Online Supervised Learning for Hardware-Based Multilayer Spiking Neural Networks Through the Modulation of Weight-Dependent Spike-Timing-Dependent Plasticity

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

***Abstract*— In this paper, we propose an online learning algorithm for supervised learning in multilayer spiking neural networks (SNNs). It is found that the spike timings of neurons in an SNN can be exploited to estimate the gradients that are associated with each synapse. With the proposed method of estimating gradients, learning similar to the stochastic gradient descent process employed in a conventional artificial neural network (ANN) can be achieved. In addition to the conventional layer-by-layer backpropagation, a one-pass direct backpropagation is possible using the proposed learning algorithm. Two neural networks, with one and two hidden layers, are employed as examples to demonstrate the effectiveness of the proposed learning algorithms. Several techniques for more effective learning are discussed, including utilizing a random refractory period to avoid saturation of spikes, employing a quantization noise injection technique and pseudorandom initial conditions to decorrelate spike timings, in addition to leveraging the progressive precision in an SNN to reduce the inference latency and energy. Extensive parametric simulations are conducted to examine the aforementioned techniques. The learning algorithm is developed with the considerations of ease of hardware implementation and relative compatibility with the classic ANN-based learning. Therefore, the proposed algorithm not only enjoys the high energy efficiency and good scalability of an SNN in its specialized hardware but also benefits from the well-developed theory and techniques of conventional ANN-based learning. The Modified National Institute of Standards and Technology database benchmark test is conducted to verify the newly proposed learning algorithm. Classification correct rates of 97.2% and 97.8% are achieved for the one-hidden-layer and two-hidden-layer neural networks, respectively. Moreover, a brief discussion of the hardware implementations is presented for two mainstream architectures.**

***Index Terms*— Hardware neural network, machine learning, Modified National Institute of Standards and Technology database (MNIST) benchmark, neuromorphic computing, spike-timing-dependent plasticity (STDP), spiking neural network (SNN), supervised learning.**

I. Introduction

OVER the past decade, an enormous amount of research effort has been made to build specialized neuromorphic computing hardware for real-life applications, while the development of the conventional von Neumann architecture-based computing approach has slowed down because of the looming end of Moore's law. In recent years, research focus has shifted from traditional rate-based artificial neural networks (ANNs), which were popular choices of hardware implementations in the 1990s, to the third-generation spiking neural networks (SNNs). This trend is attributed to two unique advantages that SNNs have. The event-triggered nature of an SNN can lead to a very power-efficient computation. It has been shown that a customized SNN hardware is two orders of magnitude more energy efficient than the traditional ANN implemented on a field-programmable gate array [1]. In addition, an SNN has better scalability because an address event representation (AER) can conveniently interconnect sub-SNNs in a large network [2]–[4]. For example, TrueNorth from IBM [2] is a hardware SNN that contains 1 million spiking neurons. It consists of 4096 cores and consumes merely 65 mW while running a real-time multiobject detection and classification task. Unfortunately, SNNs implemented on a general-purpose processor are not able to demonstrate superiority compared to ANNs due to the lack of support of event-based computation in the processor. Therefore, to better exploit the aforementioned advantages of an SNN, many specialized hardware systems have been built, such as the TrueNorth from IBM [2], CAVIAR in Europe [5], and neuromorphic chips from HRL Laboratories [6]. In addition to the conventional CMOS technology, emerging nanotechnology also helps to accelerate the development of the next-generation neuromorphic computing hardware. Since it was first demonstrated in 2008 [7], the memristor has emerged as a popular choice for building neuromorphic hardware because it satisfies all of the requirements of being a synapse. Many studies have been conducted with the attempt to incorporate memristors into a neuromorphic system, including building synapses for both SNNs [8]–[11] and ANNs [12] and developing functional neuromorphic systems with memristors [13], [14]

Even though many SNN hardware implementations have been demonstrated over the past few years, many of them do not have the capability to conduct on-chip learning. One popular way to utilize the SNN hardware is to train a conventional ANN counterpart offline using conventional learning algorithms and then convert it into an SNN and download the learned weights into the specialized hardware [1], [15]–[17].

Manuscript received August 20, 2016; revised March 21, 2017, July 18, 2017, and October 3, 2017; accepted October 3, 2017. Date of publication November 1, 2017; date of current version August 20, 2018. This work was supported by the National Science Foundation under ECCS Grant 1227879 and Grant 1710940. *(Corresponding author: Nan Zheng.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2121 USA (e-mail: zhengn@umich.edu; mazum@umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2017.2761335

One drawback of this method is the performance degradation that is associated with the conversion from ANNs to SNNs. In addition, online learning with low power consumption is an expected feature for specialized neuromorphic hardware to be deployed in the future smart Internet of Things devices, yet it is not feasible to conduct online learning using this method.

Over the past few decades, there were many efforts from both the computational intelligence community and the neuroscience community to develop learning algorithms for SNNs. Spike-timing-dependent plasticity (STDP), which was first observed in real biological experiments, was proposed as an empirically successful learning rule that could be used for unsupervised learning [13], [18]–[20]. In a typical STDP protocol, the synaptic weight updates according to the relative order of spikes and the difference between the presynaptic and postsynaptic spike timings. Unsupervised learning is useful in discovering the underlying structure of the data, yet it is not as powerful as supervised learning in many real-life applications, at least at the current stage. There also exist various algorithms for supervised learning in SNNs, such as SpikeProp [21], ReSuMe [22], tempotron learning rule [23]–[25], and precise-spike-driven (PSD) rule [26], [27]. SpikeProp is one of the earliest proposed methods for supervised learning in SNNs, and it is analogous to the backpropagation employed in conventional ANNs. The tempotron rule relies on a more biologically plausible mechanism. Both of these methods are based on a gradient descent method, and they both limit each neuron in the SNN to fire only once, making them well suitable for classification applications where the output is a one-hot code. However, it is not convenient to employ such a learning algorithm for a softmax classifier or a universal function approximator. ReSuMe and the PSD rule both originate from the Widrow–Hoff rule. They have the advantage that they can learn precise spatiotemporal patterns of desired spikes, which makes them attractive for systems in which the information is mainly modulated on the timings of the spikes. Another advantage is that these two learning rules are applicable to many different neuron models. However, how to apply these learning rules to a multilayer neural network is not obvious. This limitation impedes these algorithms from being employed in a deep neural network, which has achieved many astonishing results recently [28]–[30]. In addition, all of these learning rules aim at learning precise firing timings of neurons. It is, however, still debatable what the best way is to encode information using spikes. Therefore, learning exact spike timing might not be the optimum and the most efficient method for many real-life applications.

In this paper, we take a different approach. We do not attempt to learn the precise spike timings of neurons. Rather, learning rules that aim to achieve desired firing densities are developed. Nevertheless, the spike timings of neurons are employed to estimate the gradient components in the learning process. It is found that a term that resembles the quantity used in an STDP protocol can be exploited to estimate the gradients. Furthermore, how to propagate the errors from the output neurons to each synapse is studied; this process leads to a learning process that is similar to a conventional backpropagation. The gradients estimated from spike timings are exploited to conduct gradient descent learning. By developing such a learning rule, we can take advantage of the power efficiency and scalability of a specialized SNN hardware. In addition, the developed learning algorithm stays relatively compatible with the learning in a conventional ANN. Therefore, many theories and techniques developed for the well-established ANN-based learning, such as momentum and mini-batch, can be applied to the proposed learning algorithm. Despite the similarities, many unique features associated with the proposed learning algorithm provide new opportunities. For example, in contrast to the layer-by-layer backpropagation in a conventional ANN, a direct backpropagation in SNN is possible by properly utilizing the spike timings of neurons, thus reducing the computational effort and improving the backpropagation speed. Another example is that the trained SNN can infer with progressive precision, thus accelerating the inference process and reducing the energy consumption of the SNN hardware.

In the following, we introduce the method of estimating gradients and conducting supervised learning using a weight-dependent STDP term in Section II. Two examples of neural networks are presented in Section III, which are used to examine various aspects of the proposed learning algorithm. In Section IV, we discuss how the newly proposed algorithm can fit into two popular hardware architectures in neuromorphic computing. Conclusions are drawn in Section V.

II. STDP as a Measure of the Gradient

We use the following notations throughout this paper.

1) A deterministic discrete-time signal is denoted by a lowercase symbol and is indexed by n, for example, $x[n]$.
2) A discrete-time stochastic process is denoted with an uppercase symbol and is indexed by n, for example, $X[n]$.
3) $\Pr(\cdot)$ and $E[\cdot]$ are used to represent the probability and expectation, respectively.
4) The expectation of a random variable is denoted by μ, and the arithmetic average of signal $x[n]$ over a finite duration is denoted by $\overline{x}$.
5) A column vector is denoted by a bold symbol such as $\mathbf{x}$, and its elements are denoted by x_i, where $i = 1, 2, \cdots$.
6) The differentiation of a function f with respect to a vector is an element-wise operation. For example, $\partial f/\partial \mathbf{x} = [\partial f/\partial x_1, \partial f/\partial x_2, \cdots]^T$.

A generic diagram of a multilayer neural network is shown in Fig. 1. In Fig. 1, a neuron that is located at the lth layer is denoted as x_i^l, where i represents the index of that neuron. There are, in total, N_l such neurons in the lth layer. For a pair of neurons, a presynaptic neuron x_i^l and a postsynaptic neuron x_j^{l+1}, their output spike trains are denoted as

$$x_i^l[n] = \sum_m \delta\left[n - n_{i,m}^l\right] \tag{1}$$

$$x_j^{l+1}[n] = \sum_m \delta\left[n - n_{j,m}^{l+1}\right] \tag{2}$$

where $\delta[n]$ is the unit sample sequence, m is the index for spikes, and $n_{i,m}^l$ and $n_{j,m}^{l+1}$ are spike timings for the mth spikes from neuron x_i^l and neuron x_j^{l+1}, respectively. Since our

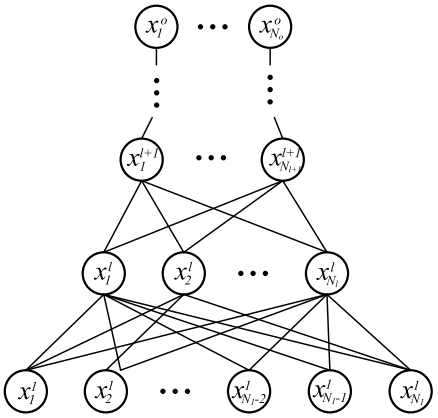

Fig. 1. Illustration of a multilayer neural network. A neuron located at the lth layer is denoted as x_i^l, where i represents the index of that neuron.

