Supervised Learning Based on Temporal Coding in Spiking Neural Networks

Hesham Mostafa

***Abstract*— Gradient descent training techniques are remarkably successful in training analog-valued artificial neural networks (ANNs). Such training techniques, however, do not transfer easily to spiking networks due to the spike generation hard nonlinearity and the discrete nature of spike communication. We show that in a feedforward spiking network that uses a temporal coding scheme where information is encoded in spike times instead of spike rates, the network input–output relation is differentiable almost everywhere. Moreover, this relation is piecewise linear after a transformation of variables. Methods for training ANNs thus carry directly to the training of such spiking networks as we show when training on the permutation invariant MNIST task. In contrast to rate-based spiking networks that are often used to approximate the behavior of ANNs, the networks we present spike much more sparsely and their behavior cannot be directly approximated by conventional ANNs. Our results highlight a new approach for controlling the behavior of spiking networks with realistic temporal dynamics, opening up the potential for using these networks to process spike patterns with complex temporal information.**

***Index Terms*— Backpropagation, spiking networks, supervised learning.**

I. Introduction

ARTIFICIAL neural networks (ANNs) are enjoying great success as a means of learning complex nonlinear transformations [1]. The idea of a distributed network of simple neuron elements that adaptively adjusts its connection weights based on training examples is partially inspired by the operation of biological spiking networks [2]. ANNs, however, are fundamentally different from spiking networks. Unlike ANN neurons that are analog-valued, spiking neurons communicate using all-or-nothing discrete spikes. A spike triggers a trace of synaptic current in the target neuron. The target neuron integrates synaptic current over time until a threshold is reached, and then emits a spike and resets. Spiking networks are dynamical systems in which time plays a crucial role, while time is abstracted away in conventional feedforward ANNs.

ANNs typically make use of gradient descent techniques to solve the weight credit assignment problem [3], that is the problem of changing the network weights so as to obtain the desired network output. ANNs typically have multiple cascaded layers of neurons. In that case, the gradient of the error function with respect to the network weights can be efficiently obtained using the backpropagation algorithm. Having multiple layers of neurons is crucial in allowing ANNs to learn using backpropagation [4].

While backpropagation is a well-developed general technique for training feedforward ANNs, there is no general technique for training feedforward spiking neural networks. Many previous approaches that train spiking neural networks to produce particular spike patterns depend on having the input layer directly connected to the output layer [5], [6]. It is unclear how multilayer networks can be trained using these approaches. Stochastic network formulations are often considered when training temporal networks [7], [8]. In a stochastic formulation, the goal is to maximize the likelihood of an entire output spike pattern. The stochastic formulation is needed to 'smear out' the discrete nature of the spike, and to work instead with spike generation probabilities that depend smoothly on network parameters and are thus more suitable for gradient descent learning. While multilayer networks have been trained using this stochastic approach [8], there are several scalability concerns due to the need for time-stepped simulations to get output spike times. In some cases, Monte Carlo simulations are needed to obtain the likelihoods of different output patterns [7]. Moreover, in previous approaches, the goal is to learn particular spike patterns, and the performance of these networks in classification settings where the goal is to learn an input–output relation that generalizes well to unseen examples, which are not merely noise corrupted training examples, is left unexplored.

An approach that bears some similarities to ours is the SpikeProp algorithm [9] that can be used to train multilayer spiking networks to produce output spikes at specific times. SpikeProp assumes that a connection between two spiking neurons consists of a number of subconnections, each with a different delay and a trainable weight. We use a more conventional network model that does not depend on combinations of prespecified delay elements to transform input spike times to output spike times, and instead relies only on simple neural and synaptic dynamics. Unlike SpikeProp, our formulation results in an analytical relation between input and output spike times. The spiking network, thus, does not have to be simulated during the training loop. This accelerates training and allows us to make use of standard GPU-accelerated ANN training packages to scale the training to larger data sets.

Many approaches for training spiking networks first train conventional feedforward ANNs and then translate the trained

Manuscript received September 19, 2016; revised February 13, 2017; accepted July 9, 2017. This work was supported in part by the Samsung Advanced Institute of Technology, in part by the Neuromorphic Processor Project, and in part by the Swiss National Science Foundation under Grant P2ZHP2_1-6496O.

The author is with the Department of Bioengineering, Jacobs School of Engineering, Institute of Neural Computation, University of California, San Diego, CA 92093 USA (e-mail: hmmostafa@ucsd.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.2017.2726060

weights to spiking networks developed to approximate the behavior of the original ANNs [10]–[12]. The spiking networks obtained using these methods use rate coding where the spiking rate of a neuron encodes an analog quantity corresponding to the analog output of an ANN neuron. High spike rates would then mask the discrete nature of the spiking activity.

In this paper, we develop a direct training approach that does not try to reduce spiking networks to conventional ANNs. Instead, we relate the time of any spike differentiably to the times of all spikes that had a causal influence on its generation. We can then impose any differentiable cost function on the spike times of the network and minimize this cost function directly through gradient descent.

Using spike times as the information-carrying quantities, we avoid having to work with discrete spike counts or spike rates, and instead work with a continuous representation (spike times) that is amenable to gradient descent training. This training approach allows detailed control over the behavior of the network (at the level of single spike times) which would not be possible in training approaches based on rate coding.

Since we use a temporal spike code, neuron firing can be quite sparse as the time of each spike carries significant information. Compared to rate-based networks, the networks we present can be implemented more efficiently on neuromorphic architectures where power consumption decreases as spike rates are reduced [13]–[15]. In conventional ANNs, each neuron calculates a weighted sum of the activities of all its source neurons then produces its output by applying a static nonlinearity to this sum. We show that the behavior of the networks we present deviates quite significantly from this conventional ANN paradigm as the output of each neuron in one layer depends on a different and dynamically changing subset of the neurons in the preceding layer. Unlike rate-based spiking networks, the proposed networks can be directly trained using gradient descent methods, as time is the principal coding dimension, not the discrete spike counts. Unlike previous temporal learning approaches, our method extends naturally to multilayer networks, and depends on deterministic neural and synaptic dynamics to optimize the spike times, rather than explicit delay elements in the network.

