted to single spike learning, thereby limiting its information representation capacity. SpikeProp was then extended in [14] to neurons firing multiple spikes. In these studies, the training problem was formulated as an optimization problem with the objective function in terms of the difference between desired and observed spike arrival instants and gradient descent was used to adjust the weights. However, since spike arrival time is a discontinuous function of the synaptic strengths, the optimization problem is non-convex and gradient descent is prone to local minima.

The biologically observed spike time dependent plasticity (STDP) has been used to derive weight update rules for SNNs in [15–17]. ReSuMe and DL-ReSuMe took cues from both STDP as well as the Widrow-Hoff rule to formulate a supervised learning algorithm [15,16]. Though these algorithms are biologically inspired, the training time necessary to converge is a concern, especially for real-world applications in large networks. The ReSuMe algorithm has been extended to multi-layer feedforward SNNs using backpropagation in [18].

Another notable spike-domain learning rule is PBSNLR [19], which is an offline learning rule for the spiking perceptron neuron (SPN) model using the perceptron learning rule. The PSD algorithm [20] uses Widrow-Hoff rule to empirically determine an equivalent learning rule for spiking neurons. The SPAN rule [21] converts input and output spike signals into analog signals and then applies the Widrow-Hoff rule to derive a learning algorithm. Further, it is applicable to the training of SNNs with only one layer. The SWAT algorithm [22] uses STDP and BCM rule to derive a weight adaptation strategy for SNNs. The Normalized Spiking Error Back-Propagation (NSEBP) method proposed in [23] is based on approximations of the simplified Spike Response Model for the neuron. The multi-STIP algorithm proposed in [24] defines an inner product for spike trains to approximate a learning cost function. As opposed to the above approaches which attempt to develop weight update rules for fixed network topologies, there are also some efforts in developing feed-forward networks based on evolutionary algorithms where new neuronal connections are progressively added and their weights and firing thresholds updated for every class label in the database [25,26].

Recently, an algorithm to learn precisely timed spikes using a leaky integrate-and-fire neuron was presented in [27]. The algorithm converges only when a synaptic weight configuration to the given training problem exists, and can not provide a close approximation, if the exact solution does not exist. To overcome this limitation, another algorithm to learn spike sequences with finite precision is also presented in the same paper. It allows a window of width ϵ around the desired spike instant within which the output spike could arrive and performs training only on the first deviation from such desired behavior. While it mitigates the non-linear accumulation of error due to interaction between output spikes, it also restricts the training to just one discrepancy per iteration. Backpropagation for training deep networks of LIF neurons has been presented in [28], derived assuming an impulse-shaped post-synaptic current kernel and treating the discontinuities at spike events as noise. It presents remarkable results on MNIST and N-MNIST benchmarks using rate coded outputs, while in the present work we are interested in training multi-layer SNNs with temporally encoded outputs i.e., representing information in the timing of spikes.

Many previous attempts to formulate supervised learning as an optimization problem employ an objective function formulated in terms of the difference between desired and observed spike arrival times [13,14,29,30]. We will see in Section 3 that a leaky integrate-and-fire (LIF) neuron can be described as a non-linear spatio-temporal filter, spatial filtering being the weighted summation of the synaptic inputs to obtain the total incoming synaptic current and temporal filtering being the leaky integration of the synaptic current to obtain the membrane potential. Thus, it can be argued that in order to train multi-layer SNNs, we would need to backpropagate error in space as well as in time, and as we will see in Section 5, it is indeed the case for the proposed algorithm. Note that while the membrane potential can directly control the output spike timings, it is also relatively more tractable through synaptic inputs and weights compared to spike timing. This observation is leveraged to derive a spaio-temporal error backpropagation algorithm by treating supervised learning as an optimization problem, with the objective function formulated in terms of the membrane potential.

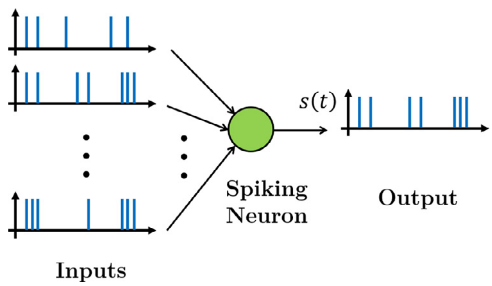


Fig. 1. Illustration of spike based information representation: a spiking neuron assimilates multiple input spike trains to generate an output spike train. Figure adapted from [12].

3. Spiking neurons

Spiking neurons are simplified models of biological neurons e.g., the Hodgkin–Huxley equations describing the dependence of membrane potential of a neuron on its membrane current and conductivity of ion channels [31]. A spiking neuron is modeled as a multi-input system that receives inputs in the form of sequences of spikes, which are then transformed to analog current signals at its input synapses. The synaptic currents are superposed inside the neuron and the result is then transformed by its non-linear integrate-and-fire dynamics to a membrane potential signal with a sequence of stereotyped events in it, called action potentials or spikes. Despite the continuous-time variations in the membrane potential of a neuron, it communicates with other neurons through the synaptic connections by chemically inducing a particular current signal in the post-synaptic neuron each time it spikes. Hence, the output of a neuron can be completely described by the time sequence of spikes issued by it. This is called *spike based information representation* and is illustrated in Fig. 1. The output, also called a spike train, is modeled as a point process of spike events. Though the internal dynamics of an individual neuron is straightforward, a network of neurons can exhibit complex dynamical behaviors. The processing power of neural networks is attributed to the massively parallel synaptic connections among neurons.

3.1. Synapse

The communication between any two neurons is spike induced and is accomplished through a directed connection between them known as a synapse. In the cortex, each neuron can receive spike-based inputs from thousands of other neurons. If we model an incoming spike at a synapse as a unit impulse, then the behavior of the synapse to translate it to an analog current signal in the post-synaptic neuron can be modeled by a linear time invariant system with transfer function $w\alpha(t)$. Thus, if a pre-synaptic neuron issues a spike at time t^f, the post-synaptic neuron receives a current $i(t) = w\alpha(t - t^f)$. Here the waveform $\alpha(t)$ is known as the post-synaptic current kernel and the scaling factor w is called the weight of the synapse. The weight varies from synapse-to-synapse and is representative of its conductance, whereas $\alpha(t)$ is independent of synapse and is commonly modeled as

$$\alpha(t) = [\exp(-t/\tau_1) - \exp(-t/\tau_2)]u(t), \tag{1}$$

where $u(t)$ is the Heaviside step function and $\tau_1 > \tau_2$. Note that the synaptic weight w can be positive or negative, depending on which the synapse is said to be excitatory or inhibitory respectively. Further, we assume that the synaptic currents do not depend on the membrane potential or reversal potential of the post-synaptic neuron.

Let us assume that a neuron receives inputs from n synapses and spikes arrive at the ith synapse at instants $t_1^i, t_2^i, \ldots$. Then, the input signal at the i^{th} synapse (before scaling by synaptic weight w_i) is given by the expression

$$c_i(t) = \sum_f \alpha(t - t_f^i). \tag{2}$$

spike due to one(ith) synapse firing at various times(tf

The synaptic weights of all input synapses to a neuron are usually represented in a compact form as a weight vector $\boldsymbol{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}^T$, where w_i is the weight of the i^{th} synapse. The synaptic weights perform spatial filtering over the input signals resulting in an aggregate synaptic current received by the neuron:

$$I(t) = \boldsymbol{w}^T \boldsymbol{c}(t), \tag{3}$$

where $\boldsymbol{c}(t) = \begin{bmatrix} c_1(t) & c_2(t) & \cdots & c_n(t) \end{bmatrix}^T$. A simplified illustration of the role of synaptic transmission in overall spike based information processing by a neuron is shown in Fig. 2, where an incoming spike train at a synaptic input is translated to an analog current with an amplitude depending on weight of the synapse. The resultant current at the neuron from all its upstream synapses is transformed non-linearly to generate its membrane potential with instances of spikes viz., sudden surge in membrane potential followed by an immediate drop.