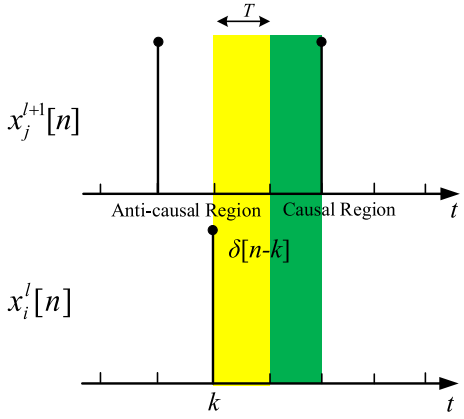

Fig. 2. Illustration of two regions divided by the spike timing of a presynaptic neuron. The causal and anti-causal regions are defined according the causal relationship between presynaptic and postsynaptic spikes.

primary interests are in training hardware-based SNNs, we restrict ourselves to discrete-time systems and the usage of a constant excitatory postsynaptic potential. This way of representing spikes is very popular in the hardware realization of SNNs, considering its ease of implementation and routing.

It was shown in [31] that a presynaptic spike partitions the time axis into two regions: a causal region and an anti-causal region, as shown in Fig. 2. With such a partition, let us define a time sequence and its sample mean as

$$stdp_{ij}^l[n] = x_i^l[n-T]\big(1 - x_i^l[n-T-1]\big) \times \big(x_j^{l+1}[n] - x_j^{l+1}[n-1]\big) \quad (3)$$

$$\overline{stdp_{ij}^l} = \sum_{n=T+1}^{D_L} stdp_{ij}^l[n]/(D_L - T) \quad (4)$$

where T is the time delay associated with the neuron model used, and D_L is the learning duration, which serves as a design parameter. The quantity $stdp_{ij}^l[n]$ measures the causality between the presynaptic and postsynaptic spikes, which is similar to the quantity measured in an STDP protocol. It is demonstrated shortly that this quantity estimates how the postsynaptic neuron changes its behavior when a presynaptic spike occurs.

We consider a class of stochastic neuron model with the dynamics of

$$X_j^{l+1}[n] = H\left(\sum_{i=1}^{N_l} w_{ij}^l X_i^l[n-T] + S_j^{l+1}[n-T]\right). \quad (5)$$

For analysis purposes, we treat spikes as stochastic processes. The spike trains $x_i^l[n]$ and $x_j^{l+1}[n]$ shown in (1) and (2) are particular realizations of $X_i^l[n]$ and $X_j^{l+1}[n]$ in (5). $H(\cdot)$ is the Heaviside function. $S_j^{l+1}[n]$ is a random process that models the internal state of neuron x_j^{l+1}. It could be, for example, the membrane voltage of an integrate-and-fire neuron. The detailed role and implications of $S_j^{l+1}[n]$ are discussed shortly. We are interested in finding out how the mean firing rate of neuron x_j^{l+1} is related to the mean firing rate of its input neuron x_i^l. Before embarking on deriving this relationship, let us assume that the following two conditions hold.

C1) $X_i^l[n]$ and $X_k^l[n]$ are independent for $k = 1, 2, \ldots, N_l$, and $k \neq i$.

C2) $X_i^l[n]$ and $X_j^{l+1}[n]$ are strictly stationary processes, and $C_{X_i^l, X_j^{l+1}}(n, m) = 0$ for $n \neq m - T$, where $C_{X,Y}(\cdot)$ stands for the cross-covariance function.

We first show that under conditions C1) and C2), the mean firing rate of neuron x_j^{l+1}, i.e., μ_j^{l+1}, is a function of the mean firing rates of its input spikes, $\boldsymbol{\mu}^{\mathbf{l}}$, or mathematically, $\mu_j^{l+1} = g(\boldsymbol{\mu}^{\mathbf{l}})$. In addition, we show that $g(\cdot)$ is differentiable with respect to $\boldsymbol{\mu}^{\mathbf{l}}$, and its mth derivative $\partial g^{(m)}/\partial \boldsymbol{\mu}^{\mathbf{l}} = \mathbf{0}$ for $m > 1$. Here, $\mathbf{0}$ denotes a zero vector in which all elements are zero. Note that the time index n is dropped for the sake of cleaner notation because we consider strictly stationary processes.

The mean firing rate of neuron x_j^{l+1} can be expressed as

$$\begin{aligned}\mu_j^{l+1} &= \Pr\big(X_j^{l+1} = 1\big) \\ &= \sum_{b_1^l=0}^{1} \cdots \sum_{b_{N_l}^l=0}^{1} \Bigg[\Pr\big(X_j^{l+1} = 1 \big| X_1^l = b_1^l, \ldots, X_{N_l}^l = b_{N_l}^l\big) \prod_{i=1}^{N_l} \Pr\big(X_i^l = b_i^l\big) \Bigg] \\ &= \sum_{b_1^l=0}^{1} \cdots \sum_{b_{N_l}^l=0}^{1} \Bigg\{ \Pr\big(X_j^{l+1} = 1 \big| X_1^l = b_1^l, \ldots, X_{N_l}^l = b_{N_l}^l\big) \prod_{i=1}^{N_l} \big[\mu_i^l\big(2b_i^l - 1\big) - b_i^l + 1\big] \Bigg\} = g(\boldsymbol{\mu}^{\mathbf{l}}) \end{aligned} \quad (6)$$

where b_1^l is a binary auxiliary variable for the convenience of derivation. In (6), we have used the assumption that X_i^l and X_k^l are independent for $i \neq k$. In addition, the effect of S_j^{l+1} on X_j^{l+1} is implicitly included in the probability $\Pr(X_j^{l+1} = 1 | X_1^l = b_1^l, \ldots, X_{N_l}^l = b_{N_l}^l)$. Qualitatively, when the mean of S_j^{l+1} increases, the neuron tends to fire more frequently. This relationship translates directly to a larger $\Pr(X_j^{l+1} = 1 | X_1^l = b_1^l, \ldots, X_{N_l}^l = b_{N_l}^l)$. It can be shown that the first derivative

of $g(\boldsymbol{\mu}^{\mathbf{l}})$ with respect to μ_i^l is

$$\frac{\partial g}{\partial \mu_i^l} = \sum_{b_1^l=0}^{1} \cdots \sum_{b_{N_l}^l=0}^{1} \left[\Pr\left(X_j^{l+1}=1 \middle| X_1^l=b_1^l,\ldots,X_{N_l}^l=b_{N_l}^l\right) \left(\prod_{\substack{k=1\\k\neq i}}^{N_l}\left[\mu_k^l\left(2b_k^l-1\right)-b_k^l+1\right]\right)\left(2b_i^l-1\right)\right]. \quad (7)$$

Clearly, $\partial g^{(1)}/\partial \mu_i^l$ is not a function of μ_i^l, which implies that $\partial g^{(m)}/\partial \boldsymbol{\mu}^{\mathbf{l}} = \mathbf{0}$ for $m > 1$.

We then show that $\overline{stdp_{i,j}^l}$ defined in (4) is an unbiased estimator of the term $(\partial \mu_j^{l+1}/\partial \mu_i^l)\mu_i^l(1-\mu_i^l)$. According to the law of large numbers, we have

$$\begin{aligned} E\left[\overline{stdp_{ij}^l}\right] &= \Pr\left(X_j^{l+1}=1, X_i^l=1, X_i^{l'}=0\right) \\ &\quad - \Pr\left(X_j^{l+1}=1, X_i^l=0, X_i^{l'}=1\right) \\ &= \left[\Pr\left(X_j^{l+1}=1 | X_i^l=1\right)\right. \\ &\quad \left. - \Pr\left(X_j^{l+1}=1 | X_i^l=0\right)\right]\mu_i^l\left(1-\mu_i^l\right). \quad (8) \end{aligned}$$

In (8), X_i^l denotes the random variable on which X_j^{l+1} depends, whereas $X_i^{l'}$ denotes the random variable of which X_j^{l+1} is independent. From (8), through expanding $g(\boldsymbol{\mu}^{\mathbf{l}})$ in a Taylor series and using the fact that $\partial g^{(m)}/\partial \boldsymbol{\mu}^{\mathbf{l}} = \mathbf{0}$ for $m > 1$, we have

$$\begin{aligned} &\frac{E\left[\overline{stdp_{ij}^l}\right]}{\mu_i^l(1-\mu_i^l)} \\ &= \Pr\left(X_j^{l+1}=1 | X_i^l=1\right) - \Pr\left(X_j^{l+1}=1 \middle| X_i^l=0\right) \\ &= g\left(\boldsymbol{\mu}^{\mathbf{l}} + \left(1-\mu_i^l\right)\mathbf{u}_{\mathbf{i}}^{\mathbf{l}}\right) - g\left(\boldsymbol{\mu}^{\mathbf{l}} - \mu_i^l\mathbf{u}_{\mathbf{i}}^{\mathbf{l}}\right) \\ &= g\left(\boldsymbol{\mu}^{\mathbf{l}}\right) + \frac{\partial g}{\partial \mu_i^l}\left(1-\mu_i^l\right) - g\left(\boldsymbol{\mu}^{\mathbf{l}}\right) - \frac{\partial g}{\partial \mu_i^l}\left(-\mu_i^l\right) = \frac{\partial g}{\partial \mu_i^l} \\ &= \frac{\partial \mu_j^{l+1}}{\partial \mu_i^l} \end{aligned} \quad (9)$$

where $\mathbf{u}_{\mathbf{i}}^{\mathbf{l}}$ is an N_l-D unit vector in which all elements are zero except that the ith element is one. In (9), we have used the assumption that X_i^l and X_k^l are independent for $i \neq k$.