II. Network Model

We use nonleaky integrate and fire neurons with exponentially decaying synaptic current kernels. The neuron's membrane dynamics are described by

$$\frac{dV_{\text{mem}}^{j}(t)}{dt} = \sum_{i} w_{ji} \sum_{r} \kappa(t - t_i^r) \tag{1}$$

where V_{mem}^{j} is the membrane potential of neuron j. The right-hand side of the equation is the synaptic current. w_{ji} is the weight of the synaptic connection from neuron i to neuron j and t_i^r is the time of the rth spike from neuron i. κ is the synaptic current kernel given by

$$\kappa(x) = \Theta(x) \exp\left(-\frac{x}{\tau_{\text{syn}}}\right) \quad \text{where} \quad \Theta(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Synaptic current thus jumps instantaneously on the arrival of an input spike, then decays exponentially with time constant τ_{syn}. Since τ_{syn} is the only time constant in the model, we set it to 1 in the rest of this paper, i.e., normalize all times with respect to it. The neuron spikes when its membrane potential crosses a firing threshold which we set to 1, i.e., all synaptic weights are normalized with respect to the firing threshold. The membrane potential is reset to zero after a spike. We allow the membrane potential to go below zero if the integral of the synaptic current is negative.

Assume that a neuron receives N spikes at times $\{t_1, \ldots, t_N\}$ with weights $\{w_1, \ldots, w_N\}$ from N source neurons. Each weight can be positive or negative. Assume that the neuron spikes in response at time t_{out}. By integrating (1), the membrane potential for $t < t_{\text{out}}$ is given by

$$V_{\text{mem}}(t) = \sum_{i=1}^{N} \Theta(t - t_i) w_i (1 - exp(-(t - t_i))). \tag{3}$$

Assume that only a subset of these input spikes with indices in $C \subseteq \{1, \ldots, N\}$ had arrived before t_{out}, where $C = \{i : t_i < t_{\text{out}}\}$. It is only these input spikes that influence the time of the output neuron's first spike. We call this set of input spikes the causal set of input spikes. The sum of the weights of the causal input spikes has to be larger than 1, otherwise they could not have caused the neuron to fire. From (3), t_{out} is then implicitly defined as

$$1 = \sum_{i \in C} w_i (1 - \exp(-(t_{\text{out}} - t_i))) \tag{4}$$

where 1 is the firing threshold. Hence

$$\exp(t_{\text{out}}) = \frac{\sum_{i \in C} w_i \exp(t_i)}{\sum_{i \in C} w_i - 1}. \tag{5}$$

Spike times always appear exponentiated. Therefore, we do a transformation of variables $\exp(t_x) \to z_x$ yielding an expression relating input spike times to the time of the first spike of the output neuron in the post-transformation domain (which we denote as the z-domain)

$$z_{\text{out}} = \frac{\sum_{i \in C} w_i z_i}{\sum_{i \in C} w_i - 1}. \tag{6}$$

Note that for the neuron to spike in the first place, we must have $\sum_{i \in C} w_i > 1$, so z_{out} is always positive (one can show this is the case even if some of the weights are negative). It is also always larger than any element of $\{z_i : i \in C\}$, i.e., the output spike time is always larger than any input spike time in the causal set which follows from the definition of the causal set. We can obtain a similar expression relating the time of the Qth spike of the output neuron, z_{out}^{Q}, to the input spike times in the z-domain

$$z_{\text{out}}^{Q} = \frac{\sum_{i \in C^Q} w_i z_i}{\sum_{i \in C^Q} w_i - Q} \tag{7}$$

where C^Q is the set of indices of the input spikes that arrive before the Qth output spike. Equation (7) is only valid if the denominator is positive, i.e., there are sufficient input spikes with large enough total positive weight to push the neuron past the firing threshold Q times. In the rest of this paper,

