Efficient Biologically-Plausible Training of Spiking Neural Networks with Precise Timing

Richard Boone
richardboone@ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

Wenrui Zhang
wenruizhang@ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

Peng Li
lip@ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

ABSTRACT

Spiking Neural networks (SNNs) are well suited to implementation on energy-efficient neuromorphic processors and to imitation of biological systems. However, promising backpropagation methods have been developed for SNNs, they tend to be either not biologically plausible or to be computationally complex. In this paper we present two biologically plausible alternatives to backpropagation while retaining high temporal precision for SNN training. We show distinct tradeoffs between complexity, accuracy, and amenability to deployment on a neuromorphic processor of several training methods. Finally, We suggest exploration of biologically plausible methods to enable low complexity training on resource-constrained neuromorphic hardware.

KEYWORDS

Neural networks, spiking neural networks, training, biological plausibility

ACM Reference Format:
Richard Boone, Wenrui Zhang, and Peng Li. 2021. Efficient Biologically-Plausible Training of Spiking Neural Networks with Precise Timing. In *International Conference on Neuromorphic Systems 2021 (ICONS 2021), July 27–29, 2021, Knoxville, TN, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3477145.3477147

1 INTRODUCTION

In recent years, Spiking Neural Networks (SNNs) have gathered interest as a brain-inspired computing model. Spike-based operations allow for three significant advantages over traditional neural networks. First, SNNs allow for simple encoding of temporal information, which in conventional non-spiking artificial neural networks (ANNs) would require complex internal structures such as LSTM [11]. Second, SNNs are well adapted to work on low-power neuromorphic hardware as has recently been demonstrated by IBM's TrueNorth [1] and Intel's Loihi [6]. Finally, SNNs provide a much more accurate model for modeling biological neurons, giving them more accurate applications in neuroscience.

Primarily due to the prevalence of backpropagation in traditional neural networks, backpropagation-based method have been the main focus of many aspects of learning research in spiking neural nets. In recent years many methods have been developed for backpropagation in spiking neural networks [4, 13, 17–19, 22, 24–27].

While backpropagation algorithms have been shown to be very effective in some situations, learning using backpropagation is not biologically plausible in that it is very dissimilar to any known methods present in the brain. Additionally, learning using backpropagation is usually much more complex than alternative learning methods. Alternative learning methods such as Direct Feedback-Alignment (DFA) [20], Kollen-Pollack (KP) [14], and Direct Kollen-Pollack (DKP) [5] have been developed to simplify the learning process or to improve biological plausiblility. While these methods are simpler than any known biological processes, their learning methods can be plausibly mapped to known brain structures. However, most of these methods have been used primarily in traditional neural networks, and not in spiking neural networks.

Backpropagation-based methods in SNNs have historically been limited by the computation of local gradients due to the discontinuity of spike activations and spike timing. Recent work has greatly simplified the effective training of deep SNNs by improving localized gradient calculation by accounting for the discontinuity of spike timings [27]. The resulting TSSL-BP method has allowed for efficient learning in short time windows for some networks and improved learning even when long time windows are included [27].

While this method shows great promise in the training of complex neural networks, it is not biologically plausible and it gets increasingly complex in networks of long time periods. While other methods such as pruning have been proposed to limit the complexity of networks, such as pruning, these methods do not consider temporal optimizations[3]. We propose two optimizations to the gradient computation presented in [27] and propose two methods by which we can utilize this method for biologically plausible learning with high temporal precision, especially in neuromorphic environments.

In this work we propose two new methods, TSSL-DFA, an extension of [20] and TSSL-KP, an extension of [14], for training spiking neural nets. Using the local gradients demonstrated by [27] we show that both methods can near the accuracy of TSSL-BP under certain situations while maintaining biological plausibility, and in the case of TSSL-DFA greatly reducing the complexity of the required feedback algorithms. We assess the complexity of these algorithms to show their usefulness under neuromorphic hardware. Additionally, we propose a method by which we reduce the timing complexity of [27] for more efficient training of complex neuromorphic systems.

ICONS 2021, July 27–29, 2021, Knoxville, TN, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8691-3/21/07.
https://doi.org/10.1145/3477145.3477147

2 BACKGROUND

2.1 History of backpropagation and learning methods in SNNs

One method for training SNNs has historically been to train an ANN, then adapt the weights and activation functions as necessary to work on SNNs[7, 8, 12, 21]. While this method can exploit the more effective training of an ANN, it has shown significant limitations due to the conversion process creating significant errors due to forward propagating approximation errors and due to an inability to effectively train SNNs for temporal data, a task for which they are specialized.

