
Article:
Shen, J, Zhao, Y, Liu, JK et al. (2023) HybridSNN: Combining Bio-Machine Strengths by Boosting Adaptive Spiking Neural Networks. IEEE Transactions on Neural Networks and Learning Systems, 34 (9). 5841 -5855. ISSN: 2162-237X

https://doi.org/10.1109/tnnls.2021.3131356

HybridSNN: Combining Bio-machine Strengths by Boosting Adaptive Spiking Neural Networks

Jiangrong Shen, Yu Zhao, Jian K. Liu, and Yueming Wang

***Abstract*—Spiking neural networks (SNNs), inspired by the neuronal network in the brain, provide biologically relevant and low-power consuming models for information processing. Existing studies either mimic the learning mechanism of brain neural networks as closely as possible, e.g. the temporally local learning rule of Spike-Timing-Dependent Plasticity (STDP), or apply the gradient descent rule to optimize a multi-layer SNN with fixed-structure. However, the learning rule used in the former is local and how the real brain might do the global-scale credit assignment is still not clear, which means that those shallow SNNs are robust but deep SNNs are difficult to be trained globally and could not work so well. For the latter, the non-differentiable problem caused by the discrete spike trains leads to inaccuracy in gradient computing and difficulties in effective deep SNN. Hence, a hyrid solution is interesting to combine shallow SNNs with an appropriate machine learning technique not requiring the gradient computing, which is able to provide both energy-saving and high-performance advantages. In this paper, we propose a HybridSNN, a deep and strong SNN composed of multiple simple SNNs, in which data-driven greedy optimization is used to build powerful classifiers, avoiding the derivative problem in gradient descent. During the training process, the output features (spikes) of selected weak classifiers are fed back to the pool for the subsequent weak SNN training and selection. This guarantees HybridSNN not only represents the linear combination of simple SNNs, as what regular AdaBoost algorithm generates, but also contains neuron connection information, thus closely resembling the neural networks of a brain. HybridSNN has the benefits of both low power consumption in weak units and overall data driven optimizing strength. The network structure in HybridSNN is learned from training samples, which is more flexible and effective compared with existing fixed multi-layer SNNs. Moreover, the topological tree of HybridSNN resembles the neural system in the brain, where pyramidal neurons receive thousands of synaptic input signals through their dendrites. Experimental results show that the proposed HybridSNN is highly competitive among the state-of-the-art SNNs.**

***Index Terms*—Spiking Neural Networks, HybridSNN, Boosting, Adaptive Structures.**

This work was partly supported by grants from the National Key $R\&D$ Program of China (2018YFA0701400), the Starry Night Science Fund of Zhejiang University Shanghai Institute for Advanced Study (SN-ZJU-SIAS-002), the Zhejiang Lab (2019KE0AD01), the Chuanqi Research and Development Center of Zhejiang University, the Fundamental Research Funds for the Central Universities (2021KYY600403-0001), Zhejiang Lab (2019KC0AB03 and 2019KC0AD02), and the Royal Society Newton Advanced Fellowship (No. NAF-R1-191082). (Corresponding author: Yueming Wang).

Jiangrong Shen, Yu Zhao are with the College of Computer Science and Technology and the Qiushi Academy for Advanced Studies, Zhejiang University. (email: jrshen@zju.edu.cn, 21821275@zju.edu.cn)

Yueming Wang is with the Qiushi Academy for Advanced Studies, State Key Lab of CAD&CG in Zhejiang University and Zhejiang Lab. (email: ymingwang@zju.edu.cn.)

Jian K. Liu, Jiangrong Shen are also with the Centre for Systems Neuroscience, Department of Neuroscience, Psychology and Behaviour, University of Leicester, Leicester LE1 7RH, U.K., and also with the School of Computing, University of Leeds, Leeds LS2 9JT, U.K. (e-mail: j.liu9@leeds.ac.uk).

I. Introduction

THE human brain, with more than 100 billion neurons, is capable of myriad complex intelligent tasks. Based on limited collective understanding of brain mechanisms, artificial models have been intensely explored to simulate the efficient information propagation of the brain [1]. Meanwhile, the rapidly increased computational power and largely enriched training datasets made it possible for brain-inspired computing systems, such as artificial neural networks (ANNs), to achieve breakthroughs, especially in pattern recognition and machine learning fields [2] [3] [4].

However, ANNs are highly energy-consuming since they transfer information in the real-number model, rather than using discrete spikes as in the brain systems. Specifically, the brain neuron emits an action potential, or spike, upon stimulus larger than the firing threshold of the neuron membrane; or otherwise, it keeps silent. Spiking neural networks (SNNs), known as the third generation of the artificial neural systems, are developed based on such event-driven mechanisms of the brain to transmit information. SNNs are therefore more biologically relevant and energy-efficient than ANNs [5] [6] [7].

Much effort has been made on designing SNN architectures trained by different kinds of methodologies. [8]. These SNN structures can be roughly categorized into single- and multi-layer networks.

Tempotron [9] is a supervised synaptic learning algorithm commonly used for binary classification problems and can perform effectively in single-layer neural networks. But the shallow structure of single-layer network shows limited feature extraction capability and Tempotron cannot be directly extended to multi-layer networks due to the abandonment of precise firing time data which is required to transfer information across layers. To overcome such problem, various methods based on the Widrow-Hoff learning rule [10] were introduced to single-layer SNNs, including remote supervised learning method (ReSuMe) [11], precise spike-driven synaptic plasticity (PSD) [12], and spike pattern association neuron (SPAN) [13]. In addition, the single-layer SNN model based on Spike-Timing-Dependent Plasticity (STDP) was also explored. For instance, the network with competitive STDP neurons was constructed to detect repeating patterns, in which the neuron firing behavior would trigger the inhabitation of other neurons through the lateral connections [14]. Among them, ReSuMe and PSD could be extended to multi-layer frameworks by backpropagation (BP) rule [15] [16]. However, these extended models still perform poorly in solving complex problems, such

as recognizing digits in the MNIST dataset. On the other hand, the shallow structures of single-layer SNNs exhibit limited feature extraction capability. Hence, sophisticated and powerful SNN frameworks remain highly desirable for carrying out complex tasks.