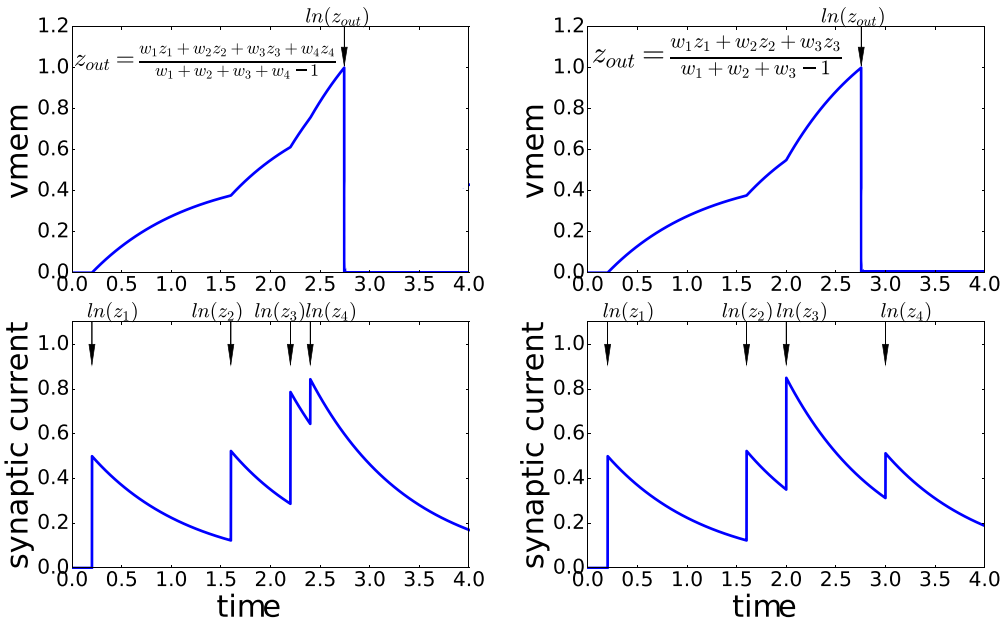

Fig. 1. Changes in the causal input set modifies the linear input–output spike times relation. Plots show the synaptic current and the membrane potential of a neuron in two situations in (a) neuron receives four spikes with weights $\{w_1, w_2, w_3, w_4\}$ at the times indicated by the black arrows in the bottom plot. This causes the neuron to spike at time $t_{\text{out}} = ln(z_{\text{out}})$, (b) input spike times change causing the neuron to spike before the fourth input spike arrives. This changes the linear z-domain input–output relation of the neuron compared to (a). Note that a neuron is only allowed to spike once, after which it cannot spike again until the network is reset and a new input pattern is presented.

we consider a neuron's output value to be the time of its first spike. Moreover, once a neuron spikes, it is not allowed to spike again, i.e., we assume it enters an infinitely long refractory period. We allow each neuron to spike at most once for each input presentation in order to make the spiking activity sparse and to force the training algorithm to make optimal use of each spike. During training, we use a weight cost term that insures the neuron receives sufficient input to spike as we describe in the next section.

The linear relation between input and output spike times in the z-domain is only valid in a local interval. A different linear relation holds when the set of causal input spikes changes. This is illustrated in Fig. 1, where the fourth input spike is part of the causal set in one case but not in the other. From (6), the effective weight of input z_p in the linear input–output relation in the z-domain is $w_p/(\sum_{i\in C} w_i - 1)$. This effective weight depends on the weights of the spikes in the causal set of input spikes. As this causal set changes due to the changing spike times from the source neurons, the effective weight of the different input spikes that remain in the causal set changes [as is the case for the first three spikes in Fig. 1(a) and (b) whose effective weight changes as the causal set changes, even though their actual synaptic weights are the same]. For some input patterns, some source neurons may spike late causing their spikes to leave the causal set of the output neuron and their effective spike weights to become zero. Other source neurons may spike early and influence the timing of the output neuron's spike and thus their spikes acquire a nonzero effective weight.

The causal set of input spikes is dynamically determined based on the input spike times and their weights. Many early spikes with strong positive weights will cause the output neuron to spike early, negating the effect of later spikes on the output neuron's first spike time regardless of the weights of these later spikes. The nonlinear transformation from $\mathbf{z} = \{z_1, \ldots, z_N\}$ to z_{out} implemented by the spiking neuron is thus fundamentally different from the static nonlinearities used in traditional ANNs where only the aggregate weighted input is considered.

The nonlinear transformation implemented by the spiking neuron is continuous in most case, i.e., small perturbation in $\mathbf{z}$ will lead to proportionately small perturbations in z_{out}. This is clear when the perturbations do not change the set of causal input spikes as the same linear relation continues to hold. Consider, however, the case of an input spike with weight w_x that occurs just after the output spike at time $z_x = z_{\text{out}} + \epsilon$. A small perturbation pushes this input spike to time $z_x = z_{\text{out}} - \epsilon$ adding it to the causal set. By applying (6), the perturbed output time is $z_{\text{out}}^{\text{perturb}} = (\sum_{i\in C} w_i z_i + w_x z_x)/(\sum_{i\in C} w_i + w_x - 1)$ where C is the causal set before the perturbation. Substituting for z_x, $z_{\text{out}}^{\text{perturb}} - z_{\text{out}} = -\epsilon w_x/(\sum_{i\in C} w_i + w_x - 1)$. The output perturbation is thus proportional to the input perturbation but this is only the case when $\sum_{i\in C} w_i + w_x > 1$, otherwise the perturbed input spike with negative weight at $z_{\text{out}} - \epsilon$ would cancel the original output spike at z_{out}. In summary, the input spike times to output spike time transformation of the spiking neuron is continuous except in situations where small perturbations affect whether a neuron spikes or not.

The fact that we get a piecewise linear relation between input and output spike times in the z-domain is a general

Algorithm 1 Forward Pass in a Feedforward Spiking Network With L Layers

Input: $\mathbf{z^0}$: Vector of input spike times
Input: $\{N^1, \ldots, N^L\}$: Number of neurons in the L layers
Input: $\{W^1, \ldots, W^L\}$: Set of weight matrices. $W^l[i, j]$ is the weight from neuron j in layer $l-1$ to neuron i in layer l
Output: $\mathbf{z^L}$: Vector of first spike times of neurons in the top layer
for r=1 to L **do**
  for i=1 to N^r **do**
    $C_i^r \leftarrow get\_causal\_set(\mathbf{z^{r-1}}, W^r[i,:])$
    if $C_i^r \neq \Phi$ **then**
      $\mathbf{z^r}[i] \leftarrow \frac{\sum_{k\in C_i^r} W^r[i,k]\mathbf{z^{r-1}}[k]}{\sum_{k\in C_i^r} W^r[i,k]-1}$
    else
      $\mathbf{z^r}[i] \leftarrow \infty$
    end if
  end for
end for

Algorithm 2 Get_Causal_Set: Gets Indices of Input Spikes Influencing First Spike Time of Output Neuron