One of the earliest methods for applying backpropagation-based learning to SNNs was spikeprop [4]. Spikeprop showed some improvements over alternative methods at the time, but was limited in that each neuron was only allowed to fire once per task, severely limiting the applications available, especially for time-variant data. Alternative methods similar to spikeprop have been explored. However, none of these methods have demonstrated significant performance on learning tasks.

Following spikeprop, a number of alternative methods for learning have been developed that historically reached competitive levels of performance including [13], [24], [17], and [22]. However, all of these used some sort of surrogate method for evaluating the gradient of a given neuron at a given time. While these methods have been shown to be effective in some situations, they are all outperformed by [27] in either pure performance, training efficiency, or both.

2.2 Direct Feedback-Alignment

Direct Feedback-Alignment (DFA) is a learning method originally implemented in [20] for directly pathing feedback from the error function of a network to each layer of the network. DFA gets feedback to hidden layers by multiplying the output of the error function on the final layer with number of neurons o with a randomized matrix of size $o \times m$ where m is the number of output neurons of a given layer. This returns a vector of size m which is then used as the error at the output of the hidden layer. DFA then computes the weight updates identically to standard backpropagation methods. Because the feedback path present in DFA is entirely separated from the feedforward pathway, it is more biologically plausible. Additionally, DFA provides two distinct improvements over traditional backpropagation. First, the feedback pathway can be entirely parallelized allowing for increased training speed as the network does not have to delay for a full backward pass, as in BP. Second, the overall complexity of the feedback algorithm is significantly reduced. As DFA is both biologically plausible and greatly reduced in complexity as compared to BP, it presents significant advantages for implementation in spiking neural networks. Because SNNs offer significant advantages when implemented on neuromorphic hardware and because they offer one of the best methods for simulation of biological neurons they benefit much more from these advantages than traditional ANNs.

2.3 Kollen-Pollack

Kollen-Pollack (KP) is a feedback method originally developed in [14] as a biologically plausible alternative to backpropagation. KP works by passing error terms backward through a feedback path parallel and identical to the existing network, but propagating backward instead of forward. Because the backward network uses different weights than the forward network and the feedback pathway is entirely separated, KP is considered a biologically plausible method of feedback. Additionally, [23] showed that KP can match the capabilities of BP in many situations on ANNs, allowing for biological plausibility without any significant loss of accuracy. While KP does not show the significant complexity reductions of DFA, its biological plausibility and significant accuracy make it similarly ideal for implementation on spiking neural networks.

2.4 Model for spiking neurons

We use a leaky integrate and fire neuronal model as shown in [9]. An input spike train from presynaptic neuron j is expressed by: $s_j(t) = \sum_{t_j^{(f)}} \delta(t - t_j^{(f)})$, where $t_j^{(f)}$ represents a particular presynaptic firing time. The presynaptic spikes are converted into a postsynaptic current (PSC) using equation (1) where $u_i(t)$ represents the neuronal membrane voltage at a given time point t for a postsynaptic neuron i that is connected with neuron j.

$$\tau_m \frac{du_i(t)}{dt} = -u_i(t) + R \sum_j w_{ij} a_j(t) + \eta_i(t) \tag{1}$$

Here R and τ_m represent the leaky resistance and the time constant of the neuron respectively, and w_{ij} represents the synaptic weight of presynaptic neuron j when applied to postsynaptic neuron i. $a_j(t)$ similarly represents the PSC of presynaptic neuron j as defined in equations (2) and (3). Finally, $\eta_i(t)$ represents the reset function of the network as defined in equation (2).

$$a_j(t) = (\epsilon * s_j)(t), \qquad \eta_i(t) = (\nu * s_i)(t) \tag{2}$$

Here $\epsilon(\cdot)$ is the synaptic response kernel. We use a first order synaptic response kernel defined in (3). $\nu(\cdot)$ is the reset kernel by which the neuron resets after a spike. The reset kernel reduces the membrane potential $u_i[t]$ to 0.

$$\tau_s \frac{a_j(t)}{dt} = -a_j(t) + s_j(t) \tag{3}$$

Because all simulations here are performed in discrete time, we discretize (1) into (4). Two significant simplifications are performed here. First, R and τ_m are consistent across all neurons, and so can be treated as part of the synaptic weight w_{ij}. This allows the removal of such terms from the synaptic potential increase computation. Second, we use a fixed step size 1 ms which removed any ΔT term.

$$u_i[t] = (1 - \frac{1}{\tau_m}) u_i[t-1] + \sum_j w_{ij} a_j[t] + \eta_i[t] \tag{4}$$

2.5 TSSL-BP

As illustrated in Fig. 1, Temporal Spike Sequence Learning via Backpropagation (TSSL-BP) [27] is a backpropagation method specific to spiking neural networks. TSSL splits the gradient at the neuron into two disparate sections.