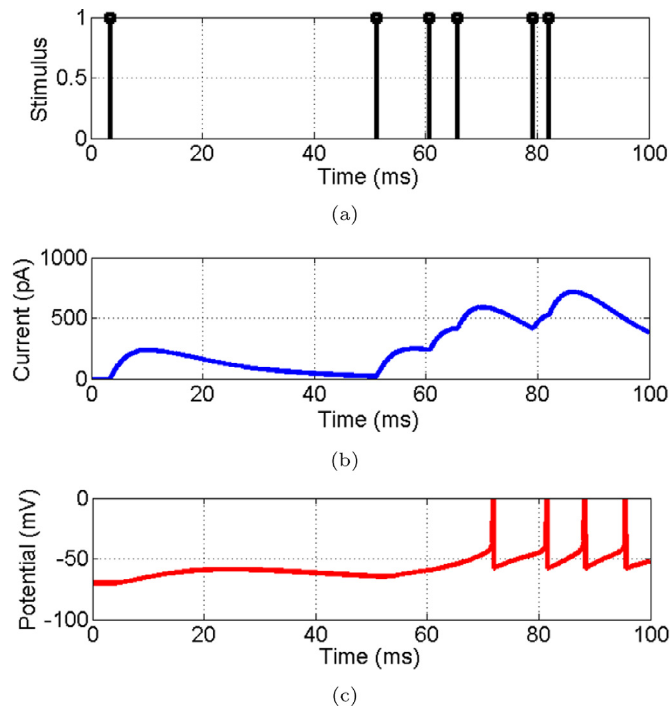


Fig. 2. Illustration of a simplified synaptic transmission and neuronal integration model: (a) exemplary spikes (stimulus) arriving at a synapse, (b) the resultant current being fed to the neuron through the synapse and (c) the resultant membrane potential of the post-synaptic neuron.

Synaptic plasticity

The response of a neuron to stimuli greatly depends on the conductance of its input synapses. Conductance of a synapse (the synaptic weight) changes based on the spiking activity of the corresponding pre- and post-synaptic neurons. A neural network's ability to learn is attributed to this activity dependent synaptic plasticity. Taking cues from biology, we will also constrain the learning algorithm we develop to have spike-induced synaptic weight updates.

3.2. Leaky integrate-and-fire (LIF) neuron

In leaky integrate-and-fire (LIF) model of spiking neurons, the transformation from aggregate input synaptic current $I(t)$ to the resultant membrane potential $V(t)$ is governed by the following differential equation and reset condition [32]:

$$C_m \frac{dV(t)}{dt} = -g_L(V(t) - E_L) + I(t),$$
$$V(t) \longrightarrow E_L \quad \text{when } V(t) \geq V_T. \tag{4}$$

Here, C_m is the membrane capacitance, E_L is the leak reversal potential, and g_L is the leak conductance. If $V(t)$ exceeds the threshold potential V_T, a spike is said to have been issued at time t. The expression $V(t) \longrightarrow E_L$ when $V(t) \geq V_T$ denotes that $V(t)$ is reset to E_L when it exceeds the threshold V_T. Assuming that the neuron issued its latest spike at time t_l, Eq. (4) can be solved for any time instant $t > t_l$, until the issue of the next spike, with the initial condition $V(t_l) = E_L$ as

$$V(t) \;=\; E_L + (I(t)u(t - t_l)) * h(t)$$
$$V(t) \longrightarrow E_L \quad \text{when } V(t) \geq V_T, \tag{5}$$

where '*' denotes linear convolution and

$$h(t) = \frac{1}{C_m} \exp(-t/\tau_L) u(t), \tag{6}$$

to remove spikes before tl

with $\tau_L = C_m/g_L$ is the neuron's leakage time constant. Note from Eq. (5) that the aggregate synaptic current $I(t)$ obtained by spatial filtering of all the input signals is first gated with a unit step located at $t = t_l$ and then fed to a leaky integrator with impulse response $h(t)$, which performs temporal filtering. So, the LIF neuron acts as a non-linear spatio-temporal filter and the non-linearity is a result of the reset at every spike.

Using Eqs. (3) and (5), the membrane potential can be represented in a compact form as

$$V(t) = E_L + \boldsymbol{w}^T \boldsymbol{d}(t), \tag{7}$$

where $\boldsymbol{d}(t) = \begin{bmatrix} d_1(t) & d_2(t) & \cdots & d_n(t) \end{bmatrix}^T$ and

$$d_i(t) = (c_i(t)u(t - t_l)) * h(t). \tag{8}$$

From Eq. (7), it is evident that $\boldsymbol{d}(t)$ carries all the information about the input necessary to determine the membrane potential. It should be noted that $\boldsymbol{d}(t)$ depends on weight vector $\boldsymbol{w}$, since $d_i(t)$ for each i depends on the last spiking instant t_l, which in turn is dependent on the weight vector $\boldsymbol{w}$.

The neuron is said to have spiked only when the membrane potential $V(t)$ reaches the threshold V_T. Hence, minor changes in the weight vector $\boldsymbol{w}$ may eliminate an already existing spike or introduce new spikes. Thus, spike arrival time t_l is a discontinuous function of $\boldsymbol{w}$. Therefore, Eq. (7) implies that $V(t)$ is also discontinuous in weight space. Supervised learning problem for SNNs is generally framed as an optimization problem with the cost function described in terms of the spike arrival time or membrane potential. However, the discontinuity of spike arrival time as well as $V(t)$ in weight space renders the cost function discontinuous and making the optimization problem non-convex. Commonly used steepest descent methods can not be applied to solve such non-convex optimization problems. In this paper, we extend the optimization method named *Normalized Approximate Descent*, introduced in [12] for single layer SNNs to multi-layer SNNs.

3.3. Refractory period

After issuing a spike, biological neurons can not immediately issue another spike for a short period of time. This short duration of inactivity is called the absolute refractory period (Δ_{abs}). This aspect of spiking neurons has been omitted in the above discussion for simplicity, but can be easily incorporated in our model by replacing t_l with $(t_l + \Delta_{abs})$ in the equations above.

Armed with a compact representation of membrane potential in Eq. (7), we are now set to derive a synaptic weight update rule to accomplish supervised learning with spiking neurons.

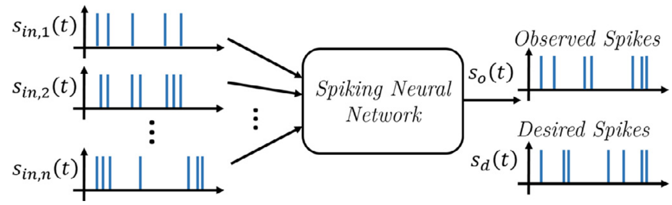


Fig. 3. Spike domain training problem: Given a set of n input spike trains fed to the SNN through its n inputs, determine the weights of synaptic connections constituting the SNN so that the observed output spike train is as close as possible to the given desired spike train.

4. Supervised learning using feedforward SNNs

Supervised learning is the process of obtaining an approximate model of an unknown system based on available training data, where the training data comprises of a set of inputs to the system and corresponding outputs. The learned model should not only fit to the training data well but should also generalize well to unseen samples from the same input distribution. The first requirement viz. to obtain a model so that it best fits the given training data is called training problem. Next we discuss the training problem in spike domain, which is a stepping stone towards solving the more constrained supervised learning problem.

4.1. Training problem

A canonical training problem for a spiking neural network is illustrated in Fig. 3. There are n inputs to the network such that $s_{in,i}(t)$ is the spike train fed at the ith input. Let the desired output spike train corresponding to this set of input spike trains be given in the form of an impulse train as

$$s_d(t) = \sum_{i=1}^{f} \delta(t - t_d^i). \tag{9}$$

Here, $\delta(t)$ is the Dirac delta function and $t_d^1, t_d^2, \ldots, t_d^f$ are the desired spike arrival instants over a duration T, also called an epoch. The aim is to determine the weights of the synaptic connections constituting the SNN so that its output $s_o(t)$ in response to the given input is as close as possible to the desired spike train $s_d(t)$.

NormAD based iterative synaptic weight adaptation rule was proposed in [12] for training single layer feedforward SNNs. However, there are many systems which can not be modeled by any possible configuration of single layer SNNs and necessarily require a multi-layer SNN. Hence, now we aim to obtain a supervised learning rule for multi-layer spiking neural networks. The change in weights in a particular iteration of training can be based on the given set of input spike trains, desired output spike train, and the corresponding observed output spike train. Also, the weight adaptation rule should be constrained to have spike induced weight updates for computational efficiency. For simplicity, we will first derive the weight adaptation rule for training a feedforward SNN with one hidden layer and then state the general weight adaptation rule for feedforward SNN with an arbitrary number of layers.