The derivation of (9) is based on assumption C1) and C2). We then examine the validity of (9) qualitatively when C1) and C2) do not hold rigorously. C1) assumes that the spike timings for different input neurons are independent. Even though the density of each neuron might be highly correlated, the spike timing of an individual neuron can be largely independent. The mild assumption that the spike timing of each neuron is somewhat uncorrelated holds for most SNNs. C2) implies that $S_j^{l+1}[n]$ is also strictly stationary and that it should be independent of X_i^l and X_j^{l+1}. Rigorously, $S_j^{l+1}[n]$ depends on all $X_i^l[m]$ in which $m < n-T$ for any neuron model with a memory, such as the popular LIF model. In practice, however, the dependence of $S_j^{l+1}[n]$ on the firing history of a presynaptic neuron is significantly diluted by the firing histories of other independent presynaptic neurons as well as the modulus or noisy reset operations that are associated with the postsynaptic neuron. In addition, the dependence can be weakened to an acceptable level through proper noise injection. This arrangement is illustrated in Section III. A natural extension of (9) is to define the time sequence $stdp_{ij}^l[n]$ in such a way that more samples can be included for each estimation. This approach is illustrated in (10). In the equation, $\mathrm{WIN}_{\mathrm{STDP}}$ is a design parameter that is used to specify the window size of the summation. This method is inspired by the biological STDP, in which an exponential integration window is employed. The purpose of the parameter $\mathrm{WIN}_{\mathrm{STDP}}$ is to include the effects of delayed perturbated outputs, which might be caused by the memory of $S_j^{l+1}[n]$

$$\begin{aligned} &stdp_{ij}^l[n] \\ &= x_i^l[n-T]\left(1-x_i^l[n-T-1]\right) \\ &\quad \times \left(\sum_{m=1}^{\mathrm{WIN}_{\mathrm{STDP}}} x_j^{l+1}[n+m-1] - \sum_{m=1}^{\mathrm{WIN}_{\mathrm{STDP}}} x_j^{l+1}[n-m]\right) \end{aligned} \quad (10)$$

Intuitively, (9) indicates that $\partial \mu_j^{l+1}/\partial \mu_i^l$ can be estimated by observing how the postsynaptic neuron alters its stochastic behavior in response to an input spike that serves as a small perturbation to the network. Even though perturbations from various presynaptic neurons might cause the same postsynaptic neuron to spike, contributions from each presynaptic neuron can be evaluated simultaneously as long as the spike timings of each of the presynaptic neurons are reasonably uncorrelated. For example, at any given time k, as shown in Fig. 2, neuron x_j^{l+1} has equal probability to fire at both regions when the input spike from neuron x_i^l is absent. When the input spike is present, the spike from neuron x_j^{l+1} is more likely to occur at one side of k depending on whether the synapse is excitatory or inhibitory. The contributions of other input neurons appear to be noise, and they can be easily filtered out if they are not correlated. To further decorrelate the spike timings of each neuron in an SNN, a stochastic neuron can be employed. More conveniently, a technique called quantization noise injection, which was introduced in [31], can be utilized. Therefore, (9) can be readily employed in a large network, and individual gradients can be estimated simultaneously. This approach has the same spirit as simultaneous perturbation stochastic approximation [32].

Next, we assume that (11) can approximately describe the input–output relationship in the chosen neuron model, where $f_j^{l+1}(\cdot)$ is a differentiable function that depends on the dynamics of the spiking neuron model that is used. The actual form of $f_j^{l+1}(\cdot)$ is not important in our derivation because it serves as only an intermediate quantity that is substituted eventually. Conceptually, $f_j^{l+1}(\cdot)$ can be obtained, for example, through function fitting

$$\mu_j^{l+1} \approx f_j^{l+1}\left(\sum_i w_{ij}^l \mu_i^l\right). \quad (11)$$

By taking the derivative with respect to μ_i^l in (11), one can obtain

$$f_j^{l+1\prime}\left(\sum_i w_{ij}^l \mu_i^l\right) = \frac{\partial \mu_j^{l+1}}{\partial \mu_i^l} / w_{ij}^l. \tag{12}$$

Then, with (9) and (12), we arrive at

$$\frac{\partial \mu_j^{l+1}}{\partial w_{ij}^l} = \mu_i^l f_j^{l+1\prime}\left(\sum_i w_{ij}^l \mu_i^l\right) = \frac{E\left[\overline{stdp_{ij}^l}\right]}{w_{ij}^l\left(1-\mu_i^l\right)}. \tag{13}$$

Equation (13) resembles the STDP learning rule in the literature. However, in contrast to a conventional STDP rule, a denominator term is included. Mathematically speaking, including the weight gives at least the sign information. If negative weights are allowed, then it is necessary for a term to change the sign in (13), which would otherwise induce a wrong direction for the gradient descent. In addition, the introduction of the weight denominator ensures an upper bound on w_{ij}^l, which serves a similar purpose as the weight-decay technique that is widely used in ANNs [33].

Equations (9) and (13) provide theoretical guidelines to estimate the gradients in an SNN in order to conduct gradient descent learning. In practice, we use $\overline{stdp_{ij}^l}/[\overline{x_i^l}(1-\overline{x_i^l})]$ and $\overline{stdp_{ij}^l}/[w_{ij}^l(1-\overline{x_i^l})]$ to approximate $\partial\mu_j^{l+1}/\partial\mu_i^l$ and $\partial\mu_j^{l+1}/\partial w_{ij}^l$, where $\overline{x_i^l} = \sum_{n=T+1}^{D_L} x_i^l[n]/(D_L - T)$.

To feature gradient descent learning, we need to propagate errors at the output neuron back to each synapse in the neural network. This process can be achieved through a chain rule that is similar to that used in a conventional ANN, as shown the following equation:

$$\frac{\partial \mu_k^o}{\partial w_{ij}^l} = \frac{\partial \mu_k^o}{\partial \mu_j^{l+1}} \cdot \frac{\partial \mu_j^{l+1}}{\partial w_{ij}^l}. \tag{14}$$

In (14), the term $\partial\mu_j^{l+1}/\partial w_{ij}^l$ can be computed according to (13), whereas the term $\partial\mu_k^o/\partial\mu_j^{l+1}$ can be obtained by propagating the gradient layer by layer, similar to the back-propagation used in a conventional ANN. This procedure is shown in the following equation:

$$\frac{\partial \mu_k^o}{\partial \mu_j^{l+1}} = \sum_{i_o=k}^{k} \sum_{i_{o-1}=1}^{N_{o-2}} \cdots \sum_{i_{l+2}=1}^{N_{l+2}} \sum_{i_{l+1}=j}^{j} \prod_{p=l+1}^{o-1} \frac{\partial \mu_{i_{p+1}}^{p+1}}{\partial \mu_{i_p}^p}. \tag{15}$$

Alternatively, a direct propagation method shown in (16) is proposed to estimate the gradient

$$\frac{\partial \mu_k^o}{\partial \mu_j^{l+1}} = \frac{E\left[\overline{cstdp_{jk}^{l+1}}\right]}{\mu_j^{l+1}\left(1-\mu_j^{l+1}\right)}. \tag{16}$$

In (16), $cstdp_{jk}^{l+1}[n]$ is a quantity that is similar to the quantity defined in (3). The only difference is that $\overline{cstdp_{jk}^{l+1}}$ measures the relationship between the $(l+1)$th layer and the output layer. The delay across multiple layers must be considered in this case. Here, (16) is an extension of (9) in the sense that instead of using perturbation to estimate the gradient of the output of a neuron with respect to its input, we estimate the gradient across a network of neurons by observing how the input spike affects the output firing probability.

To verify (13)–(16), we conduct simulations on a two-hidden-layer neural network. The operations of the neural network largely follow the conventions used in TrueNorth [2] because it is the most recently developed, powerful general-purpose neuromorphic hardware. In [2], spikes from neurons can occur only synchronously with a time unit called a tick. This setting guarantees a one-to-one mapping between software and hardware at a tick level, albeit the internal evaluations of the neuron states are asynchronous to save energy. In the remainder of this paper, a tick is used as the minimum temporal resolution as well as the unit for time-related quantities, e.g., $\mathrm{WIN}_{\mathrm{STDP}}$.

The configuration of the neural network is 80-30-100-1, where each number represents the number of neurons at each layer, from input layer to output layer. Here, we employ only one output neuron because the gradients estimated for each output neuron in a neural network are independent of the other output neurons. A modified integrate-and-fire neuron model, shown in (17) and (18), is used. In the model, $x_j^{l+1}[n]$ and $V_j^{l+1}[n]$ are the output and membrane potential of a neuron x_j^{l+1} at tick n, th_j^{l+1} is the threshold to fire, and L_j^{l+1} represents the leakage. It has been shown in [31] that such a neuron model behaves similar to a first-order $\Sigma-\Delta$ modulator, and the quantization noise associated with this model is helpful in achieving less correlated spike timings. In other words, we can randomize the spike timing of each neuron without explicitly using random number generators. In addition, this modified model is one of the models employed in the TrueNorth chips [34]. Therefore, we utilize this model in this paper unless otherwise stated. Nevertheless, our proposed algorithm is not restricted to this modified model. For example, it can also be applied to a conventional leaky integrate-and-fire (LIF) model if noise is properly injected. This approach is demonstrated in Section III. In our simulations, input neurons in the network are injected with excitatory currents at every tick. The injected currents are randomly chosen at the beginning of learning and are fixed throughout the learning. More information on the input encoding is detailed in Section III

$$x_j^{l+1}[n] = \begin{cases} 0, & V_j^{l+1}[n] < \mathrm{th}_j^{l+1} \\ 1, & V_j^{l+1}[n] \geq \mathrm{th}_j^{l+1} \end{cases} \tag{17}$$

$$V_j^{l+1}[n] = \max\Bigg(0, V_j^{l+1}[n-1] + \sum_i w_{ij}^l x_i^l[n-1] - L_j^{l+1} - x_j^{l+1}[n-1]\cdot \mathrm{th}_j^{l+1}\Bigg). \tag{18}$$

Figs. 3–5 show scatter charts that compare gradients estimated from the spike timing and gradients calculated numerically. Ten sets of experiments are conducted, and 100 weights from each layer are randomly chosen for each set of experiments. A thousand data points, in total, are collected in Figs. 3–5 for each layer. Numerical results are obtained with the finite-difference (FD) method. In other words, a small perturbation is applied to the weight, and the gradient is

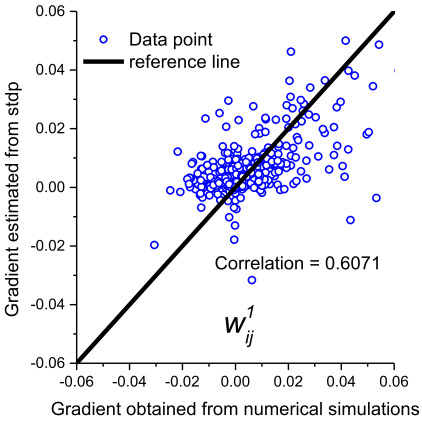

Fig. 3. Comparison of gradients obtained from numerical simulations and gradients obtained from STDP. The gradients are associated with the first-layer synapses w_{ij}^1.

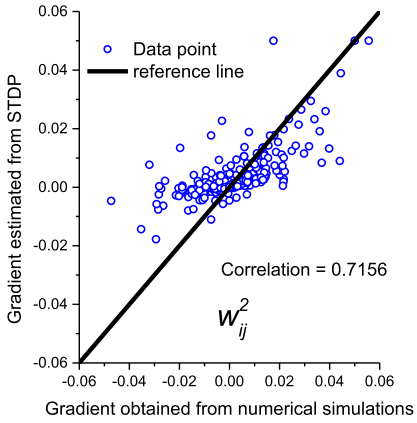

Fig. 4. Comparison of gradients obtained from numerical simulations and gradients obtained from STDP. The gradients are associated with the second-layer synapses w_{ij}^2.