Since, researchers have proposed several advanced multi-layer SNN models. The mainstream ones aim to train the systems by gradient descent algorithms (supervised) and/or STDP (unsupervised) rule [17]. Based on the approximate gradient descent rule, the Spikeprop learning algorithm [18] was introduced, followed by the Quickprop and RProp for faster convergence [19]. Combining event-based STDP and BP rule, BP-STDP is developed to update weights at each time step as a temporal local learning approach [20]. To overcome the non-differentiable nature of spike event, Lee et al. [21] considered the discontinuity of spiking signal as noise and treated the membrane potential of spiking neurons as the differentiable signal. Hence, the error backpropagation mechanism can be employed to train deep SNNs directly. To achieve the remarkable performance comparable to convolutional networks (ConvNets)[22], the conversion method from ANNs to SNNs was designed in [23]. By employing the rectified linear units (ReLUs) during training and introducing the new weight normalization method to regulate the firing rate, it further enhanced the performance of SNNs obtained from the conversion process. Meanwhile, unsupervised learning methods have also been developed for SNNs. A three-layer network, with the winner-take-all (WTA) circuit of excitatory neurons and inhibitory ones, was designed and trained by STDP to categorize data [24]. However, this model could not be extended to multiple layers directly to build a strong and deep SNN. There are also some SNN models proceeding optimization by combining the unsupervised learning and supervised learning method. For instance, to better initialize the parameters in the unsupervised training multi-layer networks prior to supervised optimization, a two-phase training methodology was introduced in [25]. It first trained the convolutional kernels in an unsupervised layer-specific way, then fine-tuned the synaptic weights with spike-based supervised gradient descent backpropagation. Moreover, the spiking deep convolutional neural network (SDNN) [26] and the spiking convolutional neural network (SpiCNN) [27] with deeper layers were proposed for object recognition. These two networks are composed of several convolutional and pooling layers followed by the linear SVM or the fully-connected layer as the final classifier. Furthermore, the SpiCNN model improved the feature learning efficiency by introducing 3*3 kernels in two convolutional layers. In these two models, the convolutional layer were trainable through unsupervised STDP rule while the classifiers were trained by supervised method. Furthermore, the ensemble unsupervised spiking neural network was proposed for objective recognition [28]. Several SNNs with the same structure were integrated together and computed the final classification result by voting algorithm. The accuracies of this model on image classification problem were comparable with the deep neural networks. In aggregate, these methods made tremendous contribution to the multi-layer SNN development. However, these existing models could not implement the adaptive learning process when building the deep SNNs model architectures in the data-driven manner.

From the above, two common types of deep SNN design strategies emerged in these studies. One is to resemble the network structure and learning paradigms of brain neural networks as closely as possible. Most of these models are trained with the temporally local unsupervised STDP and always introduce some brain-inspired component, such as lateral inhabitation connections, to make model efficient to generate selective responses. Nevertheless, the scalability of those model is limited by the insufficient understanding of brain mechanisms. That is, the exact networks structures and the global credit assignment mechanism in brain networks are not clear enough to support the designing and learning of a thorough brain-style deep SNN [29] [30]. In consequence, the performance of those models usually is limited by their shallow structures and can hardly compete with the supervised deep SNNs or the traditional artificial neural networks, such as convolutional neural networks (CNNs), on open AI challenges using public datasets. The other strategy is to combine the basic SNN principles with machine learning (ML) techniques. Those models often employ the predefined and fixed network structures and are trained by the BP algorithms with gradient descent rule. However, the non-differential problem caused by the discrete spike trains hampers the optimization progress by BP algorithms in SNNs. Since then, the BP algorithms in some SNNs are implemented by approximate functions or surrogate gradient method with the built-in inaccuracy or the lack of theoretical foundations [31]. Meanwhile, there are some SNNs converted from a fully-trained deep neural networks, which may suffer from the performance loss caused by the converting process. In addition, most of the structures of these models are fixed and can not be generated adaptively. Despite such drawback, the strategy combining SNN units with ML methods remains attractive because the SNN frameworks have the advantage of energy efficiency and the data-driven ML methods offer high performance. With the above two parts complementing each other, such hybrid SNN hence became highly appealing.

Herein, we propose a new SNN framework, called HybridSNN. Unlike the existing models using fixed network structures, HybridSNN can construct the deep SNNs adaptively. It treats simple SNNs as basic processing units, greedily searches the best units in a data-driven manner, and assembles them into a deep and strong SNN. An SNN unit can be a single- or multi-layer SNN trained to become a weak classifier. Then the best weak classifiers are greedily selected and integrated into the model, one at a time. Next, the training data gets re-weighted so that the correctly classified samples are assigned with small weights and the mistaken samples are set with large values. The process iterates until a final strong model is established. This approach has something in common but quite different from the ensemble procedure of AdaBoost [32], that is, we use simple SNNs as weak classifiers, and the output features (spikes) of selected classifiers are fed back to the sample pool for subsequent training and selection. This feedback operation is important for building an SNN system with neural networks, rather than just a linear combination of weak SNNs.

Our method has the following advantages:

- HybridSNN has the benefits of both biologically plausible model and data-driven optimizing strength by using a data-driven greedy optimization method to boost an effective combination of shallow SNN units instead of gradient descent. The resulting model retains an energy-saving advantage and the boosting learning part is able to train a deep SNN pursuing higher performance comparable to ANNs. Thus it has both strengths in energy efficiency and performance.
- The network topologies of HybridSNN are more flexible and adaptable compared with those of the existing single- and multi-layer SNNs. Instead of using fixed deep structures, we start from a single unit and then assemble a deep network through iterations of classifier selection and incorporation. The network structure in HybridSNN is learned adaptively and depends on the training samples, which provides promising scalability for our model to suit different pattern recognition tasks and improves the computational efficiency.
- HybridSNN has a tree-like structure and each weak classifier contributes to the final result. This is different from most existing deep multi-layer SNNs that use only the output of the last layer to produce the final result. The topology of HybridSNN is closer to that of biological neuronal networks [33], [34], where pyramidal neurons receive thousands of input signals through their dendrites. HybridSNN is therefore not just the linear combination of simple SNNs. It contains neuron connection information since the output spikes are fed back to the pool of training samples. Thus, it performs tasks by making committee decisions like a brain neural system.