Performance metric

Training performance can be assessed by the correlation between desired and observed outputs. It can be quantified in terms

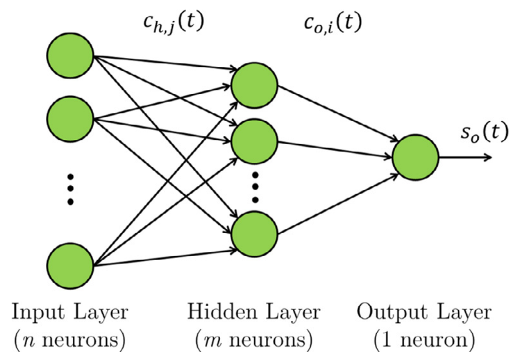


Fig. 4. Feedforward SNN with one hidden layer ($n \to m \to 1$) also known as 2-layer feedforward SNN.

of the cross-correlation between low-pass filtered versions of the two spike trains. The correlation metric which was introduced in [33] and is commonly used in characterizing the spike based learning efficiency [12,34] is defined as

$$C = \frac{\langle L(s_d(t)), L(s_o(t))\rangle}{\|L(s_d(t))\| \cdot \|L(s_o(t))\|}. \tag{10}$$

Here, $L(s(t))$ is the low-pass filtered spike train $s(t)$ obtained by convolving it with a one-sided falling exponential i.e.,

$$L(s(t)) = s(t) * (\exp(-t/\tau_{LP})u(t)),$$

with $\tau_{LP} = 5\,\text{ms}$.

4.2. Feedforward SNN with one hidden layer

A fully connected feedforward SNN with one hidden layer is shown in Fig. 4. It has n neurons in the input layer, m neurons in the hidden layer and 1 in the output layer. It is also called a 2-layer feedforward SNN, since the neurons in input layer provide spike based encoding of sensory inputs and do not actually implement the neuronal dynamics. We denote this network as a $n \to m \to 1$ feedforward SNN. This basic framework can be extended to the case where there are multiple neurons in the output layer or the case where there are multiple hidden layers. The weight of the synapse from the jth neuron in the input layer to the i^{th} neuron in the hidden layer is denoted by $w_{h,\,ij}$ and that of the synapse from the ith neuron in the hidden layer to the neuron in output layer is denoted by $w_{o,\,i}$. All input synapses to the ith neuron in the hidden layer can be represented compactly as an n-dimensional vector $\boldsymbol{w}_{h,i} = \begin{bmatrix} w_{h,i1} & w_{h,i2} & \cdots & w_{h,in}\end{bmatrix}^T$. Similarly input synapses to the output neuron are represented as an m-dimensional vector $\boldsymbol{w}_o = \begin{bmatrix} w_{o,1} & w_{o,2} & \cdots & w_{o,m}\end{bmatrix}^T$.

Let $s_{in,j}(t)$ denote the spike train fed by the jth neuron in input layer to neurons in hidden layer. Hence, from Eq. (2), the signal fed to the neurons in the hidden layer from the jth input (before scaling by synaptic weight) $c_{h,j}(t)$ is given as

$$c_{h,j}(t) = s_{in,j}(t) * \alpha(t). \tag{11}$$

every spike is modeled as a shifted version of alpha .....sumed up as an convolution

Assuming $t^{last}_{h,i}$ as the latest spiking instant of the ith neuron in the hidden layer, define $\boldsymbol{d}_{h,\,i}(t)$ as

$$\boldsymbol{d}_{h,i}(t) = \left(\boldsymbol{c}_h(t)u\left(t - t^{last}_{h,i}\right)\right) * h(t), \tag{12}$$

where $\boldsymbol{c}_h(t) = \begin{bmatrix} c_{h,1} & c_{h,2} & \cdots & c_{h,n}\end{bmatrix}^T$. From Eq. (7), membrane potential of the ith neuron in hidden layer is given as

$$V_{h,i}(t) = E_L + \boldsymbol{w}^T_{h,i}\boldsymbol{d}_{h,i}(t). \tag{13}$$

Accordingly, let $s_{h,i}(t)$ be the spike train produced at the ith neuron in the hidden layer. The corresponding signal fed to the output neuron is given as

$$c_{o,i}(t) = s_{h,i}(t) * \alpha(t). \tag{14}$$

Defining $\boldsymbol{c}_o(t) = \begin{bmatrix} c_{o,1} & c_{o,2} & \cdots & c_{o,m}\end{bmatrix}^T$ and denoting the latest spiking instant of the output neuron by t^{last}_o we can define

$$\boldsymbol{d}_o(t) = \left(\boldsymbol{c}_o(t)u\left(t - t^{last}_o\right)\right) * h(t). \tag{15}$$

Hence, from Eq. (7), the membrane potential of the output neuron is given as

$$V_o(t) = E_L + \boldsymbol{w}^T_o\boldsymbol{d}_o(t) \tag{16}$$

and the corresponding output spike train is denoted $s_o(t)$.

4.3. Mathematical formulation of the training problem

To solve the training problem employing an $n \to m \to 1$ feedforward SNN, effectively we need to determine synaptic weights $W_h = \begin{bmatrix} \boldsymbol{w}_{h,1} & \boldsymbol{w}_{h,2} & \cdots & \boldsymbol{w}_{h,m}\end{bmatrix}^T$ and $\boldsymbol{w}_o$ constituting its synaptic connections, so that the output spike train $s_o(t)$ is as close as possible to the desired spike train $s_d(t)$ when the SNN is excited with the given set of input spike trains $s_{in,i}(t)$, $i \in \{1, 2, \ldots, n\}$. Let $V_d(t)$ be the corresponding ideally desired membrane potential of the output neuron, such that the respective output spike train is $s_d(t)$. Also, for a particular configuration W_h and $\boldsymbol{w}_o$ of synaptic weights of the SNN, let $V_o(t)$ be the observed membrane potential of the output neuron in response to the given input and $s_o(t)$ be the respective output spike train. We define the cost function for training as

$$J(W_h, \boldsymbol{w}_o) = \frac{1}{2}\int_0^T \left(\Delta V_{d,o}(t)\right)^2 |e(t)|dt, \tag{17}$$

where

$$\Delta V_{d,o}(t) = V_d(t) - V_o(t) \tag{18}$$

and

$$e(t) = s_d(t) - s_o(t). \tag{19}$$

That is, the cost function is determined by the difference $\Delta V_{d,o}(t)$, only at the instants in time where there is a discrepancy between the desired and observed spike trains of the output neuron. Thus, the training problem can be expressed as following optimization problem:

$$\begin{aligned} \min \quad & J(W_h, \boldsymbol{w}_o) \\ \text{s.t.} \quad & W_h \in \mathbb{R}^{m\times n}, \boldsymbol{w}_o \in \mathbb{R}^m \end{aligned} \tag{20}$$

Note that the optimization with respect to $\boldsymbol{w}_o$ is same as training a single layer SNN, provided the spike trains from neurons in the hidden layer are known. In addition, we need to derive the weight adaptation rule for synapses feeding the hidden layer viz., the weight matrix W_h, such that spikes in the hidden layer are most suitable to generate the desired spikes at the output. The cost function is dependent on the membrane potential $V_o(t)$, which is discontinuous with respect to $\boldsymbol{w}_o$ as well as W_h. Hence the optimization problem (20) is non-convex and susceptible to local minima when solved with steepest descent algorithm.

5. NormAD based spatio-temporal error backpropagation

In this section we apply Normalized Approximate Descent to the optimization problem (20) to derive a spike domain analogue of error backpropagation. First we derive the training algorithm for SNNs with single hidden layer, and then we provide its generalized form to train feedforward SNNs with arbitrary number of hidden layers.

5.1. NormAD – Normalized Approximate Descent

Following the approach introduced in [12], we use three steps viz., (i) stochastic gradient descent, (ii) normalization and (iii) gradient approximation, as elaborated below to solve the optimization problem (20).