Input: $\mathbf{z}$: Vector of input spike times of length N
Input: $\mathbf{w}$: Weight vector of the input spikes
Output: C: Causal index set
$\mathbf{sort\_indices} \leftarrow argsort(\mathbf{z})$ //Ascending order argsort
$\mathbf{z^{sorted}} \leftarrow \mathbf{z}[\mathbf{sort\_indices}]$ //sorted input vector
$\mathbf{w^{sorted}} \leftarrow \mathbf{w}[\mathbf{sort\_indices}]$ //weight vector rearranged to match sorted input vector
for i=1 to N **do**
  if $i == N$ **then**
    $next\_input\_spike \leftarrow \infty$
  else
    $next\_input\_spike \leftarrow \mathbf{z^{sorted}}[i+1]$
  end if
  if $\sum_{k=1}^{i} \mathbf{w^{sorted}}[k] > 1 \quad \wedge \quad \frac{\sum_{k=1}^{i} \mathbf{w^{sorted}}[k]\mathbf{z^{sorted}}[k]}{\sum_{k=1}^{i} \mathbf{w^{sorted}}[k]-1} < next\_input\_spike$ **then**
    return $\{\mathbf{sort\_indices}[1], \ldots, \mathbf{sort\_indices}[i]\}$
  end if
end for
return Φ

phenomenon and not an accident of our model. Scaling the input vector $\mathbf{z} = [z_1, \ldots, z_N]$ by a factor K is equivalent to shifting all input spike times in the time domain, $\{t_1, \ldots, t_N\}$, forward by time $\ln(K)$. This would shift the output spike time, t_{out}, forward by time $\ln(K)$ as well since the network has no internal time reference and shifting all input spikes in time would thus shift all output spikes by the same amount. This would in turn linearly scale z_{out} by a factor K, realizing a (locally) linear input–output relation.

III. Training

We consider feedforward neural networks where the neural and synaptic dynamics are described by (1) and (2). The neurons are arranged in a layer-wise manner. Neurons in one layer project in an all-to-all fashion to neurons in the subsequent layer. A neuron's output is the time of its first spike and we work exclusively in the z-domain. Once a neuron spikes, it is not allowed to spike again until the network is reset and a new input pattern is presented. The forward pass is described in Algorithm 1. z^r is the vector of the first spike times from each neuron in layer r. All indices are 1-based, vectors are in boldface, and the ith entry of a vector $\mathbf{a}$ is $\mathbf{a}[i]$. At each layer, the causal set for each neuron is obtained from the get_causal_set function. The neuron output is then evaluated according to (6). The get_causal_set function is shown in Algorithm 2. It first sorts the input spike times, and then considers increasingly larger sets of the early input spikes until it finds a set of input spikes that causes the neuron to spike. The output neuron must spike at a time that is less than the time of any input spike not in the causal set, otherwise the causal set is incomplete. If no such set exists, i.e., the output neuron does not spike in response to the input spikes, get_causal_set returns the empty set Φ and the neuron's output spike time is set to infinity (maximum positive value in implementation).

From (6), the derivatives of a neuron's first spike time with respect to synaptic weights and input spike times are given by

$$\frac{dz_{\text{out}}}{dw_p} = \begin{cases} \frac{z_p - z_{\text{out}}}{\sum_{i\in C} w_i - 1} & \text{if} \quad p \in C \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$\frac{dz_{\text{out}}}{dz_p} = \begin{cases} \frac{w_p}{\sum_{i\in C} w_i - 1} & \text{if} \quad p \in C \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Unlike the spiking neuron's input–output relation which can still be continuous at points where the causal set changes, the derivative of the neuron's output with respect to inputs and weights given by (8) and (9) is discontinuous at such points. This is not a severe problem for gradient descent methods. Indeed, many feedforward ANNs use activation functions with a discontinuous first derivative such as rectified linear units (ReLUs) [16] while still being effectively trainable.

A differentiable cost function can be imposed on the spike times generated anywhere in the network. The gradient of the cost function with respect to the weights in lower layers can be evaluated by backpropagating errors through the layers using (8) and (9) through the standard backpropagation technique. In the next section, we use the spiking network in a classification setting. In training the network, we had to use the following techniques to enable the networks to learn:

a) Constraints on synaptic weights: We add a term to the cost function that heavily penalizes neurons' input weight vectors whose sum is less than 1. During training, this term pushes the sum of the weights in each neuron's input weight vector above 1 which ensures that a neuron spikes if all its input neurons spike. This term in the cost function is crucial, otherwise the network can become quiescent and stop spiking.

This cost term has the form

$$WeightSumCost = K * \sum_j \max\left(0, 1 - \sum_i w_{ji}\right) \quad (10)$$

where the summation over j runs over all neurons and the summation over i runs over all the neurons that project to neuron j. w_{ji} is the connection weight from neuron i to neuron j. K is a hyper-parameter. K is typically chosen to be larger than 1 to strictly enforce the constraint that the incoming weight vector to each neuron sums to more than one. Large positive weights are problematic as they can enable a source neuron to almost unilaterally control the target neuron's spike time, compromising the ability of the target neuron to integrate information from all its input neurons. Therefore, we use L2 weight regularization to stop weights from becoming too large.

b) Gradient normalization: We observed that the gradients can become very large during training. This is due to the highly nonlinear relation between the output spike time and the weights when the sum of the weights for the causal set of input spikes is close to 1. This can be seen from (6), (8), and (9) where a small denominator can cause the output spike time and the derivatives to diverge. This hurts learning as it causes weights to make very large jumps. We use gradient normalization to counter that: if the Frobenius norm of the gradient of a weight matrix is above a threshold, we scale the gradient matrix so that its Frobenius norm is equal to the threshold before doing the gradient descent step. To reduce the dependence of this gradient normalization scheme on weight matrix size, we first normalize the Frobenius norm of the weight gradient matrix by the number of source neurons (number of rows in the weight matrix).