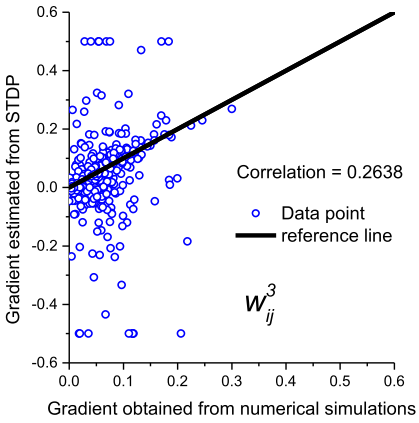

Fig. 5. Comparison of gradients obtained from numerical simulations and gradients obtained from STDP. The gradients are associated with the third-layer synapses w_{ij}^3.

obtained by dividing the change at the output by the amount of perturbation that is applied. Due to the complicated dynamics of the SNN and the limited computational resources, the gradient obtained from the FD method is not the true gradient

TABLE I

INFORMATION FOR THE LIMITING OPERATIONS USED TO OBTAIN DATA IN FIGS. 3–7

Layer index	Maximum/minimum gradient allowed	# of outliers being clamped
1	±0.05	1
2	±0.05	4
3	±0.5	16

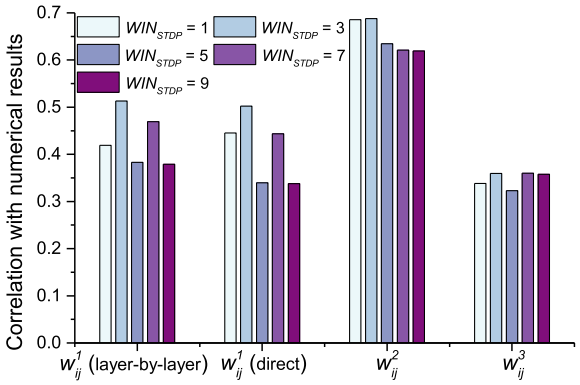

Fig. 6. Correlations between estimated gradients and gradients obtained with the FD numerical method. The results obtained for all three layers of synaptic weights (w_{ij}^1, w_{ij}^2, and w_{ij}^3) are compared. Two different backpropagation methods (layer by layer and direct) are also compared. Different window sizes for evaluating STDP are compared. The estimation accuracy does not show significant dependence on the sizes of the STDP window.

but is instead a noisy version of the true gradient. This gradient asymptotically approaches the true gradient as the number of evaluation ticks increases. Nevertheless, a comparison with such noisy gradients can provide some useful insights into how well the spike timing can be used for estimating gradients. As shown in (13), when the weight is small, the quantization noise in the density of the spike might induce a large estimated gradient variation. Therefore, a limiting operation is needed to limit the maximum and minimum gradients obtained from the spike timing information. Detailed information for this clamping is shown in Table I. Estimated gradients and gradients obtained numerically match well in Figs. 3–5, which demonstrate the effectiveness of the proposed algorithm. It is observed in Fig. 5 that the correlation for w_{ij}^3 is comparatively low. It is found in the simulations that few negative outliers in Fig. 5 are responsible for this low correlation. The reason is that the clamping values for all layers in Table I are chosen symmetrically for convenience, yet the gradients associated with the last-layer weight are actually nonnegative. In practice, this nonnegative characteristic can be exploited during learning.

To evaluate the importance of the parameter WIN_{STDP}, which is used in (10), simulations are conducted to examine the results obtained with different window sizes. Fig. 6 shows the correlations obtained between estimated gradients and numerical gradients. As shown in Fig. 6, estimating with different window sizes results in similar accuracies. Preliminary numerical studies on the effect of the window size on learning

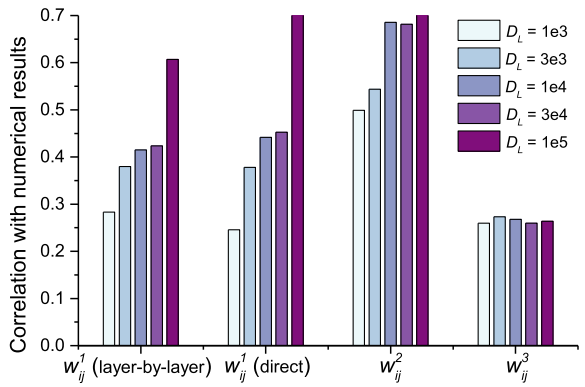

Fig. 7. Correlations between estimated gradients and gradients obtained with the FD numerical method. The results obtained for all three layers of synaptic weights (w_{ij}^1, w_{ij}^2, and w_{ij}^3) are compared. Two different backpropagation methods (layer by layer and direct) are also compared. Different evaluation durations are used. The longer the evaluation duration is, the more accurate the estimated gradients.

also show that changing the window size does not yield a noticeable difference. Therefore, in this paper, we focus on the case with a window size of one. How the size and shape of the summation window affect other aspects of learning will be a future research topic.

Another set of simulations is conducted to study how the evaluation duration affects the accuracy of the estimated gradients. As shown in Fig. 7, a general trend is that the longer the evaluation duration is, the more accurate the estimated gradients. This relationship is coherent with all of the stochastic approximation method because any possible unbiased noise is filtered out through averaging.

With gradients estimated through spike timings, a stochastic gradient descent method can be readily employed for learning. Following the convention in a standard backpropagation algorithm, we define an error function as:

$$E = \frac{1}{2}\sum_{k=1}^{N_o}\left(e_k^o\right)^2 \tag{19}$$

where $e_k^o = \overline{x_k^o} - t_k^o$ is the error at each output neuron. Here, t_k^o is the target mean firing rate of neuron x_k^o.

The weight update Δw_{ij}^l can be calculated as

$$\Delta w_{ij}^l = -\alpha \cdot \sum_{k=1}^{N_o} \frac{\partial E}{\partial \mu_k^o} \cdot \frac{\partial \mu_k^o}{\partial w_{ij}^l} = -\alpha \cdot \sum_{k=1}^{N_o} e_k^o \cdot \frac{\partial \mu_k^o}{\partial w_{ij}^l} \tag{20}$$

where α is the learning rate, and the term $\partial \mu_k^o / \partial w_{ij}^l$ can be obtained from (14). Updating weights according to (20) leads to a reduction in the error function toward the gradient-decent direction.

It is worthwhile to note that the proposed learning algorithm can be readily extended to other popular learning schemes, such as unsupervised learning and reinforcement learning, when the target is to minimize some forms of cost functions, even though this paper focuses mainly on supervised learning, which has achieved great success in real-life applications.

III. Learning Examples

In Section II, we demonstrate that spike timing information can be readily employed for estimating the gradient components needed in gradient descent algorithm. In this section, we apply the proposed learning algorithm to two neural networks. The sizes of the neural networks are chosen according to two examples demonstrated in [35] in such a way that a direct comparison can be made. The Modified National Institute of Standards and Technology database (MNIST) benchmark task [36] is employed to examine the proposed algorithm. The MNIST data set contains, in total, 70 000 28×28 images of handwritten digits. The number of images in the training and testing sets are 60 000 and 10 000, respectively. The data set is categorized into 10 classes, which correspond to ten integers (0–9), and each image has an associated label. Unless otherwise stated, for all of the learning examples in this section, we use a training set that contains the first 500 images from the standard MNIST training set to accelerate the simulation. For testing, we use all of the 10 000 images from the standard MNIST testing set. It should be noted that the results obtained with such an experiment setting are only for verifying the proposed techniques and exploring the design spaces. Benchmark performance obtained with the full training set is reported in Section III-D.

To feed the double-precision real values, which are used to encode the grayscale images, into the SNN, proper encoding mechanisms are needed. For SNNs attempting to learn the exact timing of spikes, temporal encoding is often employed, such as the single-spike temporal coding [24] and the temporal population coding [37]. In this paper, we use a pulse-density modulation scheme, which is a rate-based encoding method. Real values from the images in the MNIST data set are injected into the input-layer neurons as incremental membrane potentials at every tick. Combined with the modified LIF model, this encoding scheme behaves similar to a $\Sigma - \Delta$ modulator, which is capable of converting high-resolution data to low-resolution bit streams through pulse-density modulation [38]. The firing rates of the input-layer neurons are then proportional to the intensities of the corresponding pixels. Such an encoding method leads to a simple implementation in hardware while achieving the desired rate encoding.

For both neural networks, 10 output neurons, which correspond to 10 digits, are used. The target of learning is that when a digit is applied to the neural network, the output neuron that corresponds to the correct digit should fire with a high density of ρ_H, whereas all of the other neurons fire with a lower density of ρ_L. The firing density is measured through $\overline{x_k^o}$. To test the trained neural network, a digit is presented to the network. After an inference duration of D_I, the output neuron with the highest firing density $\overline{x_k^o}$ is chosen as the winning neuron, and its corresponding digit is chosen as the inferred result. All of the results presented in this section are obtained from 10 independent runs. Error bars that correspond to the 95% confidence interval are plotted together with the simulation data.

A. One-Hidden-Layer Neural Network

As most useful feed-forward neural networks have at least one hidden layer, the first example that we consider is a one-hidden-layer neural network with 784 input neurons, 300 hidden-layer neurons, and 10 output neurons. Neverthe-

less, neural networks with only two layers are also useful in some cases. For example, a two-layer neural network with a special input encoding similar to the radial basis function has been demonstrated in our previous work [31]. Because learning in a two-layer neural network is essentially a subproblem of learning in a multilayer (more than two) network, we do not study them in this paper separately. Nevertheless, most conclusions and techniques developed in this section can be readily applied to two-layer neural networks as well.

As demonstrated in [31], the gradients estimated from STDP started saturating and diverging from the actual gradients as the density of spike trains reaches a certain limit. This saturation occurs because it is difficult to tell whether a postsynaptic spike is a causal spike or an anti-causal spike when the presynaptic spike train is too dense. To tackle this issue, it was suggested in [31] that a clock that is fast enough to avoid dense spike trains should be used. This approach is similar to avoiding the hidden unit in a conventional ANN to be driven close to 1 or 0, which would otherwise lead to a significantly slowed learning process. Despite its effectiveness, this method of manually adjusting weights or the clock frequency is inconvenient. In this paper, we propose to leverage a biologically inspired refractoriness to achieve the desired sparsity. More specifically, each neuron has a refractory period after firing. During the refractory period, it is not allowed to fire again. By utilizing this technique, dense spike trains can be avoided. One potential drawback with a fixed refractory period is that all neurons that are saturated are highly correlated in their spike timings. To tackle this problem, a random refractory period technique is proposed. In a discrete-time implementation, it is convenient to implement according to the following equation:

$$x_j^{l+1}[n] = \begin{cases} 0, & V_j^{l+1}[n] < \text{th}_j^{l+1} \\ 1, & V_j^{l+1}[n] \geq \text{th}_j^{l+1} \ \& \ x_j^{l+1}[n-1] = 0 \\ 1-R, & V_j^{l+1}[n] \geq \text{th}_j^{l+1} \ \& \ x_j^{l+1}[n-1] = 1 \end{cases} \tag{21}$$

where R is a random variable with a Bernoulli distribution $B[1, p_r]$. Here, p_r is a design parameter that is used for controlling the sparsity. A larger p_r can lead to sparser spike trains.