5.1.1. Stochastic gradient descent

Instead of trying to minimize the aggregate cost over the epoch, we try to minimize the instantaneous contribution to the cost at each instant t for which $e(t) \neq 0$, independent of that at any other instant and expect that it minimizes the total cost $J(W_h, \boldsymbol{w}_o)$. The instantaneous contribution to the cost at time t is denoted as $J(W_h, \boldsymbol{w}_o, t)$ and is obtained by restricting the limits of integral in Eq. (17) to an infinitesimally small interval around time t:

$$J(W_h, \boldsymbol{w}_o, t) = \begin{cases} \frac{1}{2}\left(\Delta V_{d,o}(t)\right)^2 & e(t) \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

Thus, using stochastic gradient descent, the prescribed change in any weight vector $\boldsymbol{w}$ at time t is given as:

$$\Delta \boldsymbol{w}(t) = \begin{cases} -k(t) \cdot \nabla_{\boldsymbol{w}} J(W_h, \boldsymbol{w}_o, t) & e(t) \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Here $k(t)$ is a time dependent learning rate. The change aggregated over the epoch is, therefore

$$\begin{aligned} \Delta \boldsymbol{w} &= \int_{t=0}^{T} -k(t) \cdot \nabla_{\boldsymbol{w}} J(W_h, \boldsymbol{w}_o, t) \cdot |e(t)| dt \\ &= \int_{t=0}^{T} k(t) \cdot \Delta V_{d,o}(t) \cdot \nabla_{\boldsymbol{w}} V_o(t) \cdot |e(t)| dt. \end{aligned} \tag{22}$$

Minimizing the instantaneous cost only for time instants when $e(t) \neq 0$ also renders the weight updates spike-induced i.e., it is non-zero only when there is either an observed or a desired spike in the output neuron.

5.1.2. Normalization

Observe that in Eq. (22), the gradient of membrane potential $\nabla_{\boldsymbol{w}} V_o(t)$ is scaled with the error term $\Delta V_{d,o}(t)$, which serves two purposes. First, it determines the sign of the weight update at time t and second, it gives more importance to weight updates corresponding to the instants with higher magnitude of error. But $V_d(t)$ and hence error $\Delta V_{d,o}(t)$ is not known. Also, dependence of the error on $\boldsymbol{w}_{h,\,i}$ is non-linear, so we eliminate the error term $\Delta V_{d,o}(t)$ for neurons in hidden layer by choosing $k(t)$ such that

$$|k(t) \cdot \Delta V_{d,o}(t)| = r_h, \tag{23}$$

where r_h is a constant. From Eq. (22), we obtain the weight update for the ith neuron in the hidden layer as

$$\Delta \boldsymbol{w}_{h,i} = r_h \int_{t=0}^{T} \nabla_{\boldsymbol{w}_{h,i}} V_o(t) e(t) dt, \tag{24}$$

since $\mathrm{sgn}\left(\Delta V_{d,o}(t)\right) = \mathrm{sgn}\left(e(t)\right)$. For the output neuron, we eliminate the error term by choosing $k(t)$ such that

$$\|k(t) \cdot \Delta V_{d,o}(t) \cdot \nabla_{\boldsymbol{w}_o} V_o(t)\| = r_o,$$

where r_o is a constant. From Eq. (22), we get the weight update for the output neuron as

$$\Delta \boldsymbol{w}_o = r_o \int_{t=0}^{T} \frac{\nabla_{\boldsymbol{w}_o} V_o(t)}{\|\nabla_{\boldsymbol{w}_o} V_o(t)\|} e(t) dt. \tag{25}$$

Now, we proceed to determine the gradients $\nabla_{\boldsymbol{w}_{h,i}} V_o(t)$ and $\nabla_{\boldsymbol{w}_o} V_o(t)$.

5.1.3. Gradient approximation

We use an approximation of $V_o(t)$ which is affine in $\boldsymbol{w}_o$ and given as

$$\widehat{\boldsymbol{d}}_o(t) = \boldsymbol{c}_o(t) * \widehat{h}(t) \tag{26}$$

$$\Rightarrow V_o(t) \approx \widehat{V}_o(t) = E_L + \boldsymbol{w}_o^T \widehat{\boldsymbol{d}}_o(t), \tag{27}$$

where $\widehat{h}(t) = (1/C_m) \exp\left(-t/\tau_L'\right) u(t)$ with $\tau_L' \leq \tau_L$. Here, τ_L' is a hyper-parameter of learning rule that needs to be determined empirically. Similarly $V_{h,i}(t)$ can be approximated as

$$\widehat{\boldsymbol{d}}_h(t) = \boldsymbol{c}_h(t) * \widehat{h}(t) \tag{28}$$

$$\Rightarrow V_{h,i}(t) \approx \widehat{V}_{h,i}(t) = E_L + \boldsymbol{w}_{h,i}^T \widehat{\boldsymbol{d}}_h(t). \tag{29}$$

Note that $\widehat{V}_{h,i}(t)$ and $\widehat{V}_o(t)$ are linear in weight vectors $\boldsymbol{w}_{h,\,i}$ and $\boldsymbol{w}_o$ respectively of corresponding input synapses. From Eq. (27), we approximate $\nabla_{\boldsymbol{w}_o} V_o(t)$ as

$$\begin{aligned} \nabla_{\boldsymbol{w}_o} V_o(t) &\approx \nabla_{\boldsymbol{w}_o} \widehat{V}_o(t) \\ &= \widehat{\boldsymbol{d}}_o(t). \end{aligned} \tag{30}$$

Similarly $\nabla_{\boldsymbol{w}_{h,i}} V_o(t)$ can be approximated as

$$\begin{aligned} \nabla_{\boldsymbol{w}_{h,i}} V_o(t) &\approx \nabla_{\boldsymbol{w}_{h,i}} \widehat{V}_o(t) \\ &= w_{o,i} \left(\nabla_{\boldsymbol{w}_{h,i}} \widehat{d}_{o,i}(t)\right), \end{aligned} \tag{31}$$

since only $\widehat{d}_{o,i}(t)$ depends on $\boldsymbol{w}_{h,\,i}$. Thus, from Eq. (26), we get

$$\nabla_{\boldsymbol{w}_{h,i}} V_o(t) \approx w_{o,i} \left(\nabla_{\boldsymbol{w}_{h,i}} c_{o,i}(t) * \widehat{h}(t)\right). \tag{32}$$

We know that $c_{o,i}(t) = \sum_s \alpha\left(t - t_{h,i}^s\right)$, where $t_{h,i}^s$ denotes the sth spiking instant of ith neuron in the hidden layer. Using the chain rule of differentiation, we get

$$\nabla_{\boldsymbol{w}_{h,i}} c_{o,i}(t) \approx \left(\sum_s \delta\left(t - t_{h,i}^s\right) \frac{\widehat{\boldsymbol{d}}_h\left(t_{h,i}^s\right)}{V_{h,i}'\left(t_{h,i}^s\right)} \right) * \alpha'(t). \tag{33}$$

Refer to the Appendix A for a detailed derivation of Eq. (33). Using Eqs. (32) and (33), we obtain an approximation to $\nabla_{\boldsymbol{w}_{h,i}} V_o(t)$ as

$$\nabla_{\boldsymbol{w}_{h,i}} V_o(t) \approx w_{o,i} \cdot \left(\sum_s \delta\left(t - t_{h,i}^s\right) \frac{\widehat{\boldsymbol{d}}_h\left(t_{h,i}^s\right)}{V_{h,i}'\left(t_{h,i}^s\right)} \right) * \left(\alpha'(t) * \widehat{h}(t)\right). \tag{34}$$

Note that the key enabling idea in the derivation of the above learning rule is the use of the inverse of the time rate of change of the neuronal membrane potential to capture the dependency of its spike time on its membrane potential, as shown in the Appendix A in detail.