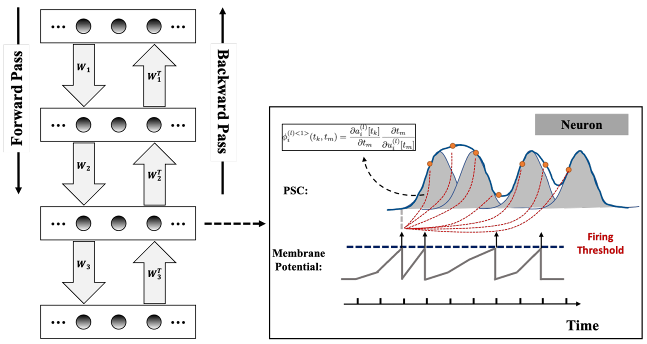

Figure 1: Visualization of TSSL-BP.

First is the inter-neuron dependency. Inter-neuron dependency is the dependency of a postsynaptic neuron on the spikes of a presynaptic neuron. Prior to backward propagation, this means that inter-neuron dependency can be calculated as the dependency of a postsynaptic neuron on the set of input currents to the neuron. The core of the computation involved is to capture the dependency of the postsynaptic current (PSC) $a_i^{(l)}$ generated by each presynaptic neuron i in layer l at time t_k on the membrane potential of the neuron at one of its firing times t_m. Note that each PSC is part of the input currents to the postsynaptic neurons in layer $l+1$, and hence a contributor to inter-neuron dependency. This dependency is represented in equation (5) where $a_i^{(l)}[t_k]$ is the PSC at time t_k, t_m is the time of a presynaptic spike, and $u_i^{(l)}$ is the membrane potential of the presynaptic neuron i. For the purpose of simplifying computation, this can be split into the two equations shown in (6).

$$\phi_i^{(l)<1>}(t_k, t_m) = \frac{\partial a_i^{(l)}[t_k]}{\partial t_m} \frac{\partial t_m}{\partial u_i^{(l)}[t_m]} \tag{5}$$

$$\frac{\partial a_i^{(l)}[t_k]}{\partial t_m} = \frac{\partial(\epsilon * s_i^{(l)}[t_m])[t_k]}{\partial t_m}, \qquad \frac{\partial t_m}{\partial u_i^{(l)}[t_m]} = \frac{-1}{\frac{\partial u_i^{(l)}[t_m]}{\partial t_m}} \tag{6}$$

The second portion of TSSL is intra-neuron dependency. Intra-neuron dependency is the dependency of a neuron on its own post-synaptic spikes. Because a postsynaptic spike resets the synaptic potential of a neuron, the chances of any potential future spikes are reduced by the existence of a postsynaptic spike. The intra-neuron dependency at a time t_k is represented by (7) where ν represents the reset kernel, and t_m and t_p are the times of previous spikes with no spikes between them.

$$\phi_i^{(l)<2>}(t_k, t_m) = \phi_i^{(l)}(t_k, t_p) \frac{\partial(\nu * s_i^{(l)}[t_m])[t_p]}{\partial t_m} \frac{\partial t_m}{\partial u_i^{(l)}[t_m]} \tag{7}$$

3 PROPOSED BIOLOGICALLY PLAUSIBLE TSSL-DFA AND RT-TSSL-DFA METHODS

The first of our new algorithms consists of adapting DFA to a spiking neural network context. While DFA has been demonstrated as a biologically plausible method for training conventional ANNs [20] and employed for spike-train level on-chip training of SNNs [16], it has not yet been successfully adapted to training of SNNs with high temporal precision. We propose two significant changes to the original DFA algorithm. First, we adapt DFA to an SNN context, feeding error from the output layer back to each spiking hidden layer as illustrated in Fig. 2. For the local gradient, we use the Temporal Spike Sequence Learning via Backpropagation (TSSL-BP) method demonstrated in [27]. As with traditional DFA, this allows the learning function to use both the output error of the network and the local gradient of a given layer to effectively generate weight updates.

