

# ADMM-Based Training for Spiking Neural Networks

Giovanni Perin<sup>\*†</sup>, Cesare Bidini<sup>\*</sup>, Riccardo Mazzieri<sup>\*</sup>, and Michele Rossi<sup>\*‡</sup>

<sup>\*</sup> Department of Information Engineering (DEI), University of Padova, Italy

<sup>†</sup> Department of Information Engineering (DII), University of Brescia, Italy

<sup>‡</sup> Department of Mathematics “Tullio-Levi Civita”, University of Padova, Italy

giovanni.perin@unibs.it, cesare.bidini@phd.unipd.it, riccardo.mazzieri@phd.unipd.it, michele.rossi@unipd.it

**Abstract**—In recent years, spiking neural networks (SNNs) have gained momentum due to their high potential in time-series processing combined with minimal energy consumption. However, they still lack a dedicated and efficient training algorithm. The popular backpropagation with surrogate gradients, adapted from stochastic gradient descent (SGD)-derived algorithms, has several drawbacks when used as an optimizer for SNNs. Specifically, the approximation introduced by the use of surrogate gradients leads to numerical imprecision, poor tracking of SNN firing times at training time, and, in turn, poor scalability. In this paper, we propose a novel SNN training method based on the alternating direction method of multipliers (ADMM). Our ADMM-based training aims to solve the problem of the SNN step function’s non-differentiability by taking an entirely new approach with respect to gradient backpropagation. For the first time, we formulate the SNN training problem as an ADMM-based iterative optimization, derive closed-form updates, and empirically show the optimizer’s convergence, its great potential, and discuss future and promising research directions to improve the method to different layer types and deeper architectures.

**Index Terms**—ADMM, spiking neural networks, NNs optimizers, gradient-free optimization, model-based learning.

## I. INTRODUCTION

Spiking neural networks (SNNs) adopt a neural architecture that closely mimics how the human brain works as an asynchronous, event-based dynamic system. They are especially interesting because they process the input over time, enabling real-time signal processing and inference. Moreover, they show three orders of magnitude of improvement in the energy-delay product (EDP) when running on dedicated hardware, being thus characterized by a great energy efficiency [1]. Remarkably, a dedicated training algorithm for SNNs has not yet been developed, and the methods used so far for their supervised training are adaptations of those adopted for traditional NNs [2]. A primary drawback with SNNs is the non-differentiable nature of the spiking neuron (Heaviside) activation function, which prevents the direct application of gradient-based optimization methods such as backpropagation. This is often circumvented by using backpropagation with *surrogate gradients* [3], where the SNN Heaviside function is

replaced with a continuous approximation during the backward pass, to enable the computation of its derivative. However, such an approximation prevents an exact tracking of the firing times of SNN neurons at training time, which ultimately impacts the quality of the solution found. This becomes more and more impactful as the number of SNN layers increases. Recently [4], researchers have adopted residual connections as a solution to this, but the problem remains; surrogate gradients still provide an approximation to the actual timing behavior of SNN neurons.

In this work, we propose a fundamentally different and new approach to SNN training. With our method, the activation times of firing neurons are *exactly tracked* without the need to use approximations of any sort (e.g., surrogate gradients). In detail, we propose a novel optimization framework specifically tailored for SNNs and based on the alternating direction method of multipliers (ADMM) [5]. The learning task is formulated as an optimization problem having the target training loss as its cost function, and defining the SNN neuronal dynamics as its optimization constraints. Our approach is *model-based*, as we optimize a model of the SNN where not only the network weights are learnable (optimized) variables, but also the state variables of the SNN neurons (i.e., the membrane potentials and firing events) are subjected to the learning process. This problem is solved by deriving closed-form updates for the involved variables, with a dedicated subroutine to *optimally handle* the SNN Heaviside function. We stress that this approach completely differs from SGD-derived optimizers, where a forward pass of the data is needed to estimate the value of the loss and the gradient direction before performing an optimization step.

In summary, the contributions of this work are: i) a new optimizer based on the ADMM thought specifically for SNNs is proposed, ii) the optimization problem is relaxed, making it treatable, and closed-form iterative updates with a solid mathematical theory are derived, iii) a subroutine to *optimally handle* the SNN Heaviside step function is developed, and iv) a proof-of-concept through numerical simulations to show the potential of the training algorithm is presented. Improvements to the proposed ADMM-based technique are possible and should be pursued to make the method scalable to very large and complex architectures. Such promising directions are discussed in Sect. VI.