IV. Results

We trained the network to solve two classification tasks: an XOR task and the permutation invariant MNIST task [17]. We chose the XOR task to illustrate how the network is able to implement a nonlinear transformation since a linear network cannot solve the XOR task. The MNIST task was chosen to examine the generalization behavior of the network as the network is tested on input patterns that it has never seen before. We used fully connected feedforward networks where the top layer had as many neurons as the number of classes (two in the XOR task and ten in the MNIST task). The goal is to train the network so that the neuron corresponding to the correct class fires first among the top layer neurons. We used the cross-entropy loss and interpreted the value of a top layer neuron as the negative of its spike time (in the z-domain). Thus, by maximizing the value of the correct class neuron value, training effectively pushes this neuron to fire earlier than the neurons representing the incorrect classes. For an output spike times vector $\mathbf{z}^L$ and a target class index g, the loss function is given by

$$L(g, \mathbf{z^L}) = -\ln \frac{\exp(-\mathbf{z^L}[g])}{\sum_i \exp(-\mathbf{z^L}[i])}. \quad (11)$$

We used standard gradient descent to minimize the loss function across the training examples. Training was done using Theano [18], [19].

A. XOR Task

In the XOR task, two spike sources send a spike each to the network. Each of the two input spikes can occur at time 0.0 (early spike) or 2.0 (late spike). The two input spike sources project to a hidden layer of four neurons and the hidden neurons project to two output neurons. The first output neuron must spike before the second output neuron if exactly one input spike is an early spike. The network is shown in Fig. 2(a) together with the four input patterns.

To investigate whether the network can robustly learn to solve the XOR task, we repeated the training procedure 1000 times starting from random initial weights each time. In each of these 1000 training trials, we used as many training iterations as needed for training to converge. We used a constant learning rate of 0.1. The weight sum cost coefficient [K in (10)] is 10. We did not use L2 regularization. The maximum allowed row-normalized Frobenius norm of the gradient of a weight matrix is 10. Each training iteration involved presenting the four input patterns 100 times. Across the 1000 trials, the maximum number of training iterations needed to converge was 61 while the average was 3.48. Fig. 2(b) shows the post-training simulation results of the network when presented with each of the input patterns. The causal input sets of the different neurons change across the input patterns, allowing the network to implement the nonlinearity needed to solve the XOR task.

B. MNIST Classification Task

The MNIST database contains 70 000 28 x 28 grayscale images of handwritten digits. The training set of 60 000 labeled digits was used for training, and testing was done using the remaining 10 000. No validation set was used. All grayscale images were first binarized to two intensity values: high and low. Pixels with high intensity generate a spike at time 0, while pixels with low intensity generate a spike at time $\ln(6) = 1.79$, corresponding to $z = 6$ in the z-domain. $\ln(6) = 1.79$ was chosen to provide a large enough temporal separation between spikes from high intensity pixels and spikes from low intensity pixels. We noticed that accuracy suffered if this temporal separation was decreased below the synaptic time constant, i.e., below a time unit of 1, while increasing temporal separation further did not have an appreciable effect on accuracy. All times are normalized with respect to the synaptic time constant [see (2)]. We investigated two feedforward network topologies with fully connected hidden layers: the first network has one hidden layer of 800 neurons (the 784-800-10 network), and the second has two hidden layers of 400 neurons each (the 784-400-400-10 network). We found that accuracy is slightly improved if we use an extra reference neuron that always spikes at time 0 and projects through trainable weights to all neurons in the network. We ran 100 epochs of training with an exponentially decaying learning rate. We tried different learning rates and fastest convergence was obtained when learning rate starts at 0.01 in epoch 1 and ends at 0.0001 in epoch 100. We used a mini-batch size of 10. The weight sum cost coefficient [K in (10)] is 100. The L2 regularization coefficient is 0.001. The maximum allowed row-normalized Frobenius norm of the gradient of a weight matrix is 10.