Fig. 8 compares the learning results achieved with different initial weights and p_r. Two sets of initial weights are employed. One set of weights is initialized uniformly from the interval $[0, 2]$, i.e., $w_{ij}^l \sim U[0, 2]$, where $U[0, 2]$ stands for a uniform distribution between 0 and 2. The results obtained with these initial weights are labeled with "2×" in Fig. 8. Another set of weights is initialized such that $w_{ij}^l \sim U[0, 8]$. The results obtained with these initial weights are labeled with "8×" in Fig. 8. For small initial weights ($w_{ij}^l \sim U[0, 2]$), a reasonable learning result can be achieved even for the case in which $p_r = 0$ because saturation has been avoided through the proper choice of small initial weights. This circumstance corresponds to the case in which proper initial weights are chosen to avoid the hidden layer unit being driven close to 0 or 1 when training a conventional ANN. When the initial weights are large ($w_{ij}^l \sim U[0, 8]$),

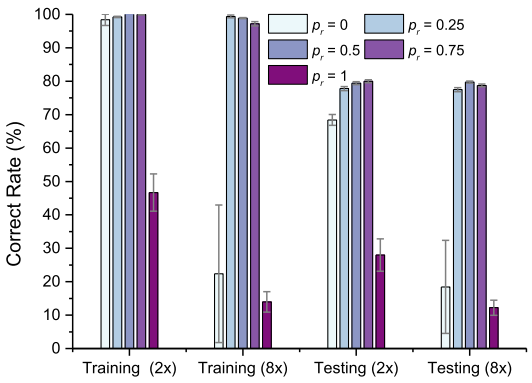

Fig. 8. Comparison of the training and testing correct rates achieved with different levels of refractoriness and different initial weights. The refractory mechanism is helpful in avoiding dense spike trains, which can improve the learning results. Two sets of initial weights are used. One set of weights is uniformly initialized between 0 and 2 (labeled with 2×), whereas another set of weights is uniformly initialized between 0 and 8 (labeled with 8×). Learning performance is severely deteriorated when a deterministic refractory period ($p_r = 1$) is used, as all saturated neurons have highly correlated spike timings.

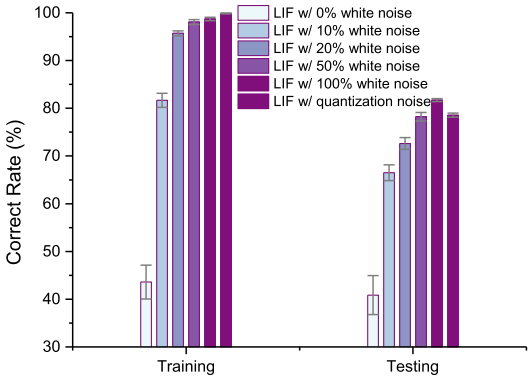

Fig. 9. Comparison of the training and testing correct rates achieved with the LIF neuron model and the modified LIF model. The results obtained with the conventional LIF model with white noise residue injection are labeled as "LIF w/white noise, " whereas the results obtained with the modified LIF model is labeled as "LIF w/quantization noise." Learning with the conventional LIF model is effective when enough noise is injected.

however, the learning performance is significantly deteriorated for the $p_r = 0$ case due to the aforementioned detrimental effect of saturated spike trains. It is noted that learning is not successful for the case $p_r = 1$ regardless of the selection of initial weights because neurons that are saturated always have high correlations in spike timings. Due to the proposed stochastic refractory period technique, good learning results are achieved when a proper p_r is employed. It should be mentioned that even though the initial weights are generated from positive uniform distributions in our learning examples, other initializations, such as negative weights and a normal distribution, can be used as well.

To study the effectiveness of the proposed learning algorithm applied to a conventional integrate-and-fire neuron model, simulations are conducted for different levels of noise injection, as shown in Fig. 9. Noise is injected into the neuron model as noisy residues. In other words, a random residue is added to the membrane voltage after each spike. The injected

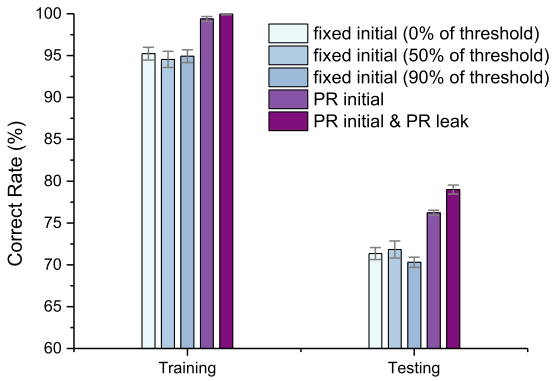

Fig. 10. Comparison of the training and testing correct rates achieved with different initial conditions. The case with pseudorandom initial membrane voltages outperforms the cases with fixed initial membrane voltages. A pseudorandom leakage technique is also employed to further improve the learning performance.

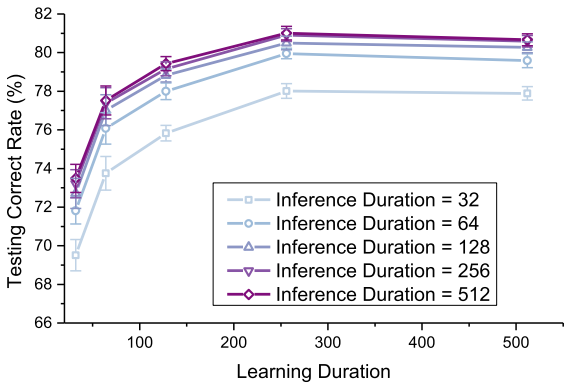

Fig. 11. Comparison of the testing correct rates achieved with different learning and inference durations. The longer the learning or inference duration is, the higher the correct rate.

noise is uniformly distributed with the range from zero to a percentage of the threshold value of that neuron. For example, the 50% white noise in Fig. 9 means that the noise injected into the neuron obeys a distribution, $U[0, 0.5 \times th_i^l]$. The results obtained with the modified integrate-and-fire model that are described in (17) and (18) are also shown for comparison. The corresponding results are labeled as "LIF w/ quantization noise." As the amount of injected noise increases, the learning is more effective. This result is expected because the proposed algorithm relies on the assumption that the spike timings of unconnected neurons should stay relatively uncorrelated. The conventional LIF model with noise injection can achieve a reasonably low correlation, yet random number generators are required for this purpose. On the other hand, the modified LIF model can decorrelate spike timings without explicitly injecting noise.

Another design consideration in our proposed learning algorithm is the initial condition of the neuron. In many applications, we need to reset neurons to certain states for each new input. Therefore, a proper initial condition needs to be set up. We propose to use a pseudorandom initial condition such that the initial membrane voltage of a neuron obeys a uniform distribution, e.g., $x_i^l[0] \sim U[0, \text{th}_i^l]$. The reason to choose such an initial condition is that the membrane voltages of an SNN in a steady state approximately follow a uniform distribution. Therefore, a warm start can be achieved by setting the initial condition as a uniformly distributed random variable. The results obtained with such a pseudorandom initial condition are compared with the fixed initial conditions in Fig. 10. As shown in Fig. 10, even though any initial condition can feature effective learning, the proposed pseudorandom initial condition achieves the best performance. The main reason that the random initial conditions outperform others is that such an initial condition helps to achieve lower correlations among the input spikes. For the MNIST data set, many pixels that correspond to strokes have values equal to one. This circumstance leads to highly correlated input spikes even when the modified LIF neuron model is used. By setting the initial condition differently, the correlations can be somewhat lowered.

With the same spirit, a pseudorandom leakage is also added at the input layer to further decorrelate the spike timings caused by the saturated intensities. The leakage for each neuron is assigned randomly beforehand and is fixed for the whole learning process. At each tick, the leakage is subtracted from the membrane voltage according to the neuron dynamics shown in (18). From another perspective, the pseudorandom leakage is helpful in breaking the possible symmetry that exists in the input data. Many input pixels from the MNIST data set have the value of one. Through introducing the random leakage, we can break this symmetry in the data. The symmetry-breaking technique has been widely used by many machine learning researchers for weight initialization [33] and asymmetric connections in convolutional neural networks (CNNs) [35]. The results obtained with this technique are also compared in Fig. 10. The advantage of pseudorandom initial conditions and leakage for neurons is that no pseudorandom/true-random number generators are actually needed in the hardware implementation. The values can be conveniently stored in an on-chip static random access memory array or can be hardcoded in the logic.

In Section II, it is shown that a longer learning duration yields more accurate estimated gradients. Therefore, it is expected that the learning performance can be improved through lengthening the learning duration. To investigate the effect of the learning duration on the learning performance, simulations are conducted, and the obtained results are compared in Fig. 11. In Fig. 11, five different learning durations are used: 32, 64, 128, 256, and 512. Five different inference durations are also used to evaluate the learned weights. A general trend shown in Fig. 11 is that increasing either the learning or inference duration helps in improving the recognition accuracy. For both the learning and inference duration, saturations occur at approximately 256, beyond which the improvement is marginal. Despite the fact that the best learning results are achieved when the learning duration is long, learning with a short duration also yields impressive results. This finding arises because stochastic gradient descent learning is quite robust against noise as long as it is not biased. Furthermore, it has been demonstrated recently that a noisy gradient is actually beneficial in learning, especially for

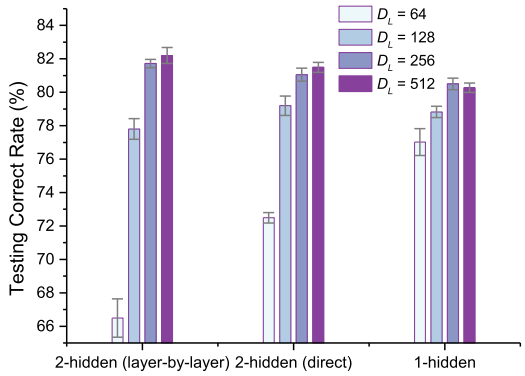

Fig. 12. Comparison of the testing correct rates achieved with the two different backpropagation schemes. The two methods achieve similar performances. The two-hidden-layer neural network can yield better performance, but it requires a longer learning duration.

a very deep neural network [39]. Therefore, a recommendation is to utilize a small learning duration at the beginning of the learning to speed up the learning process as well as to reduce the power consumption. The learning duration should be gradually lengthened to obtain more and more accurate gradients.