This work has been supported by the EU through the Horizon Europe/JU SNS project ROBUST-6G (grant no. 101139068) and by the EU under the Italian National Recovery and Resilience Plan (NRRP) Mission 4, Component 2, Investment 1.3, CUP C93C22005250001, partnership on “Telecommunications of the Future” (PE00000001 - program “RESTART”).

## II. RELATED WORK

### A. SNNs and training algorithms

SNNs offer a promising energy-efficient alternative to traditional artificial NNs through sparse, event-driven, and asynchronous computation. However, their non-differentiable spike dynamics pose a major challenge for their training.

The surrogate gradient method [3] is currently the dominant approach for training SNNs. It bypasses the spike function's discontinuity by introducing a smooth, differentiable surrogate gradient during the backward pass, enabling the application of the backpropagation through time (BPTT) algorithm. Despite its empirical success, this technique comes with several drawbacks: it requires additional state variables and memory, introduces approximation errors that degrade performance, and is also susceptible to vanishing or exploding gradients, as the number of NN layers grows.

Complementary to gradient-based methods, biologically plausible, gradient-free "Hebbian" learning rules have historically received attention in the neuroscience literature. A notable example is spike-timing-dependent plasticity (STDP), a local and biologically inspired learning approach, showing promising results for unsupervised learning and feature extraction [6]. However, STDP struggles with scalability and accuracy in complex tasks and requires extensive hyperparameter tuning. More recently, forward-only learning algorithms [7]–[9] aim to bypass the need for backward gradient computation altogether by integrating learning directly during the forward pass. However, these approaches lack a strong theoretical underpinning, are not yet scalable to deep architectures, and are not competitive with gradient-based alternatives. Additionally, very few explore their use with SNNs and mainly focus on training traditional neural networks.

Altogether, existing approaches entail trade-offs between biological plausibility, learning effectiveness, and hardware efficiency. This motivates the increasing research interest in alternative training paradigms beyond backpropagation, specifically tailored to the dynamics of SNNs.

### B. Gradient-free training with the ADMM

The use of dual methods and especially the ADMM [5] to develop gradient-free optimizers for NNs is rooted in the seminal paper [10], where the authors show a model-based optimization formulation of a feed-forward NN. The proposed alternating direction solution can be seen as a split Bregman iteration or an inexact formulation of the ADMM. In recent years, the approach gained momentum and has been improved with relaxed approaches to non-convex constraints and a computationally cheap estimation of the pseudoinverse matrix, making the algorithm faster and more efficient [11]–[13]. Some of these works also prove the convergence of the approach to a stable minimizer. Notably, in [14] the authors integrate the recent advances with Anderson acceleration, making the ADMM significantly faster and outperforming, among others, advanced SGD-based approaches like Adam and RMSprop. In recent years, alternatives to gradient-based methods have also been investigated for recurrent NNs

(RNNs) [15]–[17], also using the ADMM. For the first time, in this paper, we apply a formulation similar to [10] to the neuronal dynamics of SNNs, which are particular in many aspects (see Sect. III). To the best of our knowledge, we are the first to propose a gradient-free optimizer specifically tailored for SNNs and based on the ADMM optimization algorithm.

## III. OPTIMIZATION PROBLEM FORMULATION

Throughout the paper, the layer and time indices will be denoted by  $l = 1, \dots, L$  and  $t = 1, \dots, T$ , respectively.  $M$  denotes the number of samples of the dataset (batch), and  $n_l$  denotes the number of neurons at layer  $l$ . The symbol  $\|\cdot\|$  will be used to refer to the Frobenius matrix norm, and  $\langle \cdot \rangle$  will denote the Frobenius inner product.

The training optimizer is formulated in this section as a model-based optimization problem. Specifically, three sets of matrices (the optimization variables) are defined, namely, i) the model parameters (weights) for each layer  $\mathbf{W} = \{W_l\}$ , where  $W_l$  has dimension  $n_l \times n_{l-1}$  (and the cumulative number of entries of the matrices in  $\mathbf{W}$  is  $\sum_{l=1}^L n_l \times n_{l-1}$ , where  $n_0$  refers to the number of input neurons); ii) the membrane potentials (pre-activations) for each layer and time  $\mathbf{z} = \{z_{l,t}\}$ , where each  $z_{l,t}$  has dimension  $n_l \times M$  (and the cumulative number of entries of  $\mathbf{z}$  is  $T \times M \times \sum_{l=1}^L n_l$ ); and iii) the spikes (post-activations) for each layer and time  $\mathbf{a} = \{a_{l,t}\}$ , where again each  $a_{l,t}$  has dimension  $n_l \times M$  (and the cumulative number of entries of  $\mathbf{a}$  is  $T \times M \times \sum_{l=1}^{L-1} n_l$ , as the output layer does not have an activation). The objective function is the target loss. We assume it depends on the membrane potentials of the last layer at the last time index  $T$  and on the dataset labels  $y$  as  $\ell(z_{L,T}, y)$ . The SNN's dynamics are modeled through the constraints relating the optimization variables. This translates to the optimization problem