Experiments were carried out on the MNIST and CIFAR-10 datasets. Three models of HybridSNN framework were tested: T-HybridSNN, a single-layer SNN with Tempotron learning [9]; M-HybridSNN, a multi-layer SNN with the learning rules proposed by Mostafa [35]; and C-HybridSNN, a deep convolutional SNN model designed in [36]. For T-HybridSNN, we explore how to speed up the convergence through the built-in pretraining method, discuss the generated tree-like network structures and investigate the effect of the ensemble way. For weak learners in M-HybridSNN and C-HybridSNN, we adopted the learner weight enhancement method to improve the performance on MNIST and CIFAR-10 dataset, respectively. Our approach outperformed not only the original weak learners but also most of the benchmark learning methods.

II. Method

In this section, we first introduce the framework of HybridSNN, in which the processing units are simple single- or multi-layer SNNs. For each unit, the best classifier is greedily sought in a data-driven manner. Finally, the selected units are assembled into a deep and powerful SNN. Also, we describe the detailed process of how to train T-HybridSNN with single-layer SNN, M-HybridSNN with multi-layer SNNs, C-HybridSNN with deep convolutional SNNs, respectively.

A. The Framework of HybridSNN

Inspired by the AdaBoost algorithm of forward stage-wise additive modeling using a multi-class exponential loss function [32], [37], we designed the HybridSNN by employing a process to iteratively generate the pool of weak SNN classifiers that are strengthened through learning and the best performing classifier is chosen during each iteration.

As shown in Fig. 1, for each iteration of the HybridSNN framework, the weak learner pool consists of m learning models of $\Xi_0^m, \Xi_1^m, \Xi_2^m, ..., \Xi_{m-1}^m$, in which Ξ_l^m denotes the classifier unit that utilizes the output of l_{th} trained classifier as the input data and Ξ_0^m regards the weighted raw data as input data. In detail, based on the raw training samples with the same normalized weights, the HybridSNN builds its initial classifier pool with only one classifier unit Ξ_0^1. Then the weights of training samples are updated according to the classification results of Ξ_0^1. As soon as a training sample is misclassified, its corresponding weight would be adjusted, which is also known as changing the bias of the input samples through an increased intensity of misclassified samples. The classifier in the first iteration can only be Ξ_0^1 and marked as T^1. The next iteration has a pool consisting of Ξ_0^2 and Ξ_1^2, where Ξ_0^2 is built on the weighted input data updated by the previous training stage, and Ξ_1^2 takes the output of Ξ_0^1 classifier as the input data. The classifier T^2 is chosen from Ξ_0^2 and Ξ_1^2 according to the performance. As such, there would be a total M chosen classifiers $(T^1, T^2, ..., T^M)$ at the completion of the training process. Due to their input data sources, these classifiers are naturally connected and each makes a contribution to the final decision making, furnishing together a tree-like topology.

B. Training Process of HybridSNN

The training process of HybridSNN is illustrated in Algorithm 1. Given input data with the same normalized weights, the HybridSNN is trained to obtain M classifiers $T^{(m)}, m \in \{1, 2, ..., M\}$ with different performance scores. The classifier pool contains m SNN units $\Xi_l^m, l \in \{0, 1, ..., m-1\}$. The first step is to train these classifiers by the weighted data. Then $err^{*(l)}$ is computed from the weighted classification results. Finally, the classifier $T^{(m)}$ with the smallest $err^{*(l)}$ value is chosen and assigned a performance score $\alpha^{(m)}$, which will further update the sample weights. The sample weights are normalized. After the training process, the assembled classifier $C(x_i)$ proceeds to predict data categorization with weighted weak learners in the testing process. In this paper, we consider three models: T-HybridSNN, M-HybridSNN, and C-HybridSNN. The specific training method for each of the three models is described as below:

1) T-HybridSNN as Single Layer Network Learned by Tempotron: T-HybridSNN is a scenario where a single-layer network is used as the basic classifier in HybridSNN. We start the training process with the classical Tempotron algorithm. To speed up the convergence, we adopt a pretraining method to initialize the weight of the weak learner.

HybridSNN with classical Tempotron learning. Each learner pool consists of different SNN classifiers. To effectively compute and assign the sample weights for the

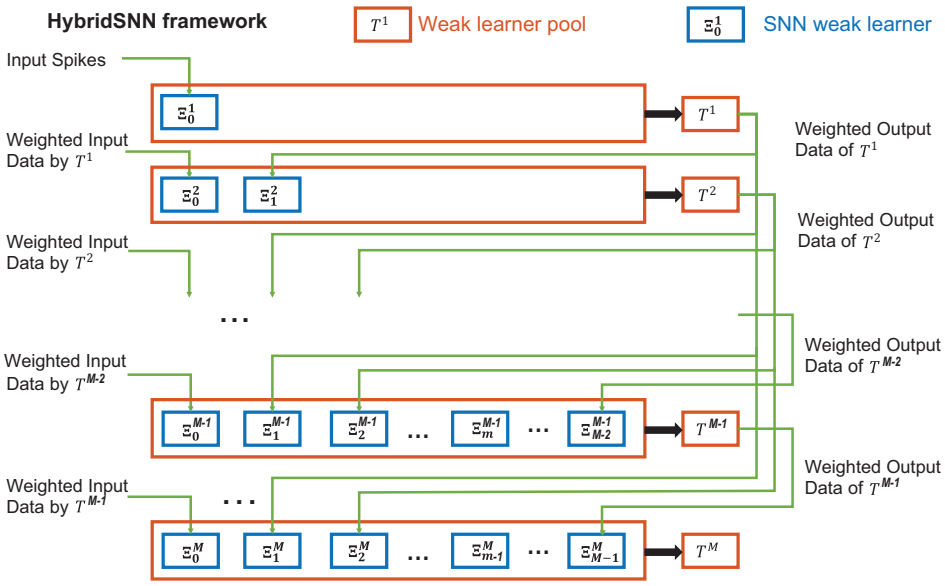


Fig. 1: The framework of HybridSNN. We assume there are M iterations in the HybridSNN model. Each iteration uses one weak learner pool. During the m_{th} iteration, the weak learner pool consists of m classifiers: $\Xi_0^m, \Xi_1^m, \Xi_2^m, ..., \Xi_{m-1}^m$. These classifiers compete against each other performance-wise. The best learner T^m is chosen after the competition. Hence, there would be M classifiers $(T^1, ..., T^M)$ chosen after the entire training process. These classifiers are associated with each other through their input data source and iteration indexes. The input data source could be the original dataset or the output from the selected classifiers in the previous iterations. The first classifier Ξ_0^m in the m_{th} iteration employs the original data weighted by the prior classifier T^{m-1} as the input source. The remaining classifiers employ the weighted output data of the corresponding prior classifiers. For instance, Ξ_2^{m-1} utilizes the weighted output spikes of T^2 as input. As such, a tree-like topology will be generated by the HybridSNN model at the completion of the training process.