B. Two-Hidden-Layer Neural Network

The second example is a two-hidden-layer neural network with 784 neurons in the input layer, 300 neurons in the first hidden layer, 100 neurons in the second hidden layer, and 10 neurons in the output layer.

Because many conclusions that we draw for the one-hidden-layer neural network also apply to the two-hidden-layer neural network, we focus mainly on investigating how different methods of propagating the errors affect the learning performances. Simulations are conducted for the two different backpropagation methods discussed in Section II: the standard layer-by-layer backpropagation and the direct backpropagation. As shown in Fig. 12, similar performances are achieved by two backpropagation methods, which agree with the results shown in Figs. 6 and 7. For comparison purposes, we also plot the results obtained with a one-hidden-layer network in Fig. 12. The recognition rates achieved with the two-hidden-layer network are higher when a learning duration of a moderate length is used (specifically, $D_L \geq 256$ in Fig. 12). In addition, the two-hidden-layer neural network requires a longer learning duration to achieve a satisfactory result compared to its one-hidden-layer counterpart. This finding is consistent with the observation in ANNs that a deeper network tends to yield better results, yet it is harder and slower to train.

Even though the conventional layer-by-layer backpropagation can always be used along with our proposed algorithm, the unique direct backpropagation method can be helpful when the number of output-layer neurons is much smaller than the number of hidden-layer neurons, thereby providing more design freedom. For the lth layer in the network, $N_l N_{l+1}$ multiply-accumulate (MAC) operations are needed for a layer-by-layer backpropagation, whereas only $N_l N_o$ MAC operations are needed for a direct backpropagation. A significant savings in the number of MAC operations can be achieved when $N_o \ll N_l$. We do pay the price of spending more memory to store the STDP information across multiple layers. Therefore, we trade more memory space for fewer computations. The memory requirement for storing $cstdp^l_{jk}$ is $N_o \cdot \sum_{i=1}^{o-1} N_i$. Fortunately, this memory requirement does not scale as badly as the synaptic weights memory, which is on the order of $O(N^2)$, where N is the average number of neurons for each layer. For the neural networks that are employed in most applications, the output layer has far fewer neurons compared to the preceding layers. Indeed, a function of a deep neural network is to extract useful information from a high-dimensional input, layer by layer. Therefore, the number of output neurons in a typical neural network is on the order of $O(1)$. Consequently, the memory requirement for this type of error backpropagation is approximately on the order of $O(N)$.

C. Inference With a Progressive Precision

The results in the previous sections are obtained with inference methods that are mapped directly from those used in a conventional ANN, for simplicity. In other words, we wait until the output of the neural network converges to the steady-state result, and then, we read out the results. An SNN, however, provides new opportunities for more rapid estimation of the results. For example, if we train the neural network such that the output neuron that corresponds to the correct digit fires with a firing density of ρ_H, and other output neurons fire with a density of ρ_L. Then, we have a noise margin of $\rho_H - \rho_L$ such that a correct inference can still be achieved as long as the noise or any disturbance is less than this margin. Similar to the signal outputted by a $\Sigma - \Delta$ modulator, output signals from neurons are buried in high-frequency quantization noise. Counting the number of spikes is essentially filtering the high-frequency noise. A longer inference duration can lead to less quantization noise, and consequently, a more reliable result. This finding is similar to the well-known progressive precision in the stochastic computation [40].

Suppose that we apply an image to a well-trained network. When the image is simple (in the sense that it is easy to be recognized), one output neuron in the well-trained neural network, which corresponds to the digit presented, should spike with a firing density of ρ_H, and all of the other output neurons should fire with a density ρ_L. In this case, the signal strength is strong, and we do not have to wait until the quantization noise is removed. On the other hand, when the input image is complex (in the sense that it is difficult to recognize), then more than one output neuron may have high spike densities, which indicates that it could be one of these digits. In this case, the quantization noise might severely deteriorate the recognition accuracy, and thus, one should wait longer to filter out the high-frequency quantization noise. This process is similar to how humans accomplish recognition. When the classification problem is easy, the response time is short, whereas a longer time is needed when the pattern is complicated.

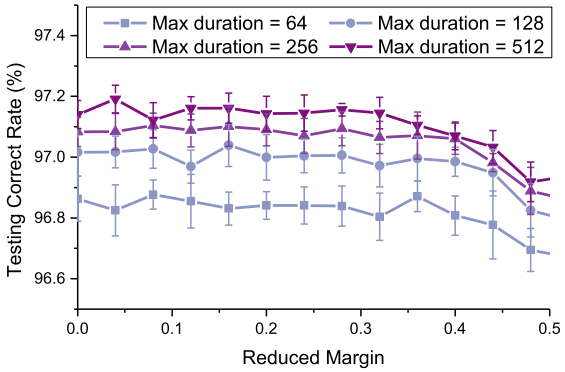

Fig. 13. Recognition accuracies on the testing set for different levels of reduced margin. Only a slight degradation in the correct rate is observed as the margin reduces. The results are obtained with the one-hidden-layer neural network.

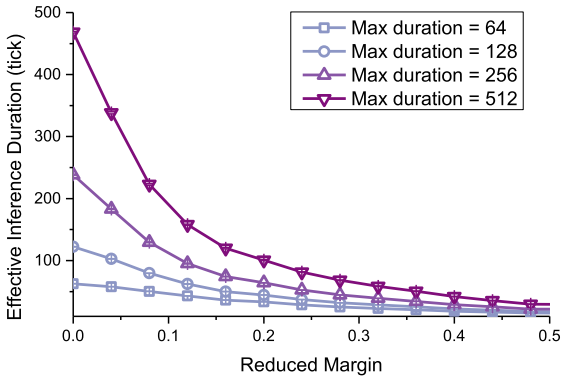

Fig. 14. Comparison of the effective inference durations needed for classifications. The number of ticks that is needed to complete one inference decreases significantly as the margin reduces. The results are obtained with the one-hidden-layer neural network.

Figs. 13 and 14 show the testing correct rate and effective inference durations that are needed to complete one classification. A trained one-hidden-layer neural network is used for illustration. At each tick, the density outputted from each of the output neurons is computed. If the density of one neuron is larger than $\rho_H - M/2$, and the densities from all of the other neurons are less than $\rho_L + M/2$, then the inference is considered to be completed, and the output neuron with the largest spike density is chosen as the answer, where M is the reduced margin. Otherwise, the inference continues until a maximum allowed inference duration is reached. The effective inference duration in Figs. 13 and 14 is obtained by averaging the inference durations in 10 000 testing cases. As shown in the figure, as the margin reduces, the length of the effective inference duration is significantly shortened. The classification accuracy, however, does not start dropping until the reduced margin reaches 0.3, where the quantization noise starts having a noticeable effect on the testing results. It should be noted that there exist some testing cases where the neural network is not able to give a confident answer regardless of how long the inference duration is. For these testing cases, the results are always produced when the maximum allowed inference duration is reached. Therefore, a trend is that the longer the maximum inference duration is, the longer the effective inference duration.

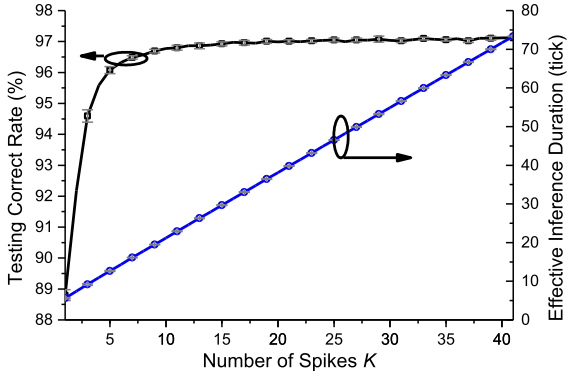

Fig. 15. Recognition accuracies on the testing set with the first-to-spike-K-spikes readout. The digit corresponding to the output neuron that first emits K spikes is readout. The black curve and the blue curve show the recognition correct rates and the average inference durations with different values of K. The results are obtained with the one-hidden-layer neural network.

Another way to demonstrate the inference with a progressive precision is shown in Fig. 15. The first output neuron that generates K spikes is determined to be the winning neuron, and the corresponding digit is read out as the inferred result. The recognition rates on the testing set images are shown in Fig.15 for different values of K. The number of ticks needed before an inference can be obtained is also recorded. As shown in Fig. 15, an accuracy as high as 89% can be achieved with an effective inference duration of only 5.6 ticks. The accuracy enhances rapidly when K increases. The growth in accuracy starts saturating when K reaches 10 in Fig. 15.

D. MNIST Benchmark

To demonstrate the effectiveness of the proposed learning algorithm, the standard MNIST benchmark is employed. Here, 60 000 training data are used for training a one-hidden-layer neural network and a two-hidden-layer neural network. The trained networks are examined with the testing set, which includes 10 000 digits. No preprocessing technique is used for a fair comparison. The obtained testing results for these two networks are compared with the results in the literature in Table II. Classification accuracies of 97.2% and 97.8% are achieved by the two neural networks, respectively. The proposed learning algorithm can achieve better classification correct rates compared to ANNs with the same configurations that are trained with sophisticated algorithms. Compared to the state-of-the-art result 98.6% in [15], our result is only slightly worse, especially considering that the size of our neural network is 9 times smaller than the network used in [15] in terms of the number of synapses. Moreover, different from the ANN-to-SNN conversion method employed in [15], our proposed learning algorithm can conduct an online learning directly on hardware SNNs, which is an expected feature in many energy-stringent applications.

In Table II, unsupervised learning in [18] and [13] and the contrast divergence learning in [16] are similar to clustering. Other decision logics in addition to the neural network are needed to perform the classification. Moreover, it is not

TABLE II
COMPARISON OF THE CLASSIFICATION ACCURACIES FOR THE MNIST BENCHMARK TASK

Reference	Preprocessing	Type	Configuration	Learning algorithm	Recognition accuracy
[18]	None	Spiking neural network	784 input neurons + 6400 excitatory neurons + 6400 inhibitory neurons + readout circuit	Unsupervised, STDP	95.0%
[13]	None	Spiking neural network	784 input neurons + 300 output neurons + readout circuit	Unsupervised, STDP	93.5%
[17]	Random translation, rotation, and scaling to extend the training set to 120000 images	Spiking deep belief network	784-500-500-10	ANN to SNN mapping	94.09%
[25]	Convert image into AER events and add noise	Spiking convolutional neural network	-	Tempotron	91.29%
[15]	None	Spiking neural network	784-1200-1200-10	ANN to SNN mapping	98.6%
		Spiking convolutional neural network	28x28-12c5-2s-64c5-2s-10o		99.1%
[16]	None	Spiking deep belief Network	484-256 + linear classifier	ANN to SNN mapping	89%
[41]	Thresholding	Spiking deep belief Network	784-500-40	Contrastive divergence	91.9%
[42]	Thresholding	Spiking neural network	784 input + 10 output neurons each with 200 dendrites and 5000 synapses	Morphological learning	90.26%
[43]	None	Spiking neural network	784-500-10	Contrastive divergence	95.6%
[35]	None	Artificial neural network	784-300-10	Stochastic gradient descent	95.3%
			784-300-100-10		96.95%
This paper	None	Spiking neural network	784-300-10	Stochastic gradient descent through modulation of weight-dependent STDP	97.2%
			784-300-100-10		97.8%

obvious how these learning algorithms can be used to train a universal function approximator that can be employed in various applications, e.g., reinforcement learning.