$$\begin{aligned} \min_{\mathbf{W}, \mathbf{z}, \mathbf{a}} \quad & \ell(z_{L,T}, y) \\ \text{s.t.} \quad & z_{l,1} = W_l a_{l-1,1}, & \forall l, \\ & z_{l,t} = \delta z_{l,t-1} + W_l a_{l-1,t} + & \\ & \quad - \vartheta a_{l,t-1}, & l \neq L, t \neq 1, \\ & z_{L,t} = \delta z_{L,t-1} + W_L a_{L-1,t}, & t \neq 1, \\ & a_{l,t} = H_{\vartheta}(z_{l,t}), & l \neq L, \forall t, \end{aligned} \quad (1)$$

where  $a_{0,t}$  are the input training data ( $M$  samples). The first constraint models the first time step of every layer, which is an equation similar to the dynamics of a common feed-forward NN: the input is multiplied by the weights matrix to obtain the output. The second constraint encodes the neuronal dynamics for the following time steps at every layer except the last one: the coefficient  $\delta$  is the exponential decay factor for the membrane potential, representing the neuron's memory concerning the previous instant, while  $\vartheta$  is the firing threshold. If a neuron at layer  $l$  fires at time  $t$ , we have  $a_{l,t} = 1$  and the membrane potential loses a voltage equal to  $\vartheta$  (otherwise  $a_{l,t} = 0$ ). The third constraint expresses the dynamics of the last layer, where firing and the reset mechanism are disabled. The fourth constraint is the relation

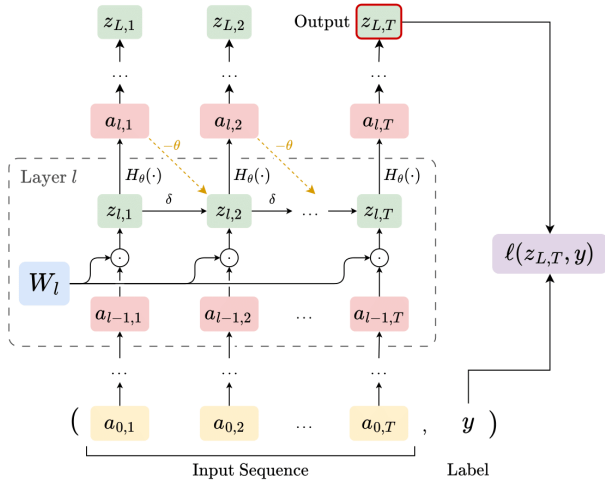


Fig. 1. Graphical representation of the SNN model used for the problem formulation in this paper.

between membrane potentials and spikes: a neuron emits a spike when the potential exceeds the threshold  $\vartheta$ . Note that this can be interpreted computationally as the activation function of the SNN: a Heaviside step function centered in  $\vartheta$  (and denoted here by  $H_{\vartheta}$ ). A pictorial representation of the model is given in Fig. 1.

#### A. Problem relaxation

Due to the non-convexity of the problem constraints, a relaxed version of problem (1) is derived, where the constraints are added to the objective function as penalties. Following the original approach in [10] and the papers that originated from it, we keep an exact constraint on the dynamics of the last membrane potential. The (relaxed) augmented Lagrangian [5] associated with problem (1) is given in Eq. (2), where  $\lambda$  is a Lagrange multiplier and  $\rho$  and  $\sigma$  are parameters tuning the relative weight of the soft constraints concerning the cost function. Intuitively, the Lagrange multiplier for the last membrane potential must be kept because the presence of the objective function adds a drift to the quadratic penalty minimizer. For all the other variables, instead, the sole optimization term is the soft penalty. Therefore, upon convergence, the optimized value of the variables will satisfy the constraints.