Algorithm 1 The training process of HybridSNN framework.

Initialization:

Input data: $x_i, i = 1, 2, ..., N$.
Labels of input data: $c_i, i = 1, 2, ..., N$.
Number of classes: K.
The weights of sample: $W_i^s = 1/N, i = 1, 2, ..., N$.
for $m = 1$ to M **do** ← **HybridSNN Iterations**
  for $l = 0$ to $m-1$ **do** ← **Weak Learner Pool**
    $min\_error = Inf$.
    (a) Train the classifier $\Xi_l^m(x_i)$ with W_i^s.
    (b) Compute error:
      $err^{*(l)} = \sum_{i=1}^N W_i^s \Gamma(c_i \neq \Xi_l^m(x_i)) / \sum_{i=1}^N W_i^s$.
    If $err^{*(l)} < min\_error$
      $min\_error = err^{*(m)} = err^{*(l)}$.
      $T^{(m)}(x_i) = \Xi_l^m(x_i)$.
  end for
  (c) Compute $\alpha^{(m)} = log\frac{1-err^{(m)}}{err^{(m)}} + log(K-1)$.
  (d) Set $W_i^s \leftarrow W_i^s \cdot exp(\alpha^{(m)} \Gamma(c_i \neq T^{(m)}(x_i)))$.
  (e) Re-normalize $W_i^s = \frac{W_i^s}{\sum_1^N W_i^s}$.
end for

Output:

$C(x_i) \leftarrow argmax_k \sum_{m=1}^M \alpha^{(m)} \cdot \Gamma(T^{(m)}(x_i) = k)$.

SNNs, we adopt the Tempotron learning rule for its excellent classification ability working with spike patterns.

Neuron model. The widely used Leaky Integrate-and-Fire (LIF) model is applied as the fundamental neuronal unit to construct the network. The membrane potential of a specific LIF neuron j is calculated as follows:

$$V_j(t) = \sum_{i=1}^{N_I} \Theta(t - t_i) W_{ij} K(t - t_i), \tag{1}$$

where N_I is the total number of input neurons and Θ is the Heaviside step function. K denotes the vanishing kernel function of postsynaptic potential for $t_i > t$, which can be further represented as:

$$K(t - t_i) = V_0(exp(-\frac{t - t_i}{\tau_m}) - exp(-\frac{t - t_i}{\tau_s})), \tag{2}$$

where V_0 is used for normalization to ensure that the maximum kernel value is 1.0 and the synaptic efficacies W_{ij} becomes the amplitude of the unitary postsynaptic potential. The parameters τ_m and τ_s are the decay time constants of membrane integration and synaptic currents, respectively, and are set to be 15 ms and 3.75 ms for the LIF neuron, respectively. Once the membrane potential V_j crosses the threshold V_{thr} of 1.0, the neuron j emits a spike. Each neuron is permitted to emit a spike only once.

In binary classification, the input patterns to the neurons belong to one of the two types, $\oplus$ and $\ominus$. The neuron fires a spike when $\oplus$ arrives, and remain inert upon $\ominus$ input. Tem-

potron rule updates the synaptic weights of W_{ij} to minimize the error signals as:

$$E_j = \begin{cases} W_j^s * (V_{thr} - V_j(t_{max})) & if \oplus error, \\ W_j^s * (V_j(t_{max}) - V_{thr}) & if \ominus error, \end{cases} \tag{3}$$

where W_j^s denotes the weight of the sample j, t_{max} is the time when the neuron reaches its maximum voltage. $\oplus$ error represents the false negative error when the neuron should emit spikes but fails to, while $\ominus$ error is the false positive error when the neuron wrongly emit spikes. The gradients of parameter W_{ij} are computed as following:

$$\begin{cases} \Delta W_{ij}^+ & = W_j^s * \lambda_w \sum_{t_i<t_{max}} K(t_{max} - t_i) \quad if \oplus error, \\ \Delta W_{ij}^- & = -W_j^s * \lambda_w \sum_{t_i<t_{max}} K(t_{max} - t_i) \quad if \ominus error, \end{cases} \tag{4}$$

where λ_w is the learning rate for network weights update.

The pretraining method for classifiers in the HybridSNN model. Before training, the parameters of each classifier (except the first one) are set to be the corresponding values of the trained classifier from the last iteration. This approach can effectively accelerate the convergence process.

2) M-HybridSNN as a Multi-layer Network Learned by Mostafa model: M-HybridSNN is a HybridSNN model using a multi-layer network as the primary weak learner. We first choose the Mostafa algorithm for the learning procedure of this model. In order to balance the weights of classifiers in different iterations, we introduce the score enhancement method for HybridSNN.

Neuron model. The Mostafa method employs non-LIF neurons with exponentially decaying synaptic current kernels. Assuming the postsynaptic neuron j receives N_I spikes at time $\{t_1, t_2, ..., t_{N_I}\}$ with weights $\{w_1, w_2, ..., w_{N_I}\}$ from N_I presynaptic neurons, and each neuron is only permitted to fire once, the membrane potential of neuron j can be represented as follows:

$$V_j(t) = \sum_{i=1}^{N_I} \Theta(t - t_i) w_i (1 - exp(-(t - t_i))). \tag{5}$$

Once the value of V_j crosses the threshold $V_{thr} = 1$, the neuron j emits a spike. The casual set $C_j = \{i : t_i < t_j\}$ is defined to collect the presynaptic spikes that determine the time point at which postsynaptic neuron fires the first spike. Hence t_j satisfies:

$$1 = \sum_{i \in C_j} w_i (1 - exp(-(t_j - t_i))). \tag{6}$$

After transforming the spike times by $exp(t_x) \to z_x$, the first spike of neuron j can be described in the z-domain as:

$$z_j = \frac{\sum_{i \in C_j} w_i z_i}{\sum_{i \in C_j} w_i - 1}. \tag{7}$$

Mostafa as a supervised learning method. For the feedforward process of fully connected multi-layer SNNs, the firing time of each neuron is computed according to its causal set. If the causal set is empty, the output spike time is set to be infinity. The cross-entropy loss is applied to make the neuron of the correct class fires earlier than others in the output layer. Assuming the spike time of the output layer is z_o, and the target class is g, the cost of output layer is as follows:

$$L(g, z_o) = W^s * N * (-ln \frac{exp(-z_o[g])}{\sum_k exp(-z_o[k])}), \tag{8}$$

where W^s denotes the sample weights. Moreover, for the backward propagation, the derivatives of the presynaptic neurons' first spike times in the z-domain and the corresponding weights are given by:

$$\frac{dz_j}{dw_i} = \begin{cases} \frac{z_i - z_j}{\sum_{i \in C_j} w_i - 1} & \text{if } i \in C_j, \\ 0 & \text{Otherwise.} \end{cases} \tag{9}$$

$$\frac{dz_j}{dz_i} = \begin{cases} \frac{w_i}{\sum_{i \in C_j} w_i - 1} & \text{if } i \in C_j, \\ 0 & \text{Otherwise.} \end{cases} \tag{10}$$

Based on these formulas, the derivatives of other variables in the networks can be obtained with standard backpropagation technique, using the errors transferred through the layers. Besides, the constraints on synaptic weights and gradient normalization are applied to ensure neurons can fire spikes.

HybridSNN stages balanced by score enhancement. Compared with single-layer SNNs, the multi-layer SNNs can generally represent features more effectively, hence achieving better performance for classification problems. However, when a multi-layer SNN is used as the basic weak learner in HybridSNN, the performance is mostly determined by the initial classifier due to its high classification accuracy and high performance score. The classifiers of the following iterations of HybridSNN seem to be unworkable due to their dramatically lower scores, and the proposed HybridSNN hence fail to take advantage of the hierarchical cascading structure. To solve this problem, we design a brand new method to achieve a more balanced state among different training stages previously disturbed by ultra-big/small imbalanced weights.

An enhanced parameter of es is introduced to the score enhancement method. With this parameter, the weight score $\alpha^{(m)}$ of trained weak learner m is computed as follows:

$$\alpha^{(m)} = \frac{log \frac{1 - err^{(m)}}{err^{(m)}} + log(K - 1)}{es}, \tag{11}$$

in which es decreases as the iterations proceed. It can be designed as C/m, where C is a constant and m denotes the iteration index. In this way, the weight score of the deep iterations can be enhanced, while the domination effect of the initial classifier on the result is avoided. Once misclassified samples occur in the first iteration, this score enhancement procedure can correct the wrong labels for the HybridSNN model.

3) C-HybridSNN model: The deep convolution SNN structure is designed with spike-based supervised gradient descent backpropagation algorithm [36], which employs an approximate derivative for LIF neuronal function.

Neuron model. The LIF neuron sub-threshold dynamics is as follows:

$$\tau_q \frac{dV}{dt} = -V + \sum_{i=1}^{N_i} \Theta(t - t_i) W_i, \tag{12}$$

where V is the postsynaptic membrane potential and τ_q is the time constant, which is set to be 100. The neuron fires a spike when $V >= V_{thr}$. Each neuron can fire multiple spikes. Using this basic LIF model with $V_{thr} = 1$, the output of hidden layers, including the convolutional layer and spatial pooling layer, can be computed according to the neuron model. Especially, in the last time step of the final layer, the firing threshold V_{thr} is set to be ∞ and the output of neuron $output_{final}$ is obtained by:

$$output_{final} = \frac{V_{mem}(T)}{number\ of\ total\ time\ steps}, \tag{13}$$

where $V_{mem}(T)$ represents the accumulated membrane potential over T time steps. With these operations, the deep convolutional SNN structure, such as VGG and ResNet, could be efficiently constructed. Herein, this deep convolutional SNN with VGG structure is chosen as the basic weak learner, and named as C-HybridSNN model. The spike train generated in the fully-connected layer of each weak learner is regarded as its output data and will be transmitted to the next iteration in the C-HybridSNN model. The aforementioned score enhancement method is also adopted in the C-HybridSNN model.

III. Results

A. Experiment Settings

Numerical experiments were conducted with the MNIST dataset and CIFAR-10 dataset. Each image can be encoded as a series of spike patterns [38], [35], [39], and used as input for HybridSNN.

In addition, we categorized the whole MNIST dataset into complex groups and simple groups according to the binary classification results using the original Tempotron rule. In particular, a pair of digits that were too similar to distinguish from each other was considered as a complex group. The rest pairs were simple groups. There were a total of $C_{10}^2 = 45$ groups for binary classification, 6 among which are complex groups, namely $2\&3, 3\&5, 3\&8, 4\&9, 5\&8, 7\&9$. A number of both complex and simple groups were chosen as experimental examples to illustrate the HybridSNN model.

B. Encoding Method

In order to encode images as spike trains, we employ a temporal coding scheme to encode the original image dataset into spatial-temporal patterns. The T-HybridSNN model uses encoding model in the convolutional SNN (CSNN) model [38], which converts the activation values of the perceptron algorithm into delay spike times by linear mapping. The strongly activated value would fire earlier, and vice versa. This sparse spatiotemporal representation extracts the key information from the original images. On the other hand, the M-HybridSNN model adopts the image binarization method to generate spike trains. The encoding neuron with high-intensity pixels fires a spike at time 0, while neuron with low-intensity pixels fires at time $ln(6) = 1.79$, corresponding to $z = 6$ in the z-domain. This setting provides a suitable temporal interval between spikes mapping from high- and low-intensity pixels. Since the C-HybridSNN model is used to recognize the color images in the CIFAR-10 dataset, the input pixels are normalized and bounded to the range of $[-1, 1]$. Next, these normalized pixel intensities are converted to Poisson-distributed spike trains.