IV. CONSIDERATIONS FOR HARDWARE IMPLEMENTATION

In this section, we consider how the proposed learning algorithm can fit into various hardware implementations. Discussions on trade-offs and design considerations are also briefly presented. There are two popular hardware architectures for neuromorphic computing. One architecture is what we call a centralized memory architecture. An example of this type of system is shown in Fig. 16. This architecture is closely related to the conventional von Neumann architecture. Synaptic weights are stored in a memory array, and they can be accessed through buses. Another architecture, the distributed memory architecture, is more related to biological neural networks. Memory cells, in this case, are distributed along with processing units. This architecture is very popular in recent years due to many emerging memory technologies, such as memristor and phase-change memory.

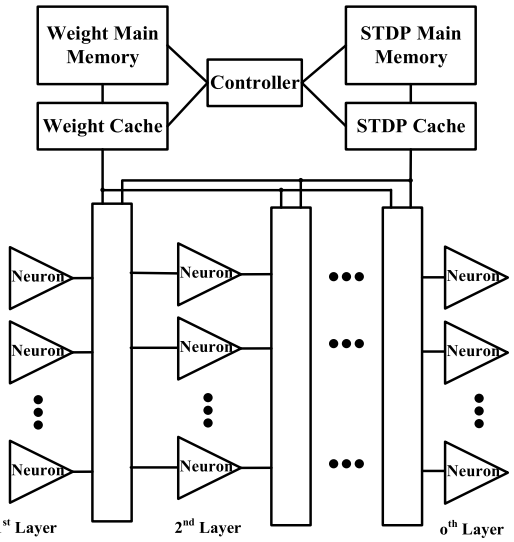

Fig. 16. Illustration of an example of the centralized memory architecture. Weights and STDP information are stored in a centralized memory, and they can be visited through buses.

Ways in which the proposed learning algorithm can be applied could be different for these two architectures, as shown in the following equations:

$$\Delta W_{ij}^{l} = -\alpha \cdot e_j^{l+1} \cdot \frac{\overline{stdp_{ij}^{l}}}{w_{ij}^{l}\left(1-\overline{x_i^{l}}\right)} \tag{22}$$

$$\Delta W_{ij}^{l} = \sum_{n} \left(\frac{-\alpha \cdot e_j^{l+1} \cdot stdp_{ij}^{l}[n]}{w_{ij}^{l}\left(1-\overline{x_i^{l}}\right)(D_L - T)} \right) \tag{23}$$

where ΔW_{ij}^{l} is the total weight change in one learning iteration, and $e_j^{l+1} = \sum_{k=1}^{N_o} e_k^{o} \cdot (\partial \mu_k^{o} / \partial \mu_j^{l+1})$ is the back-propagated error at neuron x_j^{l+1}.

In (22), the STDP information is accumulated over one iteration, and then, the weight change for that iteration is calculated. It is desirable to minimize the number of weight memory accesses in a centralized memory architecture because

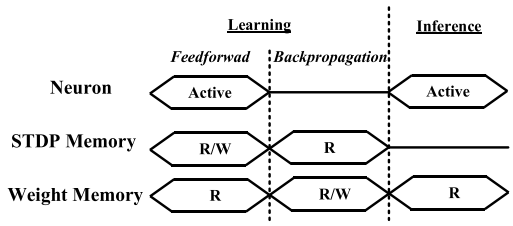

Fig. 17. Example of the timing diagram of the proposed learning algorithm employed in a centralized memory system.

the access of memory is a relatively expensive operation. Therefore, (22) can be used for weight updating in this case. On the other hand, in (23), the weight updating occurs whenever there is an STDP event. The summation of the weight change in one iteration occurs in the weight memory itself. Consequently, there is no need to store STDP information separately. Writing memory in a distributed memory system can be conducted easily with the help of auxiliary circuits associated with neurons while the system is operating. Therefore, (23) is more suitable for a distributed memory system. It should be noted that the above recommendations are not absolute. It is possible to use (22) in a distributed memory system and (23) in a centralized memory system, depending on the actual need.

The main focus of this paper is on the learning algorithm itself rather than its hardware implementation. Detailed hardware implementation is a future research direction. Therefore, only a brief discussion on how the proposed learning algorithm can be employed in a hardware realization is presented in this section.

A. Centralized Memory Architecture

Fig. 16 shows a generic diagram of a centralized memory architecture. This architecture is very popular in CMOS implementations [2], [44], [45]. In Fig. 16, neurons with dedicated computational resources are used for illustration because this configuration applies to both analog neurons [44], [45] and digital neurons [2]. Nevertheless, the discussions presented here also apply to virtual, time-multiplexed neurons, where the computational resources are shared among neurons.

Fig. 17 shows a typical timing diagram of operations and resource usages for the proposed learning algorithm. During the learning phase, feed-forward computations are first conducted. Spike timing information is generated and stored in the STDP memory. In the backpropagation phase, errors at the output neurons are propagated back to each synapse, and the weights are updated according to (22). During an inference phase, STDP memory is not used, and it can be power gated to save energy.

In the proposed learning algorithm, an array of memory is required to store the spike timing information. In the worst case, the spike timing information that is associated with each synapse needs to be stored. Fortunately, the number of bits that is needed to represent the STDP information is much less than the number needed for the synaptic weights. For example, in a fixed-point implementation, even though only

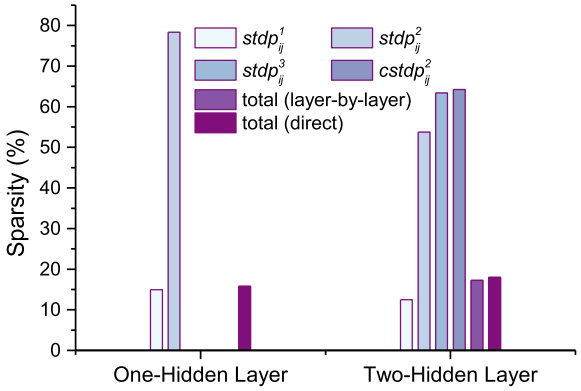

Fig. 18. Percentage of the active (nonzero) STDP information $stdp_{ij}^l$ and cross-layer STDP information $cstdp_{ij}^l$ for the synapses in different layers. The results are normalized to the number of synaptic weights in the corresponding layer.

a few bits are necessary for inference, more than 20 bits are usually needed to represent the weights for successful learning. The reason is that the weight changes are typically controlled to be 10^{-3} of the weights [33], and the ratio between the maximum and minimum weight in a learning task, such as the MNIST benchmark task that we consider, is normally a thousand. On the other hand, at most $\lceil \log_2(2D_L \cdot \mathrm{WIN}_{\mathrm{STDP}} + 1) \rceil$ bits are needed to store the spike timing information. The actual memory requirement, however, is much less than this upper bound, considering the sparseness of spikes. It is found through simulations that only 5 bits are required to represent the STDP information for the case of $\mathrm{WIN}_{\mathrm{STDP}} = 1$ and $D = 128$.

Furthermore, in contrast to synaptic weights that must be stored as "static" information in the memory even for inactive synapses, only the STDP information that is associated with recently active synapses is stored. This type of "dynamic" information storage can be leveraged to reduce the memory requirement and help improve the memory access latency. To illustrate this aspect, Fig. 18 shows the percentage of active (nonzero) STDP field in the memory for the synapses in each layer. Handwritten digits from the MNIST database are used as the input. It is shown in Fig. 18 that only approximately 15% of the synapses in the first layer are active. This finding occurs because only a few input neurons that are associated with strokes of a handwritten digit are active. For deeper layers, the percentage of active synapses is higher, yet there still exists some sparsity. Furthermore, in our implementation, we do not impose any sparsity regulation. In some applications, however, learning with a sparsity requirement is needed. In that case, the overall activity of the synapses can be further reduced. This sparsity in the spike timing information can be leveraged in a memory hierarchy to reduce the memory requirement significantly.

B. Distributed Memory Architecture

An example of the distributed memory architecture is shown in Fig. 19. Memristors are used in Fig. 19 for demonstration. Nevertheless, discussions in this section also apply to many

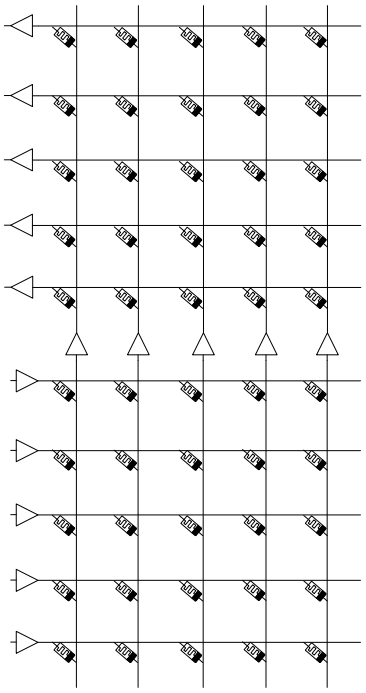

Fig. 19. Example of the distributed memory-based neuromorphic system. A crossbar structure consists of memristors is used for demonstration.

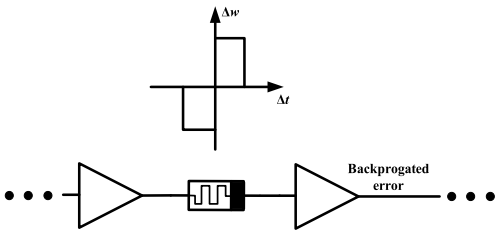

Fig. 20. Illustration of a memristor-based synapse sandwiched by two neurons. The STDP protocol can be implemented by either the neuron circuitry or the device itself.

other emerging memory technologies. In Fig. 19, a crossbar architecture is employed. Triangles in Fig. 19 represent neurons, and the memristor at each cross point represents a synapse that connects two neurons. When a neuron decides to spike, a voltage pulse is emitted by the neuron. The voltage potential across the memristors induces current flowing between the presynaptic neuron and the postsynaptic neuron. The magnitude of the current is determined by the conductance of the memristor. The currents flowing to the same postsynaptic neuron are summed up and accumulated on the capacitor associated with the neuron.