5.2. Spatio-temporal error backpropagation

Incorporating the approximation from Eq. (30) into Eq. (25), we get the weight adaptation rule for $\boldsymbol{w}_o$ as

$$\Delta \boldsymbol{w}_o = r_o \int_0^T \frac{\widehat{\boldsymbol{d}}_o(t)}{\|\widehat{\boldsymbol{d}}_o(t)\|} e(t) dt. \tag{35}$$

Similarly incorporating the approximation made in Eq. (34) into Eq. (24), we obtain the weight adaptation rule for $\boldsymbol{w}_{h,\,i}$ as

$$\begin{aligned} \Delta \boldsymbol{w}_{h,i} &= r_h \cdot w_{o,i} \\ &\cdot \int_{t=0}^{T} \left(\left(\sum_s \delta\left(t - t_{h,i}^s\right) \frac{\widehat{\boldsymbol{d}}_h\left(t_{h,i}^s\right)}{V_{h,i}'\left(t_{h,i}^s\right)} \right) * \alpha'(t) * \widehat{h}(t) \right) e(t) dt. \end{aligned} \tag{36}$$

Thus the adaptation rule for the weight matrix W_h is given as

$$\Delta W_h = r_h \cdot \int_{t=0}^{T} \Big(\big(U_h(t) \boldsymbol{w}_o \widehat{\boldsymbol{d}}_h^T(t) \big) * \alpha'(t) * \widehat{h}(t) \Big) e(t) dt, \tag{37}$$

where $U_h(t)$ is a $m \times m$ diagonal matrix with ith diagonal entry given as

$$u_{h,ii}(t) = \frac{\sum_s \delta\big(t - t_{h,i}^s\big)}{V'_{h,i}(t)}. \tag{38}$$

Note that Eq. (37) requires $\mathcal{O}(mn)$ convolutions to compute ΔW_h. Using the identity (derived in Appendix B)

$$\int_t (x(t) * y(t)) z(t) dt = \int_t (z(t) * y(-t)) x(t) dt, \tag{39}$$

Eq. (37) can be equivalently written in following form, which lends itself to a more efficient implementation involving only $\mathcal{O}(1)$ convolutions.

$$\Delta W_h = r_h \cdot \int_{t=0}^{T} \big(e(t) * \alpha'(-t) * \widehat{h}(-t) \big) U_h(t) \boldsymbol{w}_o \widehat{\boldsymbol{d}}_h^T(t) dt. \tag{40}$$

Rearranging the terms as follows brings forth the inherent process of *spatio-temporal backpropagation of error* happening during NormAD based training.

$$\Delta W_h = r_h \cdot \int_{t=0}^{T} U_h(t) \Big((\boldsymbol{w}_o e(t)) * \alpha'(-t) * \widehat{h}(-t) \Big) \widehat{\boldsymbol{d}}_h^T(t) dt. \tag{41}$$

Here spatial backpropagation is done through the weight vector $\boldsymbol{w}_o$ as

$$\boldsymbol{e}_h^{spat}(t) = \boldsymbol{w}_o e(t) \tag{42}$$

and then temporal backpropagation by convolution with time reversed kernels $\alpha'(t)$ and $h(t)$ and sampling with $U_h(t)$ as

$$\boldsymbol{e}_h^{temp}(t) = U_h(t) \Big(\boldsymbol{e}_h^{spat}(t) * \alpha'(-t) * \widehat{h}(-t) \Big). \tag{43}$$

It will be more evident when we generalize it to SNNs with arbitrarily many hidden layers.

From Eq. (36), note that the weight update for synapses of a neuron in hidden layer depends on its own spiking activity thus suggesting the spike-induced nature of weight update. However, in case all the spikes of the hidden layer vanish in a particular training iteration, there will be no spiking activity in the output layer and as per Eq. (36) the weight update $\Delta \boldsymbol{w}_{h,i} = \boldsymbol{0}$ for all subsequent iterations. To avoid this, regularization techniques such as constraining the average spike rate of neurons in the hidden layer to a certain range can be used, though it has not been used in the present work.

5.2.1. Generalization to deep SNNs

For the case of feedforward SNNs with two or more hidden layers, the weight update rule for output layer remains the same as in Eq. (35). Here, we provide the general weight update rule for any particular hidden layer of an arbitrary fully connected feedforward SNN $N_0 \to N_1 \to N_2 \cdots N_{L-1} \to 1$ with L layers as shown in Fig. 5. This can be obtained by the straight-forward extension of the derivation for the case with single hidden layer discussed above. For this discussion, the subscript h or o indicating the layer of the corresponding neuron in the previous discussion is replaced by the layer index to accommodate arbitrary number of layers. The iterative weight update rule for synapses connecting neurons in layer $l-1$ to neurons in layer l viz., W_l $(0 < l < L)$ is given as follows:

$$\Delta W_l = r_h \int_{t=0}^{T} \boldsymbol{e}_l^{temp}(t) \widehat{\boldsymbol{d}}_l^T(t) dt \qquad \text{for } 0 < l < L, \tag{44}$$

where

$$\boldsymbol{e}_l^{temp}(t) = \begin{cases} U_l(t) \Big(\boldsymbol{e}_l^{spat}(t) * \alpha'(-t) * \widehat{h}(-t) \Big) & 1 < l < L \\ e(t) & l = L, \end{cases} \tag{45}$$

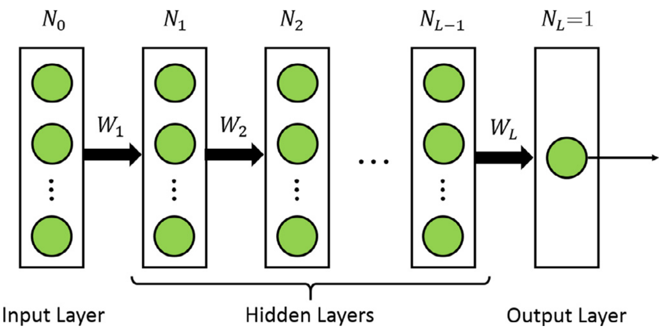


Fig. 5. Fully connected feedforward SNN with L layers ($N_0 \to N_1 \to N_2 \cdots N_{L-1} \to 1$).

Table 1
XOR problem set-up from [13], which uses arrival time of spike to encode logical 0 and 1.

Input spike time (ms)			Output
Bias	Input 1	Input 2	spike time (ms)
0	-	-	16
0	-	6	10
0	6	-	10
0	6	6	16

performs *temporal backpropagation* following the *spatial backpropagation* as

$$\boldsymbol{e}_l^{spat}(t) = W_{l+1}^T \boldsymbol{e}_{l+1}^{temp}(t) \qquad \text{for } 1 < l < L. \tag{46}$$

Here $U_l(t)$ is an $N_l \times N_l$ diagonal matrix with nth diagonal entry given as

$$u_{l,nn}(t) = \frac{\sum_s \delta\big(t - t_{l,n}^s\big)}{V'_{l,n}(t)}, \tag{47}$$

where $V_{l,n}(t)$ is the membrane potential of nth neuron in layer l and $t_{l,n}^s$ is the time of its sth spike. From Eq. (45), note that temporal backpropagation through layer l requires $\mathcal{O}(N_l)$ convolutions.

6. Numerical validation

In this section we validate the applicability of NormAD based spatio-temporal error backpropagation to the training of multilayer SNNs. The algorithm comprises of Eqs. (44)–(47).

6.1. XOR problem

XOR problem is a prominent example of non-linear classification problems which can not be solved using the single layer neural network architecture and hence compulsorily require a multilayer network. Here, we present how the proposed NormAD based training was employed to solve a spike domain formulation of the XOR problem for a multi-layer SNN. The XOR problem is similar to the one used in [13] and is represented by Table 1. There are 3 input neurons and 4 different input spike patterns given in the 4 rows of the table, where temporal encoding is used to represent logical 0 and 1. The numbers in the table represent the arrival time of spikes at the corresponding neurons. The bias input neuron always spikes at $t = 0$ ms. The other two inputs can have two types of spiking activity viz., presence or absence of a spike at $t = 6$ ms, representing logical 1 and 0 respectively. The desired output is coded such that an early spike (at $t = 10$ ms) represents a logical 1 and a late spike (at $t = 16$ ms) represents a logical 0.