C. The Performance of T-HybridSNN with Single-layer SNN

In this section, we evaluate the performance of the T-HybridSNN model, which is the HybridSNN framework with single-layer Tempotron model. Firstly, the influence of the pretraining method for the T-HybridSNN is explored. Secondly, the generated tree-like structure by the T-HybridSNN model is analyzed. Finally, the effect of the ensemble learning way of the T-HybridSNN is investigated by comparing its performance to the Adaboost method with Tempotron model.

1) The Influence of Pretraining Method on T-HybridSNN: We investigate the performance of T-HybridSNN with pretraining method by computing the classification accuracies on MNIST dataset. In detail, the training and test accuracies of the T-HybridSNN model are evaluated on two complex groups, $4\&9$ and $5\&8$ within 200 training iterations. For these binary classifications, the number of decoding neurons in the basic weak learner of Tempotron is set to be 80.

We find that this pretraining method could accelerate the convergence of the T-HybridSNN model. As illustrated in Fig. 2 (a), the test accuracy for $4\&9$ classification with pretraining rises to 95% at the 23^{rd} iteration, while it takes 165 iterations to reach the same level of test accuracy for model without pretraining, even though the final test results tend to be close after 200 iterations. Meanwhile, as shown in Fig. 2 (b), the test accuracy for $5\&8$ classification with pretraining outperforms the model without pretraining by at least 1% after 100 iterations, although the gap between the two decreases over the subsequent course. In summary, experimental results show that the learning process of T-HybridSNN model can be accelerated by pretraining method.

2) The Generated Tree-like Structure of T-HybridSNN: We then proceed to explore the learned network structure of T-HybridSNN. After 200 iterations, the connection among T-HybridSNN learners appears to be a tree-like topology, resembling the neural system of a brain. Here we choose several representative subtrees/networks generated by T-HybridSNN for the binary classification of $4\&9$. As illustrated in Fig. 3, the deepest subtree has six layers. Combining with Fig. 2 (a), we further analyze the relationship between the generated network structure and classification accuracy. Firstly, we find the weak learner prefers to take raw data as input when the test accuracy of the prior classifier declines. For discrete cascading T-HybridSNN, the test accuracy keeps falling from the 40^{th} to the 50^{th} iteration. Thus, the following 50^{th} to 54^{th} iterations employ raw data as input features, which in turn improve the test accuracy until a small peak is reached. Secondly, after the model is trained thoroughly and becomes stable, the iterations tend to distribute in the deeper layers of the tree. The test accuracy converges from the 150^{th} to 200^{th} iteration, located in the 3^{rd} to 6^{th} layers in the generated networks.

3) The Effect of the Ensemble Way of T-HybridSNN: Here, we analyze the impact of different cascading systems on SNN

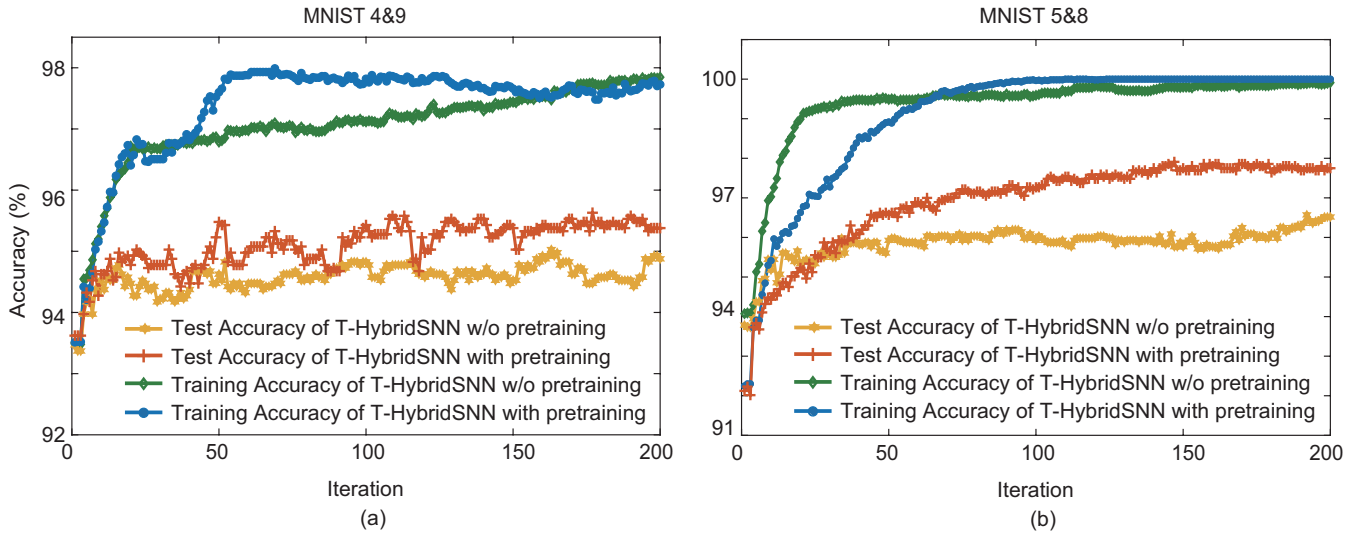


Fig. 2: The classification accuracies of the pretraining method for T-HybridSNN. 'T-HybridSNN with pretraining' and 'T-HybridSNN w/o pretraining' denote the discrete cascade SNNs with and without the pretraining method, respectively. (a), (b) show the performances of $4\&9$, $5\&8$, respectively, in comparison.