#### IV. ADMM-BASED SOLUTION

In this section, we tackle the solution to the relaxed version of the problem via the augmented Lagrangian of Eq. (2), using an alternating direction optimization along the three directions

defined by the three sets of optimization variables, plus the dual variable (Lagrange multiplier) update. Without loss of generality, in this work, for the loss function, we use the mean squared error (MSE) between the last membrane potential and the target label, i.e.,

$$\ell(z_{L,T}, y) = \|z_{L,T} - y\|^2. \quad (3)$$

This choice is convenient as it leads to a closed-form update. However, closed-form updates can be obtained for many other valid loss functions, and the method can be used even by retrieving the solution to subproblems via numerical solvers when a closed-form solution is not available.

We observe that by minimizing for each variable separately while considering the others as fixed parameters, the Lagrangian in Eq. (2) is a convex function. Specifically, each ADMM update solves an unconstrained quadratic program (least squares). This ensures that every update has a simple closed-form solution, which can be easily computed via matrix multiplications (and efficiently implemented on GPUs). Next, we use  $\mathbb{1}(\cdot)$  as the indicator function, e.g.,  $\mathbb{1}(t < T)$  denotes all time steps smaller than  $T$ .

#### A. Weights update

We conveniently define the auxiliary tensors

$$x_l = [z_{l,1}, z_{l,2} - \delta z_{l,1} + \vartheta a_{l,1}, \dots, z_{l,T} - \delta z_{l,T-1} + \vartheta a_{l,T-1}],$$

for  $l = 1, \dots, L-1$ , and

$$x_L = [z_{L,1}, z_{L,2} - \delta z_{L,1}, \dots, z_{L,T} - \delta z_{L,T-1}].$$

By setting to zero the partial derivatives, the following updates for the weights variables are obtained, as the solution of a linear regression:

$$W_l = \left( \sum_{t=1}^T x_{l,t} a_{l-1,t}^\top \right) \left( \sum_{t=1}^T a_{l-1,t} a_{l-1,t}^\top \right)^{-1} \quad l \neq L, \quad (4)$$

$$W_L = \left( \frac{1}{\rho} \lambda a_{L-1,T}^\top + \sum_{t=1}^T x_{L,t} a_{L-1,t}^\top \right) \left( \sum_{t=1}^T a_{L-1,t} a_{L-1,t}^\top \right)^{-1}. \quad (5)$$

#### B. Pre-activations update

By minimizing the direction associated with the membrane potential variables  $z_{l,t}$  and by momentarily ignoring the presence of the non-differentiable activation function  $H_{\vartheta}$ , we find

$$z_{l,t} = \frac{q_{l,t} + \delta(r_{l,t+1} + \vartheta a_{l,t}) \mathbb{1}(t < T)}{1 + \delta^2 \mathbb{1}(t < T)} \quad l \neq L, \quad (6)$$

$$z_{L,t} = \frac{\rho s_{L,t} + \rho \delta r_{L,t+1} \mathbb{1}(t < T)}{\rho + \rho \delta^2 \mathbb{1}(t < T) + 2 \mathbb{1}(t = T)} + \frac{(2y - \lambda) \mathbb{1}(t = T) + \delta \lambda \mathbb{1}(t = T - 1)}{\rho + \rho \delta^2 \mathbb{1}(t < T) + 2 \mathbb{1}(t = T)}, \quad (7)$$

$$\begin{aligned} \mathcal{L}_{\rho,\sigma}(\mathbf{W}, \mathbf{z}, \mathbf{a}, \lambda) = & \ell(z_{L,T}, y) + \frac{\rho}{2} \sum_{l=1}^L \|z_{l,1} - W_l a_{l-1,1}\|^2 + \frac{\rho}{2} \sum_{l=1}^{L-1} \sum_{t=2}^T \|z_{l,t} - \delta z_{l,t-1} - W_l a_{l-1,t} + \vartheta a_{l,t-1}\|^2 \\ & + \frac{\rho}{2} \sum_{t=2}^T \|z_{L,t} - \delta z_{L,t-1} - W_L a_{L-1,t}\|^2 + \frac{\sigma}{2} \sum_{l=1}^{L-1} \sum_{t=1}^T \|a_{l,t} - H_{\vartheta}(z_{l,t})\|^2 + \langle z_{L,T} - \delta z_{L,T-1} - W_L a_{L-1,T}, \lambda \rangle. \end{aligned} \quad (2)$$

where, for convenience, we have defined the auxiliary tensors

$$\begin{aligned} p_l &= W_l[a_{l-1,1}, \dots, a_{l-1,T}], \\ s_l &= p_l + \delta[0, z_{l,1}, \dots, z_{l,T-1}], \\ q_l &= s_l - \vartheta[0, a_{l,1}, \dots, a_{l,T-1}], \quad \text{and} \\ r_l &= -p_l + z_l. \end{aligned}$$

Now, we observe that, for  $l \neq L$ , the terms  $\|a_{l,t} - H_\vartheta(z_{l,t})\|^2$  have to be considered (see Eq. (2)).