In a distributed memory architecture, the updating of the weights can occur when the SNN is operating. The error is back propagated to each neuron, and the weight update is then conducted according to (23). This arrangement is illustrated in Fig. 20. Here, (23) is essentially a scaled version of STDP divided by the weight. Let us ignore the denominator for now. There are various methods discussed in the literature that involve implementing neurons with STDP [11], [14]. In addition, there are also memristors that possess such a property on

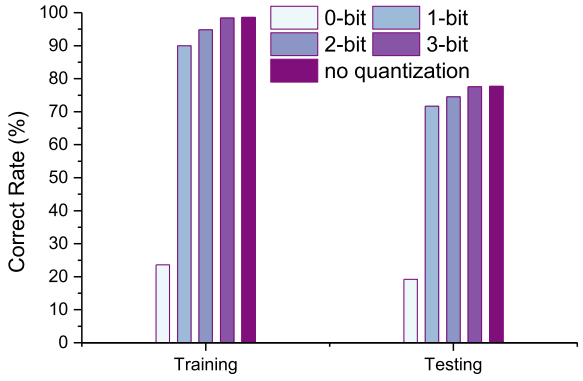

Fig. 21. Comparison of the classification accuracies for different levels of quantization in the weight denominator. Impressive results can be achieved even with a one-bit precision.

their own [8], [10]. Multiplying the error and learning rate can be easily achieved through controlling the pulsewidth or signal strength. In addition, stochastic multiplication is a candidate for this purpose.

Another matter needs to be addressed is the denominator shown in (23). The term $(1 - x_i^l)$ can be neglected when the firing rate of the neuron is low, and the constant term $(D_L - T)$ can be absorbed into the learning rate. It is not obvious how the division by the weight operation can be included in the memristance update. Fortunately, it has been shown in [31] that learning is still possible without accurately accounting for this denominator. This finding is due to the well-known robustness of the stochastic gradient descent method against noisy gradients. To illustrate this aspect, Fig. 21 compares the learning results that are obtained with different levels of precision in the denominator. The denominator is quantized into different numbers of bits to show how the inaccuracy of this division affects the learning performance. The weight in the denominator is quantized according to the following equation:

$$w_{ij}^l = \begin{cases} \cdots & \\ w_{q1}, & w_{q1} \le w_{ij}^l < w_{q2} \\ w_{q0}, & 0 \le w_{ij}^l < w_{q1} \\ -w_{q0}, & -w_{q1} \le w_{ij}^l < 0 \\ -w_{q1}, & -w_{q2} \le w_{ij}^l < -w_{q1} \\ & \cdots \end{cases} \tag{24}$$

where w_{q0}, w_{q1}, and so on are manually chosen constants that are used to partition the quantization intervals.

As shown in Fig. 21, impressive learning results can even be achieved with only a one-bit precision. In other words, only the sign information is preserved. This one-bit-precision division can be easily implemented with memristors. More specifically, an STDP rule is used for the excitatory synapses, whereas an anti-STDP rule is used for the inhibitory synapses. It is worthwhile to note that a memristor device cannot represent a negative weight by its nature. However, we can utilize a combination of two memristor devices to form a synapse that can be programmed to be either excitatory or inhibitory.

In addition, to feature better learning, a more accurate weight division can be explored. This goal can be achieved at both the circuit level and the device level.

V. Conclusion

In this paper, we formulate an online learning algorithm for multilayer SNNs. The proposed learning method can estimate the gradient components with the help of the spike timings in an SNN. The readily available gradient information is then exploited in stochastic gradient descent learning. How the error can be propagated back to each layer is studied. A direct backpropagation is proposed in addition to the conventional layer-by-layer approach. The newly proposed algorithm is employed in two neural networks for the purposes of demonstration. To feature more effective learning, techniques such as random refractory period and pseudorandom initial conditions are proposed. Furthermore, the progressive precision provided by a trained SNN is leveraged to accelerate the inference process. Extensive parametric studies are conducted to verify the proposed techniques as well as to examine many aspects of the proposed learning rules. To further demonstrate the effectiveness of the proposed algorithm, the MNIST benchmark test is conducted. Recognition accuracies of 97.2% and 97.8% are achieved with the neural networks trained by the proposed algorithm. Last, how the proposed learning rules can be implemented in hardware is discussed, and several trade-offs are identified. Detailed implementation of the proposed algorithm in hardware will be a future research direction.

Acknowledgment

The authors would like to thank the anonymous reviewers and the Associate Editor for their valuable comments and suggestions on improving the quality of this paper.

References

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

[2] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

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

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

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

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

[7] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[8] C. Du, W. Ma, T. Chang, P. Sheridan, and W. D. Lu, "Biorealistic implementation of synaptic functions with oxide memristors through internal ionic dynamics," *Adv. Funct. Mater.*, vol. 25, no. 27, pp. 4290–4299, 2015.

[9] 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.*, vol. 10, no. 4, pp. 1297–1301, 2010.

[10] S. Kim, C. Du, P. Sheridan, W. Ma, S. Choi, and W. D. Lu, "Experimental demonstration of a second-order memristor and its ability to biorealistically implement synaptic plasticity," *Nano Lett.*, vol. 15, no. 3, pp. 2203–2211, 2015.

[11] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "STDP and STDP variations with memristors for spiking neuromorphic learning systems," *Front. Neurosci.*, vol. 7, p. 2, 2013.

[12] H. Kim, M. P. Sah, C. Yang, T. Roska, and L. O. Chua, "Memristor bridge synapses," *Proc. IEEE*, vol. 100, no. 6, pp. 2061–2070, Jun. 2012.

[13] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Trans. Nanotechnol.*, vol. 12, no. 3, pp. 288–295, May 2013.

[14] T. Serrano-Gotarredona, T. Prodromakis, and B. Linares-Barranco, "A proposal for hybrid memristor-CMOS spiking neuromorphic learning systems," *IEEE Circuits Syst. Mag.*, vol. 13, no. 2, pp. 74–88, 2nd Quart., 2013.

[15] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.

[16] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45 pJ per spike in 45 nm," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Sep. 2011, pp. 1–4.

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

[18] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers Comput. Neurosci.*, vol. 9, p. 99, Aug. 2015.

[19] T. Masquelier, "Relative spike time coding and STDP-based orientation selectivity in the early visual system in natural continuous and saccadic vision: A computational model," *J. Comput. Neurosci.*, vol. 32, no. 3, pp. 425–441, 2012.

[20] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput. Biol.*, vol. 3, no. 2, p. e31, 2007.

[21] S. M. Bohte, J. N. Kok, and H. L. Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, nos. 1–4, pp. 17–37, 2002.

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

[23] R. Gütig and H. Sompolinsky, "The tempotron: A neuron that learns spike timing-based decisions," *Nature Neurosci.*, vol. 9, no. 3, pp. 420–428, 2006.

[24] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Rapid feedforward computation by temporal encoding and learning with spiking neurons," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 10, pp. 1539–1552, Oct. 2013.

[25] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward categorization on AER motion events using cortex-like features in a spiking neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 9, pp. 1963–1978, Sep. 2015.

[26] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns," *PLoS ONE*, vol. 8, no. 11, p. e78318, 2013.

[27] Q. Yu, R. Yan, H. Tang, K. C. Tan, and H. Li, "A spiking neural network system for robust sequence recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 621–635, Mar. 2016.

[28] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[30] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, Feb. 2010.

[31] N. Zheng and P. Mazumder, "Hardware-friendly actor-critic reinforcement learning through modulation of spike-timing-dependent plasticity," *IEEE Trans. Comput.*, vol. 62, no. 1, pp. 299–311, Feb. 2017.

[32] J. C. Spall, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins APL Tech. Dig.*, vol. 19, no. 4, pp. 482–492, 1998.

[33] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 599–619.
[34] A. S. Cassidy *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Aug. 2013, pp. 1–10.
[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[36] Y. LeCun and C. Cortes. *The MNIST Database of Handwritten Digits*. Accessed: Oct. 18, 2017 [Online]. Available: http://yann.lecun.com/exdb/mnist/
[37] J. Hu, H. Tang, K. C. Tan, and H. Li, "How the brain formulates memory: A spatio-temporal model research frontier," *IEEE Comput. Intell. Mag.*, vol. 11, no. 2, pp. 56–68, May 2016.
[38] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2010.
[39] A. Neelakantan *et al.* (Nov. 2015). "Adding gradient noise improves learning for very deep networks." [Online]. Available: https://arxiv.org/abs/1511.06807
[40] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 12, p. 92, May 2013.
[41] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers Neurosci.*, vol. 7, p. 272, Jan. 2013.
[42] S. Hussain, S.-C. Liu, and A. Basu, "Improved margin multi-class classification using dendritic neurons with morphological learning," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 2640–2643.
[43] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, "Stochastic synapses enable efficient brain-inspired learning machines," *Frontiers Neurosci.*, vol. 10, p. 241, Jun. 2016.
[44] S. Moradi and G. Indiveri, "An event-based neural network architecture with an asynchronous programmable synaptic memory," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 1, pp. 98–107, Feb. 2014.
[45] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, and B. Linares-Barranco, "A neuromorphic cortical-layer microchip for spike-based event processing vision systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 12, pp. 2548–2566, Dec. 2006.

Nan Zheng (S'13) received the B.S. degree in information engineering from Shanghai Jiao Tong University, Shanghai, China, in 2011, and the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2014, where he is currently pursuing the Ph.D. degree in electrical engineering.

In 2012, he joined Qualcomm, San Diego, CA, USA, as an intern, where he was involved in developing an antenna system for the next-generation communication network. His current research interests include low-power circuit design, modeling, and optimization with an emphasis on machine-learning applications.

Pinaki Mazumder (S'84–M'87–SM'95–F'99) received the Ph.D. degree from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 1988.

He is currently a Professor with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA. He was for six years with industrial Research and Development centers that included AT&T Bell Laboratories, Naperville, IL, USA, where in 1985, he started the CONES Project—the first C modeling-based very large scale integration (VLSI) synthesis tool at India's premier electronics company, Bharat Electronics, Ltd., Bangalore, India, where he had developed several high-speed and high-voltage analog integrated circuits intended for consumer electronics products. He has authored or co-authored of more than 320 technical papers and five books on various aspects of VLSI research works. His current research interests include current problems in nanoscale CMOS VLSI design, computer-aided design tools, and circuit designs for emerging technologies including quantum MOS and resonant tunneling devices, semiconductor memory systems, and physical synthesis of VLSI chips.

Dr. Mazumder was a Fellow of the American Association for the Advancement of Science in 2008. He was a recipient of the Digital's Incentives for Excellence Award, the BF Goodrich National Collegiate Invention Award, and the Defense Advanced Research Projects Agency Research Excellence Award.