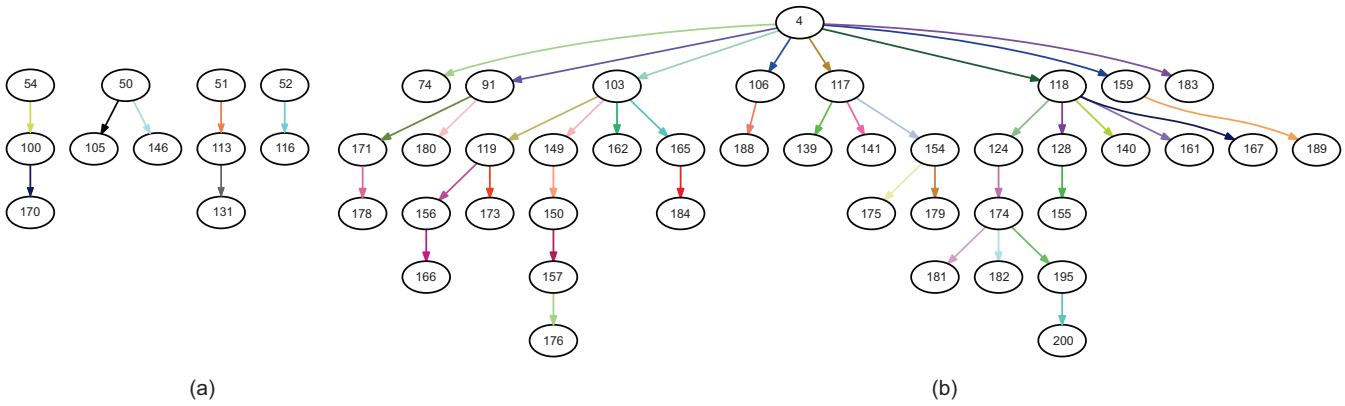


Fig. 3: Generated tree topology of T-HybridSNN by the classification of $4\&9$. Several subtrees from the topology are visualized. Each node represents a weak learner chosen from the pool of weak classifiers and is marked as T_m in Fig. 1. The number on each node denotes the index of the corresponding weak learner pool in the HybridSNN framework.

and compare their performances on two complex groups of MNIST dataset. Two models are compared: T-AdaBoost that is the original AdaBoost framework with Tempotron classifier and T-HybridSNN with Tempotron learning algorithm.

We find that the ensemble way of T-HybridSNN has an advantage over T-Adaboost. We compute the classification accuracies on two complex groups of the MNIST dataset, $4\&9$ and $5\&8$. These results are obtained with validation dataset, which is split from training dataset with 80% proportion. The validation set can be used to find the appropriate moment to measure the result during the training process. As shown in Fig. 2 (a), the test accuracy curve exists the oscillation phenomenon. These small oscillations are caused by few samples that are classified corretly in the last iteration but wrongly in the next iteration. Considering that case, we employ the validation set to record the optimal parameters that contain the number of iterations and the corresponding weak learners during the training process of T-HybridSNN. For each iteration, the weak learner is chosen only when the validating accuracy increases after it is assembled. As illustrated in Fig. 4 (a), the T-HybridSNN model performs better than T-AdaBoost on $4\&9$ binary recognition, with final test accuracies of 95.73% and 94.02%, respectively. For $5\&8$ (Fig. 4 (b)) binary recognition, the test accuracy of T-HybridSNN increases first, then decreases, and increases again to the value of 96.68%. Also, this T-HybridSNN model achieves higher test accuracy than T-AdaBoost on $5\&8$ recognition. In addition, both these two results in Fig. 4 obtain competitive test accuracies with fewer weak learners compared with the results in Fig. 2, 30 and 47 weak learners for $4\&9$ and $5\&8$ classification, respectively. These weak learners compose the necessary nodes for the generated tree-like structure by T-HybridSNN. In this way, the generated tree is relatively simple, instead of the redundancy structure in Fig. 3. We therefore conclude that T-HybridSNN outperform T-AdaBoost on these classifications, which indicates the effect of the ensemble way of T-HybridSNN.

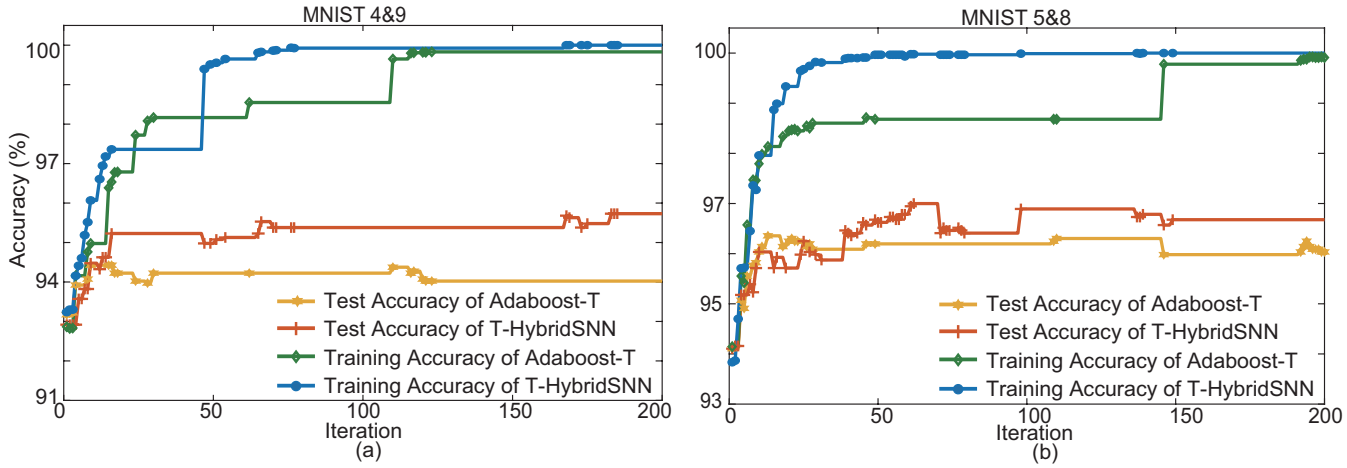


Fig. 4: The classification accuracies of the T-Adaboost, T-HybridSNN on two complex groups, 4&9 and 5&8. T-AdaBoost represents the AdaBoost algorithm with the Tempotron model as the weak learner. T-HybridSNN denotes the HybridSNN with the original Tempotron. The markers on the lines represents the chosen classifiers by validation datasets within 200 iterations.

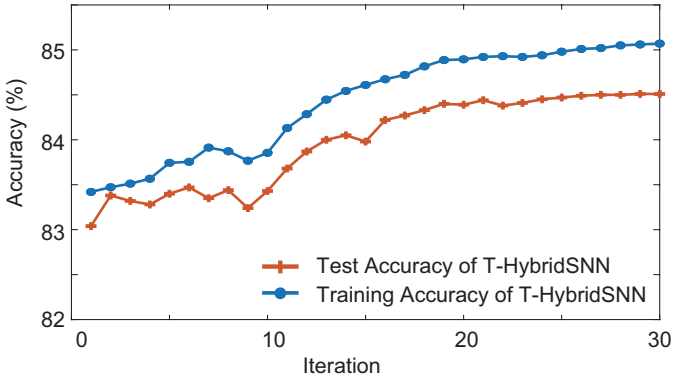


Fig. 5: The classification accuracies of the T-HybridSNN on the entire MNIST dataset.