Under TSSL-BP, the loss $\delta^{(l)}[t_m]$ at a given layer is given as (8) where l is the current layer. Under TSSL-DFA, we edit this to directly apply feedback from the output error function to the hidden layer l using a random matrix M. This results in equation (9) where $E[t_k]$ is the error at the output layer at time k.

$$\delta^{(l)}[t_m] = (W^{(l+1)})^T \sum_{k=m}^{N_t} \frac{\partial a^{(l)}[t_k]}{\partial u^{(l)}[t_m]} \delta^{(l+1)}[t_k]. \tag{8}$$

$$\delta^{(l)}[t_m] = \sum_{k=m}^{N_t} \frac{\partial a^{(l)}[t_k]}{\partial u^{(l)}[t_m]} (M^{(l)} E[t_k]). \tag{9}$$

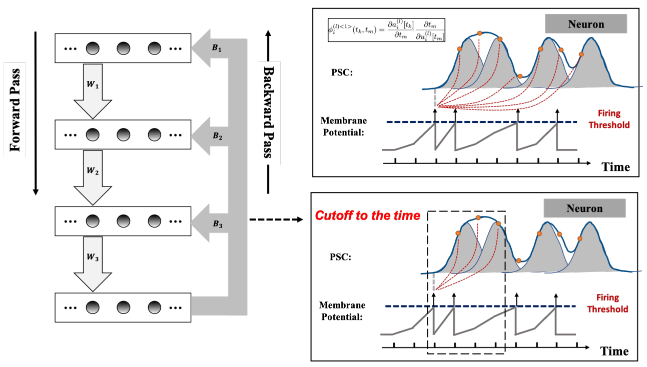

Figure 2: Proposed TSSL-DFA for SNN training.

The second significant change is a reduction in complexity of the original TSSL algorithm. Like many other BP methods, TSSL-BP scales with the square of time. It is not very efficient for datasets with a large time component. To combat this, we limit the amount of time backward TSSL considers when it is applied. We call this additional time reduction Reduced Time Temporal Spike Sequence Learning (RT-TSSL), and refer to its variants here as RT-TSSL-BP and RT-TSSL-DFA. This reduction in time reduces the complexity to scaling linearly with time as shown in the bottom-right subfigure of Fig. 2. When applying this change, the RT value is represented by N_r, and changes equation (9) to (10). Additionally, the intra-neuron effects present in TSSL-BP do not significantly contribute to the accuracy of the learning process and so have been removed in RT-TSSL-DFA. These reductions together leave us with a learning algorithm significantly less complex than TSSL-BP while also gaining the benefit of biological plausibility.

Table 1: Number of multiplies in one training iteration of one sample for two example layers

	Example convolution layer	Example FC layer
Number of layer inputs (n)	784	1440
Number of layer outputs (m)	1440	100
Network outputs (o)	10	10
Convolution dimension (p and q)	5	N/A
Outputs channels (r)	10	1
Stride (s)	2	N/A
Time (t)	5	5
Time reduction (l)	3	3
Forward pass	180000	720000
TSSL gradient calculations	93600	6500
RT-TSSL gradient calculations	43200	3000
Backward propagation	180000	720000
DFA propagation	72000	5000
Weight updates	180000	720000
TSSL-BP and TSSL-KP	633600	2166500
RT-TSSL-BP and RT-TSSL-KP	583200	2163000
RT-TSSL-DFA	475200	1448000
RT-TSSL-BP and RT-TSSL-KP percentage reduction	**7.95%**	**0.16%**
RT-TSSL-DFA percentage reduction	**25.00%**	**33.16%**

$$\delta^{(l)}[t_m] = \sum_{k=m}^{min(m+N_r,N_t)} \frac{\partial a^{(l)}[t_k]}{\partial u^{(l)}[t_m]}(M^{(l)}E[t_k]). \qquad (10)$$

4 PROPOSED BIOLOGICALLY PLAUSIBLE TSSL-KP AND RT-TSSL-KP METHODS

The second of our new algorithms involves adapting KP to a spiking neural network context as shown in Fig. 3. Just as with the original KP algorithm, we apply localized gradients on a parallel path backward. These localized gradients come in the form of the TSSL algorithm just as with TSSL-DFA and are simultaneously used both for localized weight updates and updates to the backward propagation of gradients. When applying this algorithm with the TSSL gradients, we edit equation (8) to (11) where B^{l+1} is a matrix of shape $(W^{(l+1)})^T$. Just as with the original KP algorithm, the weight updates generated by $\delta^{(l)}[t_m]$ are applied to both the B and W matrices.