The non-differentiability of  $H_\vartheta$  requires a dedicated subroutine that tests the value of the objective function when  $H_\vartheta$  is active/inactive (if logic). Specifically, since each entry contributes to the Lagrangian cost in an additive and separable way, we evaluate if commuting the Heaviside step yields a better solution entry-wise (see Algorithm 1). In Line 4 the algorithm checks whether  $z_{l,t}^{n_l,m} > \vartheta$ , causing a spike. However, lowering it below the threshold to prevent firing would yield a better solution, hence we set  $z_{l,t}^{n_l,m}$  to the highest possible value not surpassing the threshold (i.e.,  $\vartheta$  itself), as it is the best solution according to the cost function (a Euclidean norm). The dual case appears in Line 6, where the membrane potential  $z_{l,t}^{n_l,m}$  is below the threshold  $\vartheta$  and generating a spike would yield a better solution. Therefore, the potential is set slightly above the threshold  $\vartheta$ , using a user-defined and small parameter  $\varepsilon > 0$ .

---

**Algorithm 1** z minimizer subroutine

---

```

1: procedure Z_MINIMIZER( $z$ )
2:   for each entry of  $z$  do
3:     if  $z_{l,t}^{n_l,m} > \vartheta$  &  $\text{cost}(\vartheta) \leq \text{cost}(z_{l,t}^{n_l,m})$  then
4:        $z_{l,t}^{n_l,m} \leftarrow \vartheta$ 
5:     else if  $z_{l,t}^{n_l,m} \leq \vartheta$  &  $\text{cost}(\vartheta + \varepsilon) < \text{cost}(z_{l,t}^{n_l,m})$ 
6:       then
7:          $z_{l,t}^{n_l,m} \leftarrow \vartheta + \varepsilon$ 
8:       end if
9:   end for
10: end procedure

```

---

*C. Post-activations update*

For the post-activations (i.e., the spikes) update, we note that this variable only exists up to and including layer  $l = L - 1$ , as layer  $L$  does not have an activation function. For compactness, we define the auxiliary tensors

$$\begin{aligned} u_l &= [z_{l,1}, z_{l,2} - \delta z_{l,1}, \dots, z_{l,T} - \delta z_{l,T-1}], \\ v_l &= u_l + \vartheta[0, a_{l,1}, \dots, a_{l,T-1}], \quad \text{and} \\ w_l &= u_l - W_l[a_{l-1,1}, \dots, a_{l-1,T}]. \end{aligned}$$

The spikes  $a_{l,t}$  are obtained as given in Eqs. (8) and (9).

Neurons' spikes are physically constrained to be binary variables: either a neuron fires ( $a_{l,t} = 1$ ) or it does not ( $a_{l,t} = 0$ ). However, projecting the value retrieved in the binary space produces instability for the ADMM iterations towards convergence, as the values might often change abruptly. Therefore, we instead adopt a clipping procedure enforcing  $0 \leq a_{l,t} \leq 1$  (i.e., we set  $a_{l,t} = \min(\max(0, a_{l,t}), 1)$ ).

*D. Training algorithm*

The full training algorithm using the  $z$  minimizer subroutine explained in Algorithm 1 is summarized in Algorithm 2. Algorithm 2 represents a single ADMM iteration, which must be repeated until a convergence criterion is met [5].

---

**Algorithm 2** ADMM optimizer for SNNs

---

```

1: for  $l = 1, \dots, L - 1$  do
2:   Update the weights  $W_l$  with Eq. (4)
3:   for  $t = 1, \dots, T$  do
4:     Update the pre-activations  $z_{l,t}$  with Eq. (6)
5:      $z_{l,t} \leftarrow \text{z\_minimizer}(z_{l,t})$ 
6:     if  $l < L - 1$  then
7:       Update the post-activations  $a_{l,t}$  with Eq. (8)
8:     else
9:       Update the post-activations  $a_{L-1,t}$  with Eq. (9)
10:    end if
11:     $a_{l,t} \leftarrow \min(\max(0, a_{l,t}), 1)$ 
12:  end for
13: end for
14: Update the weights  $W_L$  with Eq. (5)
15: for  $t = 1, \dots, T$  do
16:   Update the pre-activations  $z_{L,t}$  with Eq. (7)
17: end for
18:  $\lambda \leftarrow \lambda + \rho(z_{L,T} - \delta z_{L,T-1} - W_L a_{L-1,T})$ 

```

---

V. NUMERICAL SIMULATIONS

The simulations were implemented in Python using PyTorch tensors<sup>1</sup> in full compatibility with the existing snnTorch package<sup>2</sup>. In these simulations, whose settings are summarized in Tab. I, we used the neuromorphic dataset N-MNIST. We also adopted a *stochastic ADMM* approach where the order of the updates was chosen randomly for the layers and the time steps, as this procedure yielded better results.