In the network reported in [13], the three input neurons had 16 synapses with axonal delays of $0, 1, 2, \ldots, 15$ ms respectively. Instead of having multiple synapses, we use a set of 18 different input neurons for each of the three inputs such that when the first

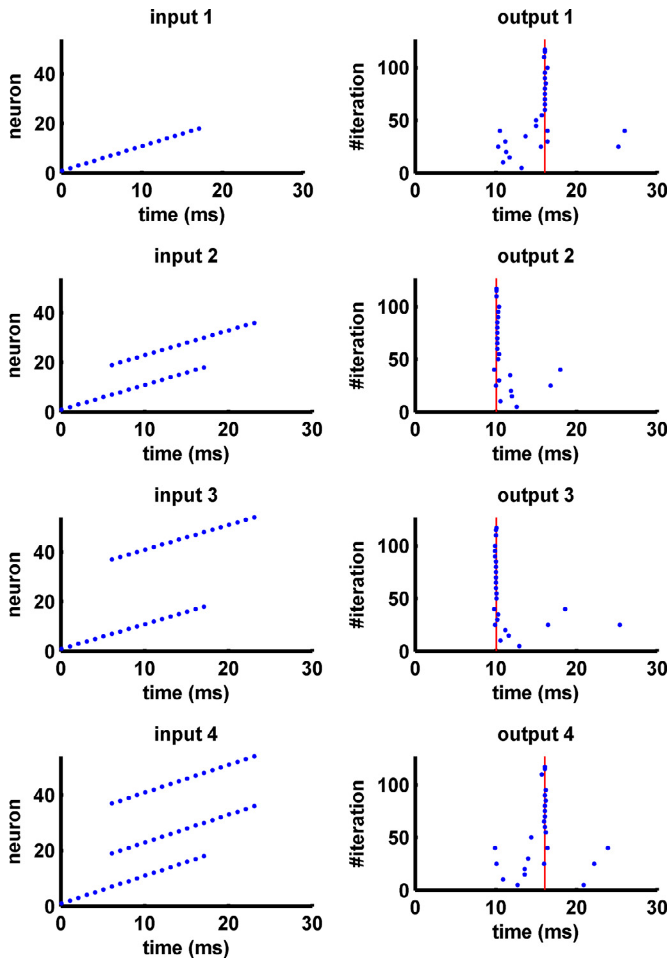
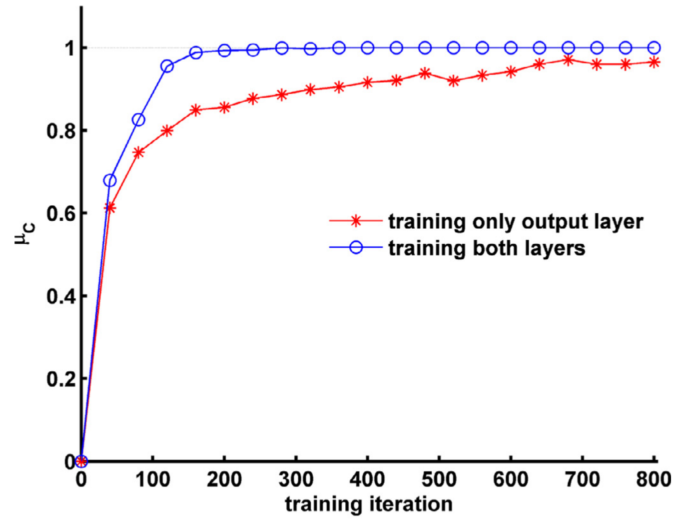
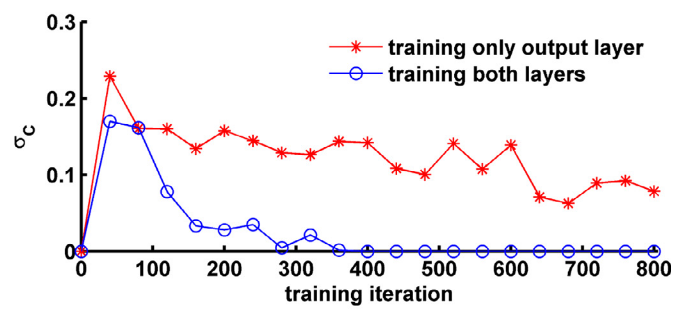


Fig. 6. XOR problem: Input spike raster (left) and corresponding output spike raster (right - blue dots) obtained during NormAD based training of a $54 \rightarrow 54 \rightarrow 1$ SNN with vertical red lines marking the position of desired spikes. The output spike raster is plotted for one in every 5 training iterations for clarity.



(a) Mean spike correlation



(b) Standard deviation of spike correlation

Fig. 7. Plots of (a) mean and (b) standard deviation of spike correlation metric over 100 different initializations of $54 \rightarrow 54 \rightarrow 1$ SNN, trained for the XOR problem with non-plastic hidden layer (red asterisk) and plastic hidden layer (blue circles).

neuron of the set spikes, second one spikes after 1 ms, third one after another 1 ms and so on. Thus, there are 54 input neurons comprising of three sets with 18 neurons in each set. So, a $54 \rightarrow 54 \rightarrow 1$ feedforward SNN is trained to perform the XOR operation in our implementation. Input spike rasters corresponding to the 4 input patterns are shown in Fig. 6 (left).

Weights of synapses from the input layer to the hidden layer were initialized randomly using Gaussian distribution, with 80% of the synapses having positive mean weight (excitatory) and rest 20% of the synapses having negative mean weight (inhibitory). The network was trained using NormAD based spatio-temporal error backpropagation. Fig. 6 plots the output spike raster (on right) corresponding to each of the four input patterns (on left), for an exemplary initialization of the weights from the input to the hidden layer. As can be seen, convergence was achieved in less than 120 training iterations in this experiment.

The necessity of a multi-layer SNN for solving an XOR problem is well known, but to demonstrate the effectiveness of NormAD based training to hidden layers as well, we conducted two experiments. For 100 independent random initializations of the synaptic weights to the hidden layer, the SNN was trained with (i) non-plastic hidden layer, and (ii) plastic hidden layer. The output layer was trained using Eq. (35) in both the experiments. Fig. 7a and 7 b show the mean and standard deviation respectively of spike correlation against training iteration number for the two experiments. For the case with non-plastic hidden layer, the mean correlation reached close to 1, but the non-zero standard deviation represents a sizable number of experiments which did not converge even after 800 training iterations. When the synapses in hidden layer were also trained, convergence was obtained for all the 100 initializations within 400 training iterations. The convergence criteria used in these experiments was to reach the perfect spike correlation metric of 1.0.

6.2. Training SNNs with 2 hidden layers

Next, to demonstrate spatio-temporal error backpropagation through multiple hidden layers, we applied the algorithm to train a $100 \rightarrow 50 \rightarrow 25 \rightarrow 1$ feedforward SNN for general spike based training problems. The weights of synapses feeding the output layer were initialized to 0, while synapses feeding the hidden layers were initialized using a uniform random distribution and with 80% of them excitatory and the rest 20% inhibitory. Each training problem comprised of $n = 100$ input spike trains and one desired output spike train, all generated to have Poisson distributed spikes with arrival rate $20\,\mathrm{s}^{-1}$ for inputs and $10\,\mathrm{s}^{-1}$ for the output, over an epoch duration $T = 500\,\mathrm{ms}$. Fig. 8 shows the progress of training for an exemplary training problem by plotting the output spike rasters for various training iterations overlaid on plots of vertical red lines denoting the positions of desired spikes.

To assess the gain of training hidden layers using NormAD based spatio-temporal error backpropagation, we ran a set of 3 experiments. For 100 different training problems for the same SNN architecture as described above, we studied the effect of (i) train-

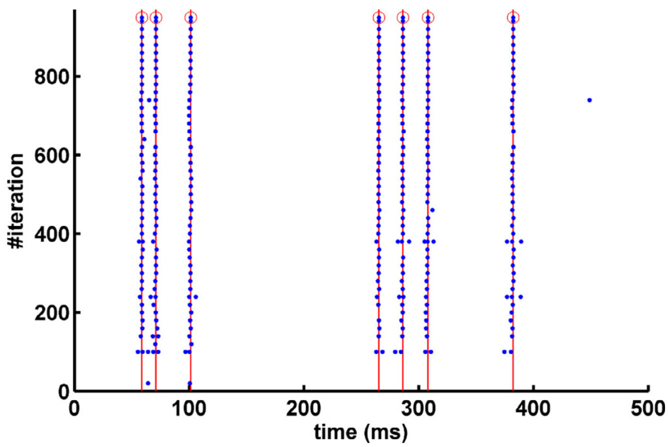


Fig. 8. Illustrating NormAD based training of an exemplary problem for 3-layer 100 → 50 → 25 → 1 SNN. The output spike rasters (blue dots) obtained during one in every 20 training iterations (for clarity) is shown, overlaid on plots of vertical red lines marking positions of the desired spikes.