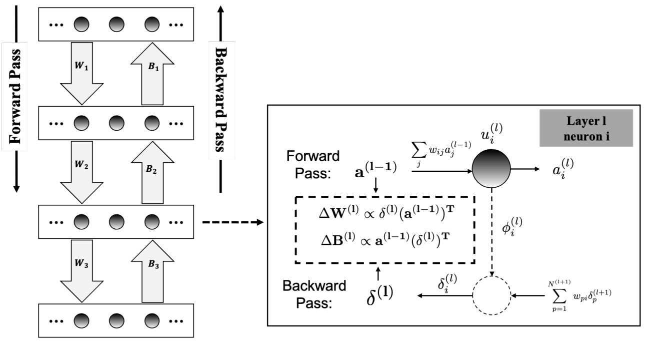
Forward Pass
Backward Pass
W_1
B_1
W_2
B_2
W_3
B_3
Layer l neuron i
Forward Pass:
$\mathbf{a}^{(l-1)}$
$\sum_j w_{ij} a_j^{(l-1)}$
$u_i^{(l)}$
$a_i^{(l)}$
$\Delta \mathbf{W}^{(l)} \propto \delta^{(l)}(\mathbf{a}^{(l-1)})^{\mathbf{T}}$
$\Delta \mathbf{B}^{(l)} \propto \mathbf{a}^{(l-1)}(\delta^{(l)})^{\mathbf{T}}$
$\phi_i^{(l)}$
Backward Pass:
$\delta^{(l)}$
$\delta_i^{(l)}$
$\sum_{p=1}^{N^{(l+1)}} w_{pi}\delta_p^{(l+1)}$

Figure 3: The proposed TSSL-KP for training SNNs.

$$\delta^{(l)}[t_m] = (B^{(l+1)}) \sum_{k=m}^{N_t} \frac{\partial a^{(l)}[t_k]}{\partial u^{(l)}[t_m]} \delta^{(l+1)}[t_k]. \qquad (11)$$

Just as with TSSL-DFA we apply a time limit on TSSL-KP and call the related algorithm RT-TSSL-KP. In a similar manner this simplifies (11) to (12). This again reduces the complexity from scaling with the square of time to scaling linearly with time. Similarly to DFA, KP allows biological plausibility as a learning algorithm due to its inherently separated feedback pathway.

$$\delta^{(l)}[t_m] = (B^{(l+1)}) \sum_{k=m}^{min(m+N_r,N_t)} \frac{\partial a^{(l)}[t_k]}{\partial u^{(l)}[t_m]} \delta^{(l+1)}[t_k]. \qquad (12)$$

5 COMPLEXITY ANALYSIS

Here we use number of multiplies as a surrogate for hardware complexity. While this is not a perfect metric, multiplication operations would be both the most prevalent and most complex operation required in most neuromorphic processors, and so provide a good analogue for hardware complexity in addition to providing a similarly good estimate for the time complexity of training in simulated environments. For the purposes of this paper, complexity calculations will be limited to fully-connected and convolutional layers as no more complex layers were used in the results and layers of simpler complexity, such as dropout, do not significantly contribute to training complexity. Lastly, all calculations are for a single iteration of a single sample during training.

5.1 Forward Propagation

Because our changes only involve the feedback methods present, forward propagation is identical for all methods demonstrated in this paper. Forward propagation can be split into two significant

Table 2: Full network complexity (number of multiplies)

	MNIST Network	DVSG (time limit 5)	DVSG (time limit 20)	DVSG (time limit 40)
TSSL-BP and TSSL-KP	2862250	1044339300	1044339300	1044339300
RT-TSSL-BP and RT-TSSL-KP	2784750	950826900	955533900	961809900
RT-TSSL-DFA	1957250	636290400	640997400	647273400
RT-TSSL-BP and RT-TSSL-KP percentage reduction	**2.71%**	**8.95%**	**8.50%**	**7.90%**
DFA percentage reduction	**29.72%**	**33.08%**	**32.92%**	**32.70%**

parts in a spiking neural network. First, weights are used to apply forward propagation in a method roughly identical to that in an ANN. For any 2D convolution layer with a known input shape, convolution size p by q, stride s, padding p, and number of output channels r we have a deterministic number of total outputs m. Complexity can be defined either by the input shape and convolution parameters p,q,s,p, and r, or by the output size m and by the convolution size p and q. For the sake of simplicity in equations, we use the number of outputs neurons m as a surrogate for the input size and shape, stride, padding, and number of output channels. Then, in a 2D convolution layer with convolution height p, convolution width q, m outputs, and t time points, we get a single convolution for each output in each time step, giving a total number of multiplies equal to $pqmt$. Similarly, in a fully-connected layer with n inputs, we get a total number of multiplies equal to nmt.

Second, each layer requires computation of the spike-level outputs of the layer. However, because of the nature of the synaptic kernel, spike-level outputs can be generated using only add and shift operations, making the complexity of the required hardware negligible when compared to the forward propagation.

This gives a forward propagation complexity equal to $O(pqmt)$ and $O(nmt)$ for convolution layers and fully-connected layers, respectively.