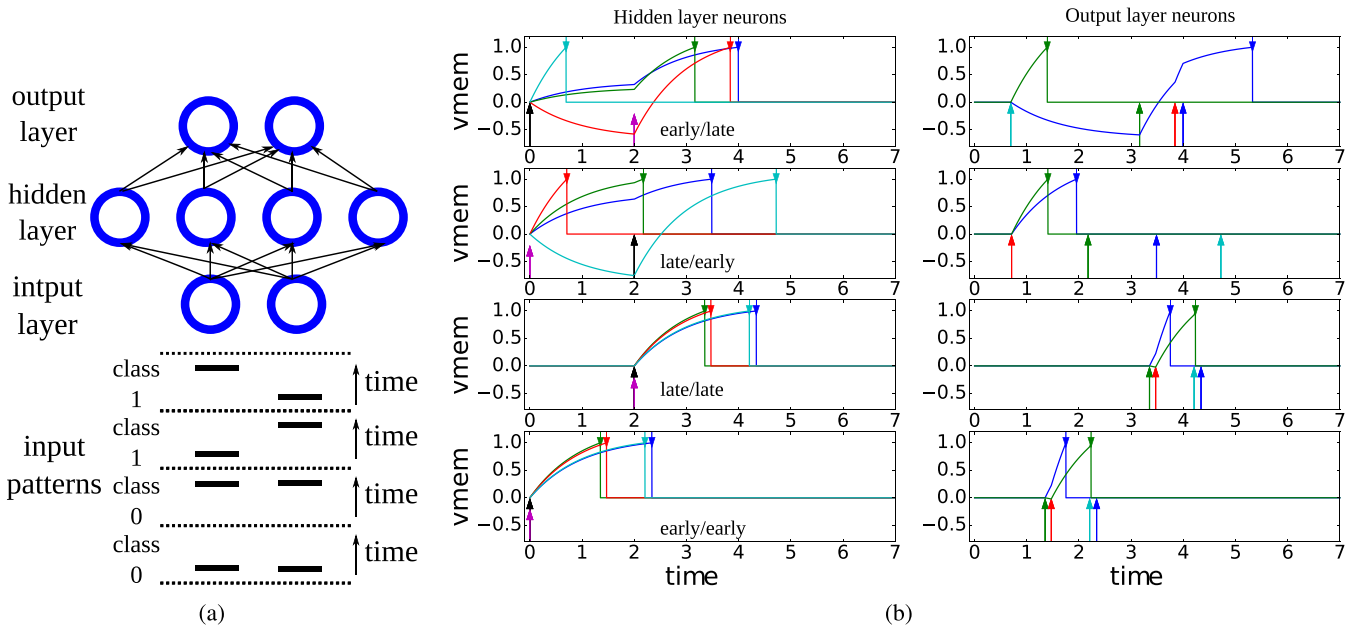

Fig. 2. (a) Network implementing the XOR task using one hidden layer. There are four input patterns divided into two classes. (b) Post-training simulation results of the network in (b) for the four input patterns, one pattern per row. Membrane potential of the four hidden layer neurons (left) and membrane potential of the two output layer neurons (right). Bottom arrows indicate input spikes to layer and top arrows indicate output spikes. The output spikes of the hidden layer are the input spikes of the output layer. The input pattern is indicated by the text in the left plots, and also by the pattern of input spikes.

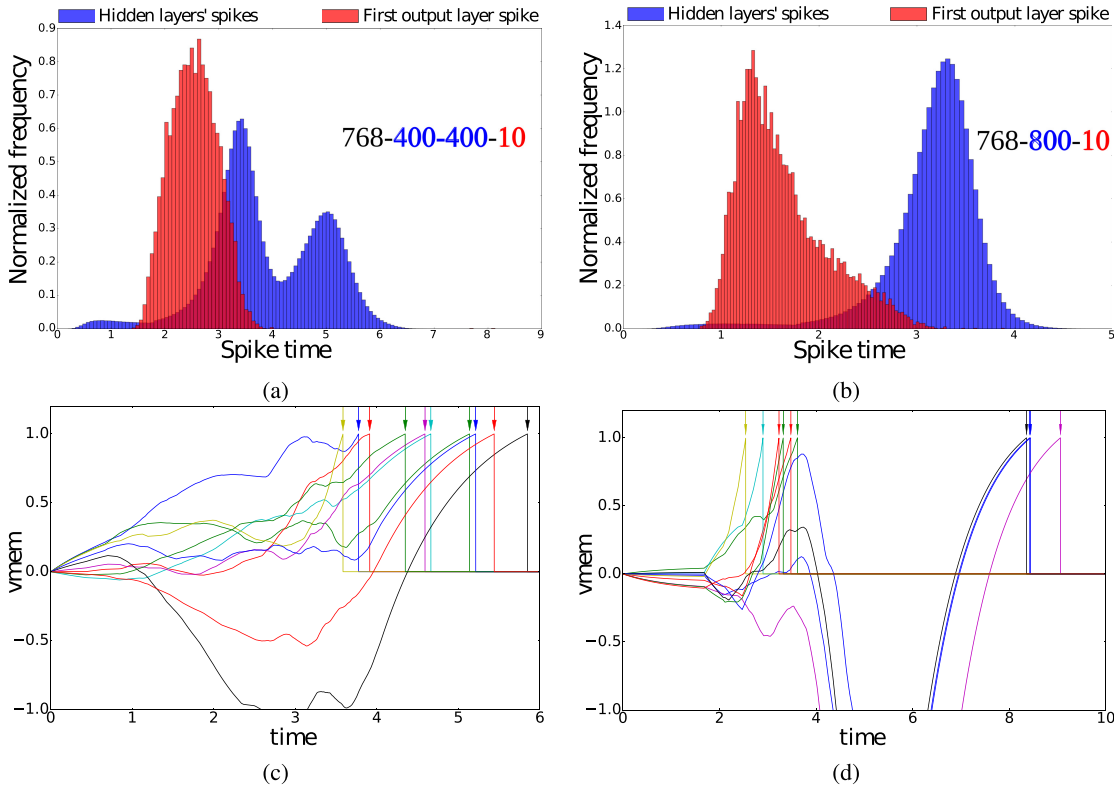

Fig. 3. (a) and (b) Histograms of spike times in the hidden layers and of the time of the first output layer spike across the 10 000 test set images for two network topologies. Both networks generate an output spike (i.e., select a class) before most of the hidden layer neurons have managed to spike. (c) Evolution of the membrane potentials of ten neurons in the second hidden layer of the 768-400-400-10 network in response to a sample image. Top arrows indicate the spike (threshold crossing) times. (d) Evolution of the membrane potentials of the ten output neurons in the same network and for the same input image as in (c). The earliest output neuron spikes (network selects a class) at time 2.5, i.e., before any of the ten hidden neurons shown in (c) have spiked.

Each of the two network topologies was trained twice, once with non-noisy input spike times and the other with noise-corrupted input spike times. In the noisy input case, noise delays each spike with the absolute value of a random quantity drawn from a zero mean, unity variance Gaussian distribution. Noise was only used during training. Table I shows the performance results for the two networks after the noisy and nonnoisy training regimes.

TABLE I
PERFORMANCE RESULTS FOR THE PERMUTATION-INVARIANT MNIST TASK

Network	Training error	Test error
784-800-10 (non-noisy training input)	0.013%	2.8%
784-800-10 (noisy training input)	0.005%	2.45%
784-400-400-10 (non-noisy training input)	0.031%	3.08%
784-400-400-10 (noisy training input)	0.255%	2.86%

The small errors on the training set indicate the networks have enough representational power to solve this task, as well as being effectively trainable. The networks overfit the training set as indicated by the significantly higher test set errors. Noisy training input helps in regularizing the networks as it reduces test set error but further regularization is still needed. We experimented with dropout [20] where we randomly removed neurons from the network during training. However, dropout does not seem to be a suitable technique in our networks as it reduces the number of spikes received by the neurons, which would often prevent them from spiking. Effective techniques are still needed to combat overfitting and allow better generalization in the proposed networks.