In Fig. 2, the training accuracy reached for 1,000 ADMM iterations is shown varying the number of hidden layers and averaging the result across four different runs (shaded areas represent the max-min range of values). As can be seen, the framework is currently solid in solving the classification task with a single hidden layer, with  $\sim 98.6\%$  accuracy on average, but the performance decreases with an increasing number of hidden layers (around 86% accuracy for two hidden layers and

<sup>1</sup>Code available at [https://github.com/cesarbid/SNN\\_ADMM\\_Optimizer](https://github.com/cesarbid/SNN_ADMM_Optimizer)

<sup>2</sup><https://snntorch.readthedocs.io/en/latest/>

$$\begin{aligned} a_{l,t} &= (\rho W_{l+1}^\top W_{l+1} + (\sigma + \rho \vartheta^2 \mathbb{1}(t < T)) I)^{-1} (\rho W_{l+1}^\top v_{l+1,t} - \rho \vartheta w_{l,t+1} \mathbb{1}(t < T) + \sigma H_\vartheta(z_{l,t})), \quad l \neq L - 1, \\ a_{L-1,t} &= (\rho W_L^\top W_L + (\sigma + \rho \vartheta^2 \mathbb{1}(t < T)) I)^{-1} (W_L^\top (\rho u_{L,t} + \lambda \mathbb{1}(t = T)) - \rho \vartheta w_{L-1,t+1} \mathbb{1}(t < T) + \sigma H_\vartheta(z_{l,t})). \end{aligned} \quad (8) \quad (9)$$

TABLE I  
PARAMETERS USED FOR THE NUMERICAL SIMULATIONS.

| Parameter                                                   | Value             |
|-------------------------------------------------------------|-------------------|
| Number of time steps $T$                                    | 150               |
| Neurons per layer $n_l$                                     | 512               |
| Number of classes $n_c$                                     | 10                |
| Optimization parameters $(\rho, \sigma, \delta, \vartheta)$ | (1, 0.1, 0.95, 1) |
| (Total, warming) ADMM iterations                            | (1000, 300)       |
| Number of training samples $M$                              | 200               |

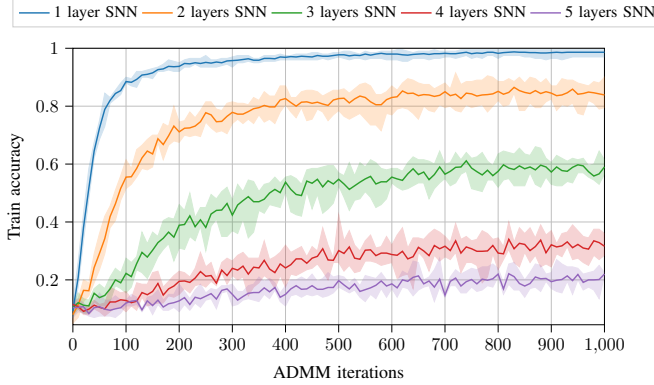


Fig. 2. Train accuracy obtained in 1,000 ADMM iterations varying the number of hidden layers.

progressively lower). The reason is better investigated in the convergence study shown in Figs. 3 and 4, evaluated for the training of the SNN with two hidden layers.

Figure 3a shows the relaxed Lagrangian value during training. From this plot, we see that, after a quick and significant decrease in the first 100 iterations, the Lagrangian reaches a plateau around iteration 200. Hence, it might seem that a stable minimizer has been found; however, by observing the target loss function (Fig. 3b), we see that the value is still decreasing and would continue the trend even after iteration 1,000. Fig. 4 shows the residuals normalized by  $\sqrt{T \times M \times n_l}$ , following a logic similar to the one used to define the stopping criterion for ADMM in [5]. Specifically, we observe that the primal residuals (Fig. 4a) suddenly drop to  $10^{-5}$  when the Lagrange multiplier is activated, showing the efficacy of the dual approach. However, the residuals of the soft constraints for the neuronal dynamics (Fig. 4b) and the activation (Fig. 4c) decrease at a slower pace: The plateau reached by the soft constraints of the hidden layers hence dominates the contribution to the relaxed Lagrangian of Fig. 3a. This causes a *mismatch between the optimization and the prediction at inference time*, since the neuronal dynamics are not respected with enough numerical precision, and activation variables  $\mathbf{a}$  are not exactly in  $\{0, 1\}$ . The effect becomes more evident as the number of layers and time steps increases. Allowing the soft constraints to further reduce their values with more training iterations would improve the performance.