5.2 TSSL local gradient calculation

TSSL local gradient calculations account for the gradient between the output of the convolutional or linear portion of a network and the spiking output of a network. This means that the gradients are used regardless of the type of layer present. The TSSL calculations can be separated into three different parts. First, the intra-neuron calculation requires a multiplication across the output of a layer and across a doubled time dimension due to the relation of each spike to all previous spikes. This gives a multiplication complexity of $O(mt^2)$ following the previous variable conventions. Under the Reduce Time (RT) algorithms, the inter-neuron calculation is removed.

Second, the inter-neuron and intra-neuron are combined with the learning gradients from the rest of the network. This again involves a multiplication across both time dimensions resulting again in a regular complexity of $O(mt^2)$. When applying the TR algorithms, each timestep is only multiplied across a limited time l which results in a complexity of $O(mlt)$.

Finally, the application of the above calculations to the latter half of (6) requires three multiplications of complexity mt. This gives a total complexity of local gradient calculation for the complex and TR cases of $O(2mt^2 + 3mt)$ and $O(mtl + 3mt)$, respectively.

5.3 Backward propagation in BP and KP

When using a backpropagation-based algorithm, backpropagation takes a significant portion of the complexity. Because passing backward is simply the inverse of the forward pass, the complexity remains the same as the forward pass equations. This gives the backward complexity to be $O(pqmt)$ and $O(nmt)$ for convolution and fully-connected layers, respectively.

5.4 DFA backpropagation

DFA backpropagation is much simpler as it only requires a matrix multiplication of the error on the output with the layer-specific backward matrix. Similar to a fully-connected layer, DFA backward propagation gives a complexity of $O(omt)$ where o represents the total number of classification categories for the network.

5.5 Weight updates from the gradient at each layer

In all learning methods, a weight update must be generated from the error gradient at the output of each individual layer. For fully connected and convolutional layers these are $O(nmt)$ and $O(pqmt)$ respectively. Because these updates are local to each layer, this calculation is required for all feedback methods.

5.6 Overall Complexity of a Convolution Layer

Here we use the equations from the previous sections to produce three different complexity options for a convolutional layer. The most complex involves a backpropagation algorithm without TR. Combining equations of the previous sections we get a complexity equal to $O(pqmt + 2mt^2 + 3mt + pqmt + pqmt)$, which simplifies to $O(3pqmt + 2mt^2 + 3mt)$. For RT-TSSL-BP and RT-TSSL-KP we can reduce the complexity of the local TSSL gradients, giving a complexity of $O(3pqmt + mtl + 3mt)$. Finally, for RT-TSSL-DFA we remove the backpropagation element of complexity $O(pqmt)$ and replace it with the direct feedback of complexity $O(omt)$. This gives a total RT-TSSL-DFA complexity of $O(2pqmt+omt+mtl+3mt)$. The complexity of one example convolution layer is shown in table 1.

5.7 Overall Complexity of a Fully Connected Layer

For FC layers we again separate complexity into three categories. For BP-based learning algorithms without RT, we get a layer level complexity of $O(nmt+2mt+2mt^2+3mt+nmt+nmt)$. This simplifies to $O(3nmt+5mt+2mt^2)$. When we apply RT to a BP-based algorithm, we remove the $2mt^2$ term and insert an mtl term to get $O(3nmt + 5mt + mtl)$ complexity for an FC layer with a BP-based algorithm and RT. Finally, for RT-TSSL-DFA, we remove the backpropagation feedback term of complexity $O(nmt)$ and replace it with the DFA

Table 3: MNIST Results over 10 trials

	Standard			Reduced Time (l=3)		
	TSSL-BP	TSSL-KP	TSSL-DFA	RT-TSSL-BP	RT-TSSL-KP	RT-TSSL-DFA
Average	98.79	98.85	97.27	98.78	98.85	97.59
Max	98.95	98.96	97.67	98.87	98.92	97.99
Min	98.63	98.77	96.61	98.67	98.8	96.97

All networks: 10C5S2->FC100->FC10

10C5S2: Convolutional layer with 10 $5x5$ filters, stride 2, FC100: Fully Connected layer with 100 outputs

feedback term of complexity $O(omt)$. This gives a total complexity for RT-TSSL-DFA in a fully connected layer of $O(2nmt + omt + 5mt + mtl)$. The complexity of one example convolution layer is shown in table 1.

6 RESULTS

We simulated our algorithms on two separate networks among two datasets that are commonly used in the neuromorphic computing community.