Fig. 3(a) and (b) shows the distribution of spike times in the hidden layers and the distribution of the times of the earliest output layer spike in the two networks. The latter is the times at which the networks made a decision for the 10 000 test examples. Both networks were first trained using noisy input. For both topologies, the network makes a decision after only a small fraction of the hidden layer neurons have spiked. For the 784-800-10 topology, an output neuron spikes (a class is selected) after only 3.0% of the hidden neurons have spiked (on average across the 10 000 test set images), while for the 784-400-400-10 topology, this number is 9.4%. The network is thus able to make very rapid decisions about the input class, after approximately 1–3 synaptic time constants from stimulus onset, based only on the spikes of a small subset of the hidden neurons. This is illustrated in Fig. 3(c) and (d) which shows the membrane potentials of ten hidden neurons and ten output neurons. The spikes of the ten hidden neurons do not factor into the network decision in this case as they all spike after the earliest output spike, i.e., after the network has already selected a class.

Fig. 4 shows the tuning properties of 30 randomly selected hidden layer neurons in the 784-800-10 network. We consider a hidden neuron to be tuned to a particular input class if it contributes to the classification of that class, i.e., if it spikes before the output layer spikes for that class. As shown in Fig. 4, neurons are typically broadly tuned and contribute to the classification of many classes.

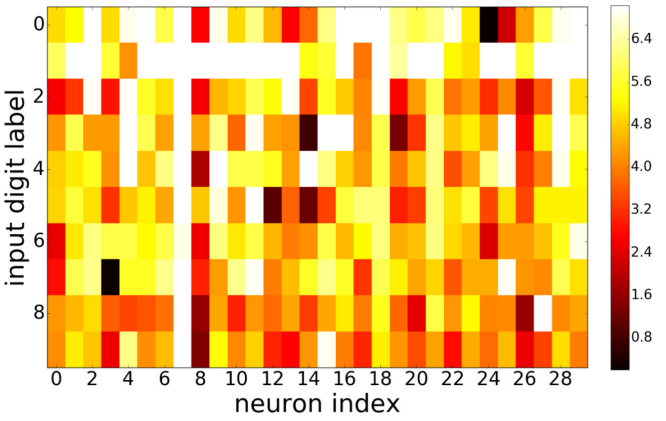

Fig. 4. Selectivity of 30 randomly selected hidden neurons in the 784-800-10 network to the 10 MNIST input classes. The negative log probability for each of the 30 neurons to spike before the output layer spikes for each of the ten input classes is shown. This is the negative log probability that a neuron participates in the classification of a particular class. Probability was obtained from the network's response to the 10 000 test digits. Some neurons are highly selective. For example, neuron 3 is highly selective to the '7' digits while most of the neurons are more broadly tuned. Some rare neurons are mostly silent such as neuron 7, yet all neurons contribute to the classification of at least one of the 10 000 test patterns.

V. DISCUSSION

We presented a form of spiking neural networks that can be effectively trained using gradient descent techniques. Using a temporal spike code, many difficulties involved in training spiking networks such as the discontinuous spike generation mechanism and the discrete nature of spike counts are avoided. The network input–output relation is piecewise linear after a transformation of the time variable. As the input spike times change, the causal input sets of the neurons change, which in turn changes the form of the linear input–output relation (Fig. 1). This is analogous to the behavior of networks using ReLUs [16] where changes in the input change the set of ReLUs producing nonzero output, thus changing the linear transformation implemented by the network. Piecewise linear transformations can approximate any nonlinear transformation to arbitrary accuracy [21]. ANNs based on ReLU networks are currently setting the state of the art in various machine vision tasks [22], [23]. As far as we know, this is the first time spiking networks have been shown to effectively implement a piecewise linear transformation from input to output spike times.

We used standard stochastic gradient descent (SGD) during training. While the second-order methods [24] could be used in principle, they are more computationally demanding than the first-order methods like SGD. Furthermore, by augmenting the first-order gradient information with various pieces of information about the gradient history [25], the performance gap between the first and the second order methods can be eliminated in many cases [26].

Recordings from higher visual areas in the brain indicate these areas encode information about abstract features of visual stimuli as early as 125 ms after stimulus onset [27]. This is consistent with behavioral data showing response times in the order of a few hundred milliseconds in visual discrimination tasks [28]. Given the typical firing rate of cortical neurons and delays across synaptic stages from the retina to higher visual areas, this indicates rapid visual processing is mostly a feedforward process where neurons get to spike at most once [29]. The presented networks follow a similar processing scheme and could thus be used as a trainable model to investigate the accuracy-response latency tradeoff in feedforward spiking networks. Output latency can be reduced using a penalty term in the cost function that grows with the output spikes latency. Scaling this penalty term controls

the tradeoff between minimizing latency and minimizing error during training. We used nonleaky integrate and fire neurons in our networks in order to obtain a closed form analytical expression relating input and output spike times. Biological neurons, on the other hand, have various leak mechanisms, allowing them to forget past subthreshold activity. A mechanism similar to leak-induced forgeting also occurs in our networks: information about the timing of incoming spikes is lost when the synaptic current for these spikes has decayed; the membrane potential indeed changes to a new value, but this value is independent of the ordering of past input spikes. Effective discrimination between different input temporal patterns can thus only happen when input spikes are within a few synaptic time constants of each other, which is also the case for leaky neurons.