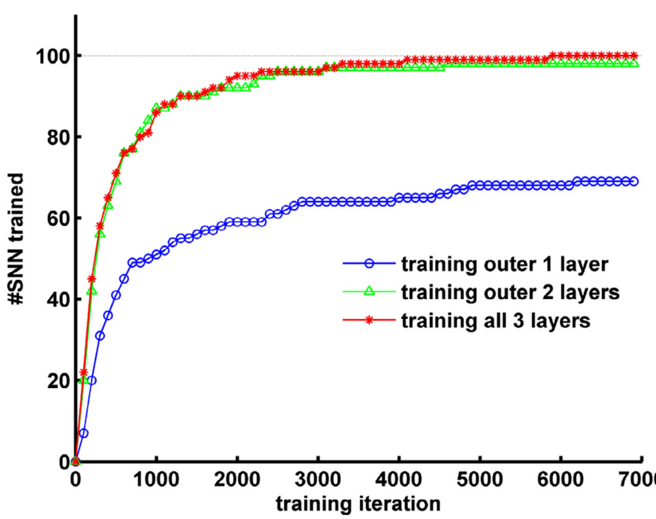
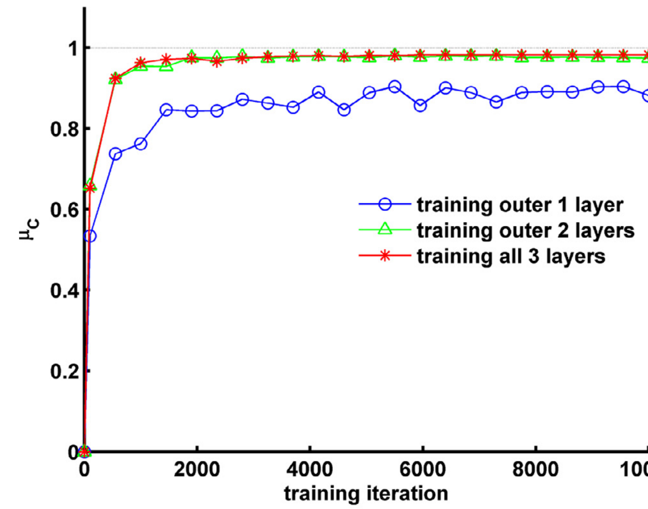
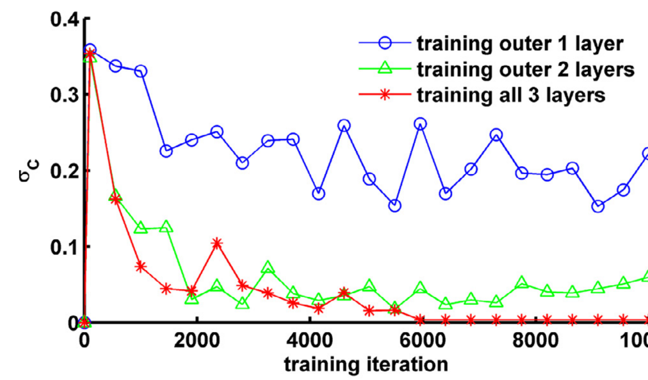


Fig. 9. Plots showing cumulative number of training problems for which convergence was achieved out of total 100 different training problems for 3-layer 100 → 50 → 25 → 1 SNNs.



(a) Mean spike correlation



(b) Standard deviation of spike correlation

Fig. 10. Plots of (a) mean and (b) standard deviation of spike correlation metric while partially or completely training 3-layer 100 → 50 → 25 → 1 SNNs for 100 different training problems.

ing only the output layer weights, (ii) training only the outer 2 layers and (iii) training all the 3 layers. Fig. 9 plots the cumulative number of SNNs trained against number of training itertions for the 3 cases, where the criteria for completion of training is reaching the correlation metric of 0.98 or above. Fig. 10a and 10b show plots of mean and standard deviation respectively of spike correlation against training iteration number for the 3 experiments. As can be seen, in the third experiment when all 3 layers were trained, all 100 training problems converged within 6000 training iterations. In contrast, the first 2 experiments have non-zero standard deviation even until 10000 training iterations indicating non-convergence for some of the cases. In the first experiment, where only synapses feeding the output layer were trained, convergence was achieved only for 71 out of 100 training problems after 10, 000 iterations. However, when the synapses feeding the top two layers or all three layers were trained, the number of cases reaching convergence rose to 98 and 100, respectively, thus proving the effectiveness of the proposed NormAD based training method for multi-layer SNNs.

7. Conclusion

We developed NormAD based spatio-temporal error backpropagation to train multi-layer feedforward spiking neural networks. It is the spike domain analogue of error backpropagation algorithm used in second generation neural networks. The derivation was accomplished by first formulating the corresponding training problem as a non-convex optimization problem and then employing Normalized Approximate Descent based optimization to obtain the weight adaptation rule for the SNN. The learning rule was validated by applying it to train 2 and 3-layer feedforward SNNs for a spike domain formulation of the XOR problem and general spike domain training problems respectively.

The main contribution of this work is hence the development of a learning rule for spiking neural networks with arbitrary number of hidden layers. One of the major hurdles in achieving this has been the problem of backpropagating errors through non-linear leaky integrate-and-fire dynamics of a spiking neuron. We have tackled this by introducing temporal error backpropagation and quantifying the dependence of the time of a spike on the corresponding membrane potential by the inverse temporal rate of change of the membrane potential. This together with the spatial backpropagation of errors constitutes NormAD based training of multi-layer SNNs.

The problem of local convergence while training second generation deep neural networks is tackled by unsupervised *pretraining* prior to the application of error backpropagation [11,35]. Development of such unsupervised pretraining techniques for deep SNNs is a topic of future research, as NormAD could be applied in principle to develop SNN based autoencoders.

Acknowledgment

This research was supported in part by the U.S. National Science Foundation through the grant 1710009. The authors acknowledge the invaluable insights gained during their stay at Indian Institute of Technology, Bombay where the initial part of this work was conceived and conducted as a part of a master's thesis project. We also acknowledge the reviewer comments which helped us expand the scope of this work and bring it to its present form.

Appendix A. Gradient approximation.

Derivation of Eq. (33) is presented below:

$$\begin{aligned}\nabla_{\boldsymbol{w}_{h,i}} c_{o,i}(t) &= \sum_s \frac{\partial \alpha\left(t - t^s_{h,i}\right)}{\partial t^s_{h,i}} \cdot \nabla_{\boldsymbol{w}_{h,i}} t^s_{h,i} \qquad \text{(from Eq. (14))} \\ &= \sum_s -\alpha'\left(t - t^s_{h,i}\right) \cdot \nabla_{\boldsymbol{w}_{h,i}} t^s_{h,i} \end{aligned} \tag{A.1}$$

To compute $\nabla_{\boldsymbol{w}_{h,i}} t^s_{h,i}$, let us assume that a small change $\delta w_{h,\,ij}$ in $w_{h,\,ij}$ led to changes in $V_{h,i}(t)$ and $t^s_{h,i}$ by $\delta V_{h,i}(t)$ and $\delta t^s_{h,i}$, respectively i.e.,

$$V_{h,i}(t^s_{h,i} + \delta t^s_{h,i}) + \delta V_{h,i}(t^s_{h,i} + \delta t^s_{h,i}) = V_T. \tag{A.2}$$

From Eq. (29), $\delta V_{h,i}(t)$ can be approximated as

$$\delta V_{h,i}(t) \approx \delta w_{h,ij} \cdot \widehat{d}_{h,j}(t), \tag{A.3}$$

hence from Eq. (A.2) above

$$V_{h,i}(t^s_{h,i}) + \delta t^s_{h,i} V'_{h,i}(t^s_{h,i}) + \delta w_{h,ij} \cdot \widehat{d}_{h,j}(t^s_{h,i} + \delta t^s_{h,i}) \approx V_T$$

$$\begin{aligned} &\Rightarrow \frac{\delta t^s_{h,i}}{\delta w_{h,ij}} \approx \frac{-\widehat{d}_{h,j}(t^s_{h,i} + \delta t^s_{h,i})}{V'_{h,i}(t^s_{h,i})} \qquad (\text{since } V_{h,i}(t^s_{h,i}) = V_T) \\ &\Rightarrow \frac{\partial t^s_{h,i}}{\partial w_{h,ij}} \approx \frac{-\widehat{d}_{h,j}(t^s_{h,i})}{V'_{h,i}(t^s_{h,i})} \\ &\Rightarrow \nabla_{\boldsymbol{w}_{h,i}} t^s_{h,i} \approx \frac{-\widehat{\boldsymbol{d}}_h\left(t^s_{h,i}\right)}{V'_{h,i}\left(t^s_{h,i}\right)}. \end{aligned} \tag{A.4}$$

Thus using Eq. (A.4) in Eq. (A.1) we get

$$\begin{aligned}\nabla_{\boldsymbol{w}_{h,i}} c_{o,i}(t) &\approx \sum_s \alpha'\left(t - t^s_{h,i}\right) \frac{\widehat{\boldsymbol{d}}_h\left(t^s_{h,i}\right)}{V'_{h,i}\left(t^s_{h,i}\right)} \\ &\approx \left(\sum_s \delta\left(t - t^s_{h,i}\right) \frac{\widehat{\boldsymbol{d}}_h\left(t^s_{h,i}\right)}{V'_{h,i}\left(t^s_{h,i}\right)}\right) * \alpha'(t). \end{aligned} \tag{A.5}$$

Note that approximation in Eq. (A.4) is an important step towards obtaining weight adaptation rule for hidden layers, as it now allows us to approximately model the dependence of the spiking instant of a neuron on its inputs using the inverse of the time derivative of its membrane potential.