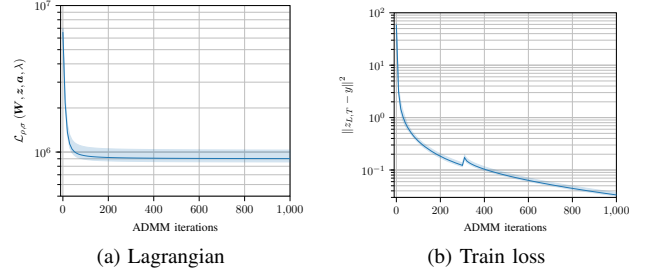


Fig. 3. Objective functions of the ADMM optimization: relaxed Lagrangian (left) and target loss function (right). Training of the SNN with two hidden layers.

## VI. DISCUSSION

The current approach shows potential, but there is still room for improvement. We note that, unlike classic NNs, SNNs have a time dimension that adds a factor of the order of  $10^3$  in the number of variables, making the problem more complex to tackle. Nonetheless, the ADMM is known for handling problems with up to  $\sim 10^9$  variables when several empirical techniques are combined with the standard framework [5], as the convergence might be extremely sensitive to hyperparameters, as we also observed in our experiments. For instance, as also done in other works, an adaptive value for  $\rho$  and  $\sigma$  across iterations and the addition of Anderson acceleration [14] are expected to speed up the convergence of the soft constraints. These enhancements would also make the optimizer more scalable concerning the number of layers and time steps.

In the present work, we demonstrated that this first approach effectively handles the presence of the non-differentiability of the Heaviside step function, as the dedicated subroutine developed produces a stable (although slow) convergence towards the minimizer, representing a significant novelty concerning the commonly used surrogate gradient methods. We also note that the approach can be extended to convolutional SNN layers by modifying the soft constraints relative to the membrane potential dynamics. In this case, the multiplication between the weight matrix  $W_l$  and the spikes  $a_{l-1,t}$  would be replaced by a correlation operation, which is still linear and thus can be handled by the ADMM. It is also important to observe that, like in previous approaches derived from [10], the proposed training algorithm is suitable for processing the entire dataset, instead of processing small batches in series and performing many (imprecise) backpropagation steps with a coarse estimation of the gradient, as SGD-based optimizers do. In sharp contrast, with the ADMM, fewer optimal subproblems are executed at each iteration, spanning, in principle, the whole dataset. We additionally note that the available memory can be a bottleneck in this case. The solution to this is the consideration that variables  $z_{l,t}$  and  $a_{l,t}$  are independent sample-wise: hence, several CPUs and/or GPUs *can process different subsets of the dataset in parallel*. However, the weights update requires