6.1 Results on MNIST dataset

On MNIST [15] we use a single, three-layer network. The layers are: a convolution layer with stride 2, convolution size 5x5, and 10 output channels, a fully connected layer with 100 output neurons, and a fully connected layer with 10 output neurons. We compare the results using TSSL-BP, TSSL-KP, TSSL-DFA, as well as our Reduced Time versions, RT-TSSL-BP, RT-TSSL-KP, and RT-TSSL-DFA. We run experiments of each algorithm 10 times and display the average, minimum, and maximum accuracies achieved. We compare these algorithms on an identical network size and show the tradeoffs of each individual algorithm. In Table 3 we show two significant findings. First, TSSL-DFA, while reducing complexity by roughly 30% as shown in table 2, still nears the performance of TSSL-BP and TSSL-KP. Second, RT does not cause significant drops in accuracy in TSSL-BP and TSSL-KP, while actually increasing accuracy in the case of TSSL-DFA.

6.2 Results on DVS Gesture dataset

The Dynamic Vision Sensor Gesture dataset [2] is a set of recordings of 29 different individuals performing a set of specific gestures. Because the dataset uses a dynamic vision sensor, which only transmits data when pixels change value and because the data comes over a period of time the dataset is well suited for testing spiking neural nets. Each trial involves a subject performing one of 11 different gestures under one of three lighting conditions over a period of 6 seconds. The task for this dataset is to classify the action demonstrated by the subject. For the purpose of this paper we apply two significant reductions to this dataset. First, we reduce the temporal frequency of the data from every 1 *us* in the original dataset to every 5 *ms*, and second we reduce the total time over which the action occurs by only using the first 1.5 seconds of each video. These two steps reduce the number of timesteps required to train the network, but lose some information and some temporal resolution. Additionally, the preprocessing steps we adopt in this work follow the ones in [22].

We use a single network size for all results, the same used in [10]. The network includes in order: One pooling layer of size 2, one fully connected layer with 512 output neurons, and one fully connected layer with 11 output neurons.

We show two significant changes from [10]. First, our localized gradient is based on the TSSL-BP method presented in [27] and includes the RT additions presented in this paper whereas the other authors used a BPTT-based method from [24]. Second, we add two additional feedback methods, TSSL-KP and TSSL-DFA as alternatives to regular backpropagation. We run each of the learning methods 5 times and display the average, minimum, and maximum accuracy achieved by each method in Table 4.

We note three significant results from this comparison. First, both TSSL-KP and TSSL-DFA, in addition to TSSL-BP, outperform STBP [10] on the same network size even without any time limitations. We believe that the improvements are due to the more effective use of local TSSL gradients.

Second, once again as shown in the MNIST dataset, RT-TSSL-DFA shows significant benefits under all shown levels of limited time while greatly reducing complexity. Using a time limit of 20 time steps provided a performance improvement of approximately 2% over the standard implementation of DFA while providing a very significant decrease in complexity. We also show that an increased time limit of 40 time steps provides 1.6% jump in accuracy over a time limit of 20 time steps, and a 3.6% increase over no time limit. While TSSL-DFA does show a noticeable performance drop over TSSL-BP, it does provide a tradeoff in reduced complexity by approximately 33% as shown in table 2.

Finally, we show in table 4 that both RT-TSSL-BP, and RT-TSSL-KP show minor increases in accuracy of less than 0.5% while also showing in table 2 that both algorithms show an 8.5% decrease in number of multiplies as compared to TSSL-BP and TSSL-KP respectively.

7 CONCLUSION

In this paper we present two new main ideas. First, we present two new biologically plauisible learning algorithms for SNNs, TSSL-DFA, and TSSL-KP. TSSL-DFA is an advancement to Direct Feedback Alignment (DFA), originally presented in [20]. Using the local gradients presented in [27] we apply DFA to spiking neural networks. This presents a biologically plausible alternative to TSSL-BP using direct feedback. TSSL-KP is an advancement to the Kollen-Pollack method (KP) originally presented in [14]. This presents a biologically plausible alternative to TSSL-BP using a separate but parallel feedback path.