Appendix B

Lemma 1. *Given 3 functions $x(t)$, $y(t)$ and $z(t)$*

$$\int_t (x(t) * y(t)) z(t) dt = \int_t (z(t) * y(-t)) x(t) dt.$$

Proof. By definition of linear convolution

$$\int_t (x(t) * y(t)) z(t) dt = \int_t \left(\int_u x(u) y(t-u) du\right) z(t) dt.$$

Changing the order of integration, we get

$$\begin{aligned}\int_t (x(t) * y(t)) z(t) dt &= \int_u x(u) \left(\int_t y(t-u) z(t) dt\right) du \\ &= \int_u x(u) (y(-u) * z(u)) du \\ &= \int_t (z(t) * y(-t)) x(t) dt. \qquad \square\end{aligned}$$

References

[1] W. Maass, Networks of spiking neurons: the third generation of neural network models, Neural Netw. 10 (9) (1997) 1659–1671.

[2] S.M. Bohte, The evidence for neural information processing with precise spike-times: a survey 3 (2) (2004) 195–206, doi:10.1023/B:NACO.0000027755.02868.60.

[3] P. Crotty, W.B. Levy, Energy-efficient interspike interval codes, Neurocomputing 65 (2005) 371–378. Computational Neuroscience: Trends in Research 2005

[4] W. Bialek, D. Warland, R. de Ruyter van Steveninck, Spikes: Exploring the Neural Code, MIT Press, Cambridge, Massachusetts, 1996.

[5] W. Gerstner, A.K. Kreiter, H. Markram, A.V.M. Herz, Neural codes: firing rates and beyond, Proc. Natl. Acad. Sci. 94 (24) (1997) 12740–12741.

[6] S.A. Prescott, T.J. Sejnowski, Spike-rate coding and spike-time coding are affected oppositely by different adaptation mechanisms, J. Neurosci. 28 (50) (2008) 13649–13661, doi:10.1523/JNEUROSCI.1792-08.2008.

[7] S. Thorpe, A. Delorme, R. Van Rullen, Spike-based strategies for rapid processing, Neural Netw. 14 (6) (2001) 715–725.

[8] 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, et al., A million spiking-neuron integrated circuit with a scalable communication network and interface, Science 345 (6197) (2014) 668–673.

[9] J. Gehlhaar, Neuromorphic processing: a new frontier in scaling computer architecture, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2014, pp. 317–318, doi:10.1145/2541940.2564710.

[10] M. Davies, et al., Loihi: a neuromorphic manycore processor with on-chip learning, in IEEE Micro 38 (1) (2018) 82–99, doi:10.1109/MM.2018.112130359.

[11] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.

[12] N. Anwani, B. Rajendran, Normad-normalized approximate descent based supervised learning rule for spiking neurons, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), IEEE, 2015, pp. 1–8.

[13] S.M. Bohte, J.N. Kok, H. La Poutre, Error-backpropagation in temporally encoded networks of spiking neurons, Neurocomputing 48 (1) (2002) 17–37.

[14] O. Booij, H. tat Nguyen, A gradient descent rule for spiking neurons emitting multiple spikes, Inf. Process. Lett. 95 (6) (2005) 552–558.

[15] F. Ponulak, A. Kasinski, Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting, Neural Comput. 22 (2) (2010) 467–510.

[16] A. Taherkhani, A. Belatreche, Y. Li, 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, doi:10.1109/TNNLS.2015.2404938.

[17] H. Paugam-Moisy, R. Martinez, S. Bengio, A supervised learning approach based on STDP and polychronization in spiking neuron networks., in: Proceedings of the ESANN, 2007, pp. 427–432.

[18] I. Sporea, A. Graning, Supervised learning in multilayer spiking neural networks, Neural Comput. 25 (2) (2013) 473–509, doi:10.1162/NECO_a_00396.

[19] Y. Xu, X. Zeng, S. Zhong, A new supervised learning algorithm for spiking neurons, Neural Comput. 25 (6) (2013) 1472–1511.

[20] Q. Yu, H. Tang, K.C. Tan, H. Li, Precise-spike-driven synaptic plasticity: learning Hetero-association of spatiotemporal spike patterns, PLoS ONE 8 (11) (2013) e78318.

[21] A. Mohemmed, S. Schliebs, S. Matsuda, N. Kasabov, Span: spike pattern association neuron for learning spatio-temporal spike patterns, Int. J. Neural Syst. 22 (04) (2012).

[22] J.J. Wade, L.J. McDaid, J.A. Santos, H.M. Sayers, Swat: a spiking neural network training algorithm for classification problems, IEEE Trans. Neural Netw. 21 (11) (2010) 1817–1830, doi:10.1109/TNN.2010.2074212.

[23] X. Xie, H. Qu, G. Liu, M. Zhang, J. Kurths, An efficient supervised training algorithm for multilayer spiking neural networks, PLoS ONE 11 (4) (2016) 1–29, doi:10.1371/journal.pone.0150329.

[24] X. Lin, X. Wang, Z. Hao, Supervised learning in multilayer spiking neural networks with inner products of spike trains, Neurocomputing (2016).

[25] S. Schliebs, N. Kasabov, Evolving spiking neural network—a survey, Evolv. Syst. 4 (2) (2013) 87–98, doi:10.1007/s12530-013-9074-9.

[26] S. SOLTIC, N. KASABOV, Knowledge extraction from evolving spiking neural networks with rank order population coding, Int. J. Neural Syst. 20 (06) (2010) 437–445. PMID: 21117268, doi: 10.1142/S012906571000253X.

[27] R.-M. Memmesheimer, R. Rubin, B.P. lveczky, H. Sompolinsky, Learning precisely timed spikes, Neuron 82 (4) (2014) 925–938, doi:10.1016/j.neuron.2014.03.026.

[28] J.H. Lee, T. Delbruck, M. Pfeiffer, Training deep spiking neural networks using backpropagation, Front. Neurosci. 10 (2016) 508, doi:10.3389/fnins.2016.00508.

[29] Y. Xu, X. Zeng, L. Han, J. Yang, A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks, Neural Netw. 43 (2013) 99–113.

[30] R.V. Florian, The Chronotron: a neuron that learns to fire temporally precise spike patterns, PLoS ONE 7 (8) (2012) e40233.

[31] A.L. Hodgkin, A.F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiol. (Lond.) 117 (4) (1952) 500.

[32] R.B. Stein, Some models of neuronal variability, Biophys. J. 7 (1) (1967) 37.

[33] M.C.W. van Rossum, A novel spike distance, Neural Comput. 13 (4) (2001) 751–763.

[34] F. Ponulak, A. Kasiński, Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting, Neural Comput. 22 (2) (2010) 467–510, doi:10.1162/neco.2009.11-08-901.

[35] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11 (2010) 625–660 http://dl.acm.org/citation.cfm?id=1756006.1756025.

Navin Anwani received an M.Tech. degree from Indian Institute of Technology Bombay in 2015. He is currently a Systems Engineer with Qualcomm Wireless R&D, Bangalore, India. His research interests include neuromorphic learning algorithms and statistical signal processing applications.

Bipin Rajendran received a B. Tech degree from I.I.T. Kharagpur in 2000, and M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 2003 and 2006, respectively. He was a Master Inventor and Research Staff Member at IBM T. J. Watson Research Center in New York during 2006-'12 and a faculty member in the Electrical Engineering Department at I.I.T. Bombay during 2012-'15. His research focuses on building algorithms, devices, and systems for brain-inspired computing. He has authored over 75 papers in peer-reviewed journals and conferences, and has been issued 59 U.S. patents. He is currently an Associate Professor of Electrical & Computer Engineering at New Jersey Institute of Technology.