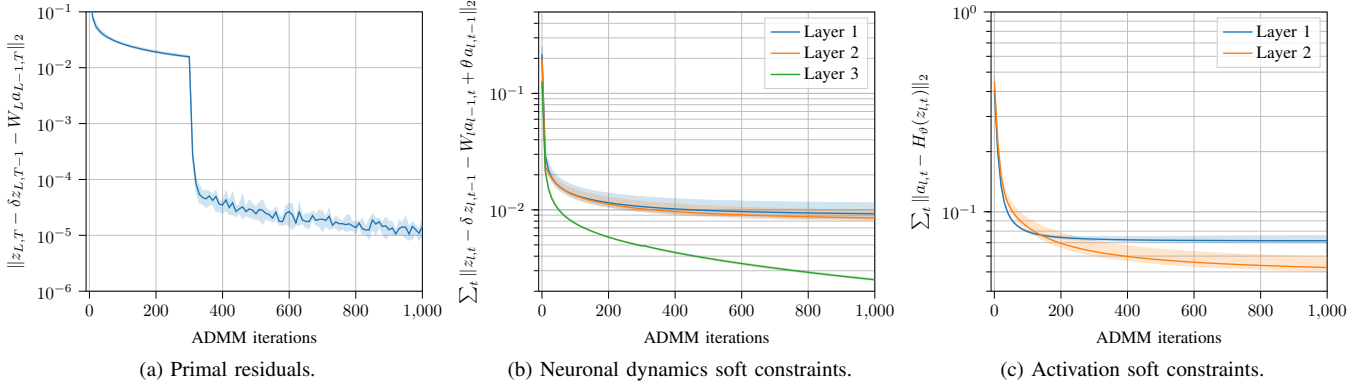


Fig. 4. Constraint residuals normalized by their dimensions ( $\sqrt{T \times M \times n_l}$ ). Exact constraint primal residual of the last membrane potential (left), norm of the soft constraints relative to the membrane potential dynamics per layer (middle), and norm of the soft constraints relative to the neurons' spikes per layer (right). Training of the SNN with two hidden layers.

integrating the information with the update

$$W_l = \left( \sum_{n=1}^N \sum_{t=1}^T x_{l,t}^n a_{l-1,t}^{n\top} \right) \left( \sum_{n=1}^N \sum_{t=1}^T a_{l-1,t}^n a_{l-1,t}^{n\top} \right)^{-1}, \quad (10)$$

where  $n$  is the worker (similarly, batch) index. Remarkably, this directly enables federated learning (FL), where the parameter server (PS) computes the reduction update in Eq. (10), while the clients keep a portion of the dataset and update the variables  $z_{l,t}^n$  and  $a_{l,t}^n$  locally and privately.

## VII. CONCLUDING REMARKS

In this work, we presented a first formulation of an ADMM optimizer specifically tailored for SNN training. The iterative solution only uses closed-form updates and a subroutine with if-else logic to optimally handle the presence of the non-differentiable Heaviside step function. Notably, the proposed solution solves the poor approximation issue of backpropagation with surrogate gradients. With this work, we aim to start a new research line on SNN optimizers: We believe that the proposed method has great potential, and there exists significant room for improvement in terms of the type of layers supported, convergence speed, memory utilization, and scalability to large networks.

## REFERENCES

- [1] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra, and A. Wild, "NxTF: An API and compiler for deep spiking neural networks on Intel Loihi," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, pp. 1–22, 2022.
- [2] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.
- [3] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.
- [4] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian, "Deep residual learning in spiking neural networks," in *Neural Information Processing Systems (NeurIPS)*, 2021.
- [5] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al., "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.
- [6] A. Safa, T. Verbelen, I. Ocket, A. Bourdoux, H. Sahli, F. Catthoor, and G. Gielen, "Fusing event-based camera and radar for slam using spiking neural networks with continual stdp learning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 2782–2788.
- [7] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.
- [8] A. Kohan, E. A. Rietman, and H. T. Siegelmann, "Signal propagation: The framework for learning and inference in a forward pass," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 6, pp. 8585–8596, 2024.
- [9] Geoffrey Hinton, "The forward-forward algorithm: Some preliminary investigations," arXiv preprint arXiv:2212.13345, <https://arxiv.org/abs/2212.13345>, 2022.
- [10] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable ADMM approach," in *International conference on machine learning*. PMLR, 2016, pp. 2722–2731.
- [11] J. Wang, F. Yu, X. Chen, and L. Zhao, "ADMM for efficient deep learning with global convergence," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 111–119.
- [12] J. Wang, Z. Chai, Y. Cheng, and L. Zhao, "Toward model parallelism for deep neural network based on gradient-free ADMM framework," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 591–600.
- [13] X. Wen and Y. Lei, "A Fast ADMM Framework for Training Deep Neural Networks Without Gradients," in *2024 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2024, pp. 1–8.
- [14] Z. Ebrahimi, G. Batista, and M. Deghat, "AA-mDLAM: An accelerated ADMM-based framework for training deep neural networks," *Neuro-computing*, vol. 633, pp. 129744, 2025.
- [15] Y. Tang, Z. Kan, D. Sun, L. Qiao, J. Xiao, Z. Lai, and D. Li, "ADMMiRNN: Training RNN with Stable Convergence via an Efficient ADMM Approach," in *Machine Learning and Knowledge Discovery in Databases*. 2021, pp. 3–18, Springer International Publishing.
- [16] A. Bemporad, "Training recurrent neural networks by sequential least squares and the alternating direction method of multipliers," *Automatica*, vol. 156, pp. 111183, 2023.
- [17] A. D. Adeoye and A. Bemporad, "An inexact sequential quadratic programming method for learning and control of recurrent neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 2, pp. 2762–2776, 2025.