Table 4: DVSG Results

	STBP [10]	TSSL-DFA	RT-TSSL-DFA		TSSL-BP	RT-TSSL-BP	TSSL-KP	RT-TSSL-KP
Reduced Time (l)	N/A	N/A	20	40	N/A	20	N/A	20
Average		76.53	78.40	80.14	87.29	87.36	85.97	86.46
Max	76.04	79.17	80.21	84.03	88.19	88.19	87.50	87.85
Min		73.96	76.74	77.43	85.76	86.46	83.68	84.38

All networks: P2->FC512->FC11

P2: Pooling layer 2, FC512: Fully Connected layer 512 outputs

Second, we present an optimization to the original TSSL-BP algorithm to limit the complexity of the algorithm, both for the original TSSL-BP and for the new TSSL-KP and TSSL-DFA methods. By limiting the time-based feedback of the original method in our Reduced Time (RT) algorithm, we greatly reduce the complexity of such algorithms while maintaining accuracy. We then demonstrate a method by which we estimate the complexity of each version of the feedback algorithm for integration onto neuromorphic hardware and show that our new algorithms, as well as the time-based optimizations result in a possible decrease of roughly 30% in complexity on multiple different networks.

Using the new methods we show that TSSL-KP matches or exceeds TSSL-BP on some networks while TSSL-DFA is outperformed by both but shows large reductions in complexity, allowing for trade-off cases when placed on constrained or neuromorphic hardware. Additionally, we demonstrate that the Reduced Time algorithm either improves, or has very little impact on the accuracy of a given network while simultaneously providing a reduction in complexity in all shown cases.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number DE-SC0021319.

REFERENCES

[1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. 2015. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems* 34, 10 (2015), 1537–1557.

[2] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. 2017. A Low Power, Fully Event-Based Gesture Recognition System. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7388–7397. https://doi.org/10.1109/CVPR.2017.781

[3] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. 2020. What is the State of Neural Network Pruning? *CoRR* abs/2003.03033 (2020). arXiv:2003.03033 https://arxiv.org/abs/2003.03033

[4] Sander M Bohte, Joost N Kok, and Han La Poutre. 2002. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 1-4 (2002), 17–37.

[5] Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. 2019. Direct Feedback Alignment with Sparse Connections for Local Learning. *CoRR* abs/1903.02083 (2019). arXiv:1903.02083 http://arxiv.org/abs/1903.02083

[6] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.

[7] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. 2015. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 1–8.

[8] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. 2015. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*. 1117–1125.

[9] Wulfram Gerstner and Werner M Kistler. 2002. *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge university press.

[10] Weihua He, YuJie Wu, Lei Deng, Guoqi Li, Haoyu Wang, Yang Tian, Wei Ding, Wenhui Wang, and Yuan Xie. 2020. Comparing SNNs and RNNs on neuromorphic vision datasets: Similarities and differences. *Neural Networks* 132 (2020), 108–120. https://doi.org/10.1016/j.neunet.2020.08.001

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Eric Hunsberger and Chris Eliasmith. 2016. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141* (2016).

[13] Yingyezhe Jin, Wenrui Zhang, and Peng Li. 2018. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in Neural Information Processing Systems*. 7005–7015.

[14] J.F. Kolen and J.B. Pollack. 1994. Backpropagation without weight transport. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, Vol. 3. 1375–1380 vol.3. https://doi.org/10.1109/ICNN.1994.374486

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[16] Jeongjun Lee, Renqian Zhang, Wenrui Zhang, Yu Liu, and Peng Li. 2020. Spike-Train Level Direct Feedback Alignment: Sidestepping Backpropagation for On-Chip Training of Spiking Neural Nets. *Frontiers in Neuroscience* 14 (2020), 143. https://doi.org/10.3389/fnins.2020.00143

[17] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. 2016. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience* 10 (2016), 508.

[18] Hesham Mostafa. 2017. Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems* 29, 7 (2017), 3227–3235.

[19] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. 2019. Surrogate gradient learning in spiking neural networks. *IEEE Signal Processing Magazine* 36 (2019), 61–63.

[20] Arild Nøkland. 2016. Direct feedback alignment provides learning in deep neural networks. *Advances in Neural Information Processing Systems* Nips (2016), 1045–1053. arXiv:1609.01596

[21] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. 2019. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience* 13 (2019).

[22] Sumit Bam Shrestha and Garrick Orchard. 2018. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*. 1412–1421.

[23] Matthew Bailey Webster, Jonghyun Choi, and changwook Ahn. 2021. Learning the Connections in Direct Feedback Alignment. https://openreview.net/forum?id=zgGmAx9ZcY

[24] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. 2018. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience* (2018).

[25] Friedemann Zenke and Surya Ganguli. 2018. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation* 30, 6 (2018), 1514–1541.

[26] Wenrui Zhang and Peng Li. 2019. Spike-train level backpropagation for training deep recurrent spiking neural networks. *Advances in neural information processing systems* (2019).

[27] Wenrui Zhang and Peng Li. 2020. Temporal Spike Sequence Learning via Backpropagation for Deep Spiking Neural Networks. *Advances in Neural Information Processing Systems* 33 (2020).