The performance of our network on the MNIST task falls short of the state of the art. Feedforward fully connected ANNs achieve error rates between 0.9% and 2% [20] on the MNIST task. ANNs, however, are evaluated layer by layer and cannot be trained to produce a classification decision as soon as possible like the networks we describe in this paper. The training pressure to produce the classification result as soon as possible forces the described networks to ignore the majority of hidden neurons' activity by producing an output spike before most of the hidden neurons have spiked, which could explain the reduced accuracy. Rate-based spiking networks with a similar architecture to ours could achieve error rates as low as 1.3% [30]. They, however, make use of thousands of spikes that are integrated over time in order to yield an accurate classification result.

We considered the case where each neuron in the network is allowed to spike once. The training scheme can be extended to the case where each neuron spikes multiple times. The time of later spikes can be differentiably related to the times of all causal input spikes [see (7)]. This opens up interesting possibilities for using the presented networks in recurrent configurations to process continuous input event streams. The backpropagation scheme we outlined in Section III would then be analogous to the backpropagation through time technique [31] used to train artificial recurrent network.

The presented networks enable very rapid classification of input patterns. As shown in Fig. 3, the network selects a class before the majority of hidden layer neurons have spiked. This is expected as the only way the network can implement nonlinear transformations (in the z-domain) is by changing the causal set of input spikes for each neuron, i.e., by making a neuron spike before a subset of its input neurons has spiked. This unique form of nonlinearity not only results in rapid processing, but it enables the efficient implementation of these networks on neuromorphic hardware since processing can stop as soon as an output spike is generated. In the 784-800-10 MNIST network, for example, the network classifies an input after only 25 spikes from the hidden layers (on average). Thus, only a small fraction of hidden layer spikes needs to be dispatched and processed.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[2] D. Rumelhart and J. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Foundations*, vol. 1. Cambridge, MA, USA: MIT Press, 1986.

[3] M. Minsky, "Steps toward artificial intelligence," *Proc. IRE*, vol. 49, no. 1, pp. 8–30, Jan. 1961.

[4] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2654–2662.

[5] R.-M. Memmesheimer, R. Rubin, B. Ölveczky, and H. Sompolinsky, "Learning precisely timed spikes," *Neuron*, vol. 82, no. 4, pp. 925–938, 2014.

[6] 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.

[7] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning," *Neural Comput.*, vol. 18, no. 6, pp. 1309–1339, 2006.

[8] B. Gardner, I. Sporea, and A. Grüning, "Learning spatiotemporally encoded pattern transformations in structured spiking neural networks," *Neural Comput.*, vol. 27, no. 12, pp. 2548–2586, 2015.

[9] 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.

[10] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers Neuroscience*, vol. 7, p. 178, Oct. 2013. [Online]. Available: http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2013.00178/abstract

[11] 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)*, 2015, pp. 1–8.

[12] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *Int. J. Comput. Vis.*, vol. 113, no. 1, pp. 54–66, 2015.

[13] N. Qiao *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers Neurosci.*, vol. 9, p. 141, Apr. 2015. [Online]. Available: http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2015.00141/abstract

[14] D. Neil and S.-C. Liu, "Minitaur, an event-driven FPGA-based spiking network accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, p. 2621–2628, Dec. 2014.

[15] 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.

[16] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.

[17] 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.

[18] F. Bastien *et al.* (Nov. 2012). "Theano: new features and speed improvements." [Online]. Available: https://arxiv.org/abs/1211.5590

[19] J. Bergstra *et al.*, "Theano: A CPU and GPU math compiler in Python," in *Proc. Python Sci. Comput. Conf. (SciPy)*, vol. 4. Austin, TX, USA, 2010, p. 3.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proc. Int. Conf. Mach. Learn.*, vol. 3, 2013, pp. 1319–1327.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[23] G. Huang, Z. Liu, K. Weinberger, and L. van der Maaten. (Aug. 2016). "Densely connected convolutional networks." [Online]. Available: https://arxiv.org/abs/1608.06993

[24] J. Martens, "Deep learning via hessian-free optimization," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 735–742.

[25] D. Kingma and J. Ba. (Dec. 2014). "Adam: A method for stochastic optimization." [Online]. Available: https://arxiv.org/abs/1412.6980

[26] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 3, 2013, pp. 1139–1147.

[27] C. Hung, G. Kreiman, T. Poggio, and J. DiCarlo, "Fast readout of object identity from macaque inferior temporal cortex," *Science*, vol. 310, no. 5749, pp. 863–866, 2005.
[28] M. Fabre-Thorpe, G. Richard, and S. Thorpe, "Rapid categorization of natural images by rhesus monkeys," *Neuroreport*, vol. 9, no. 2, pp. 303–308, 1998.
[29] S. J. Thorpe, A. Delorme, and R. VanRullen, "Spike-based strategies for rapid processing," *Neural Netw.*, vol. 14, nos. 6–7, pp. 715–725, 2001.
[30] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers Neurosci.*, vol. 10, p. 508, Nov. 2016.
[31] M. Mozer, "A focused back-propagation algorithm for temporal pattern recognition," *Complex Syst.*, vol. 3, no. 4, pp. 349–381, 1989.

Hesham Mostafa received the master's degree in electrical engineering from the Technical University of Munich, Munich, Germany, in 2010, and the Ph.D. degree in neuroinformatics from the Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zürich, Switzerland, in 2016.

He currently holds a post-doctoral position at the Integrated Systems Neuro-Engineering Laboratory at the Institute of Neural Computation, University of California, San Diego, CA, USA. His current research interests include combining ideas from machine learning and computational neuroscience for developing biologically inspired and hardware-efficient learning and optimization algorithms, and physically implementing these algorithms using CMOS and novel device technologies.