However, the performance improvement of T-HybridSNN seems quite limited, which is caused by the weakness of the basic weak learner. To illustrate this phenomenon more clearly, we further investigate the classification accuracy of T-HybridSNN on the entire MNIST dataset. To improve the fitting ability of the basic weak learner, a relatively powerful Tempotron with a bit more complex network structure is employed as the basic weak learner here. That is, the number of decoding neurons in the output layer of Tempotron is set to be 120. The T-HybridSNN with this Tempotron is attempted to classify the MNIST dataset into 10 digit classes using 30 iterations. As shown in Fig. 5, we find that the training accuracy and test accuracy of T-HybridSNN keep improving slowly with the increasing of iterations and finally converge to be 85.07% and 84.51%, respectively. These training accuracy and test accuracy are higher than that of the original Tempotron classifier with 83.58% and 83.3%. This is because the HybridSNN can learn something more useful information for recognition compared with the single one Tempotron. The boosting theory indicates that weak classifiers can be assembled to a stronger one by adjusting sample weights and training each weak classifier recognizing a part of samples only [37]. In this case, a weak Tempotron does not need to learn information on all samples but only focuses on a few samples. This reduces the influence of the weak property of Tempotron and makes that information learnable. Then step by step, the trained Tempotron are connected to complement each other until a final strong classifier (T-HybridSNN). As shown in Fig. 5, the experiment results demonstrate the above analysis. The learning curve of T-HybridSNN continuously goes up meaning it has successfully learned information for recognition. Accordingly, both the training accuracy and the test accuracy are higher than those of the single Tempotron. It is worth noting that this does be one advantage of HybridSNN. Nevertheless, the results achieved by T-HybridSNN are still not competitive to most of the commonly used classifiers. Hence, we change the weak learner unit from a single-layer Tempotron to multi-layer Mostafa SNN. The test accuracy in ten-class MNINST reaches 97.84%, which is higher than baselines and comparable to the state-of-the-art as shown in Table IV. Therefore, the limited performance of HybridSNN model is related to the weak learners' capability.

In fact, this is consistent with another property of the boosting algorithm [40] [41]. That is, when the problem is not so difficult such as the frontal face detection (a binary classification problem) [42], the weak classifier can be really weak but still helpful for the final strong classifier, as long as its error rate is lower than 50%. It is easy to satisfy since it just needs an error rate slightly better than the random guess in a binary classification problem. Thus, T-HybridSNN obtains a higher accuracy in binary classification with a weak and single-layer SNN, i.e. Tempotron. However, classification units cannot be too weak when the problem changes to multi-class classification. Because the task becomes difficult, the boosting algorithm needs relatively strong classifier units to guarantee that the training process reachs a strong classifier with nice generalization ability. Thus, in this case, the multi-layer Mostafa SNN works better than Tempotron and obtains comparable performance to the state-of-the-art model as il-

lustrated in Table IV. Therefore, how to choose the basic SNN unit in the HybridSNN model to achieve comparable performance depends on the properties of problems and the unit itself. Hence, we would choose the more powerful weak learners, such as the three-layer SNNs and convolutional SNNs, to compose the M-HybridSNN and C-HybridSNN model and implement the data recognition for MNIST dataset and CIFAR-10 dataset in the next sections, respectively.

D. The Performance of M-HybridSNN with Multi-layer SNNs

As introduced in the last section, considering the limited performance of T-HybridSNN with single-layer SNN, the more powerful M-HybridSNN with multi-layer SNN is explored in this section. Firstly, the influence of the parameter in the score enhancing operation is investigated to find the optimal parameter. Secondly, the relationship between the recognized dataset's complexity and the depth of the generated tree-like structure is discussed. Finally, the performance of M-HybridSNN is compared to different kinds of SNNs models to show its benefits.

1) The Influence of the Score Enhancing Parameters on M-HybridSNN: To explore the effect of different parameters of M-HybridSNN, we recorded the classification accuracy of HybridSNN after 10 iterations with fixed 800 hidden neurons. Experiments are done with four groups of datasets containing two complex sample groups $(2\&3, 5\&8)$ and two simple ones $(0\&1, 1\&2)$ in MNIST.

As shown in Table I, the best results are achieved when the weak learner enhancement parameter es is set to be $(e * ln10)/t$ after 20 iterations, where e is the base of the natural logarithm. Meanwhile, the accuracy continues to improve when the number of iteration grows with otherwise the same enhancement parameters. Noticeably, the average accuracy of M-HybridSNN exceeds that of the original Mostafa method, indicating that the proposed score enhancement mechanism can effectively avoid the problem of domination by the first iteration in our M-HybridSNN model.

2) The Depth of the Tree-like Structure Generated by M-HybridSNN: To explore the learned structures of M-HybridSNN, we assess the performance of HybridSNN models on several representative complex and simple groups in MNIST datasets by recording the accuracy of both training and test processes. Meanwhile, the maximum number of cascading layers is counted to show the depth of the tree-like network structures.

We conducted the binary classification on all complex groups. As illustrated in Table II, the average accuracy of M-HybridSNN is better than that of the original Mostafa method with a 0.4% improvement on the test dataset. Besides, there are 3 or 4 cascading layers after the training process for the M-HybridSNN. Meanwhile, we randomly select six simple groups for binary classification, as shown in Table III. The average accuracy of HybridSNN reaches 99.52% on the test dataset, exceeding that of the original Mostafa method by 0.24%. And the average cascading layers are 2.5 for simple groups, confirming that the network needs a deeper structure to solve complex problems than simple ones. In conclusion, we find that M-HybridSNN can achieve better average accuracy than the original Mostafa model for both complex and simple samples. Moreover, the performance can be improved more readily on complex sample groups than simple ones due to more cascading layers. Hence, the more complicated the problems to solve, the more complex and deeper network structures are needed. And our HybridSNN model proves to have the scalability to adapt to different pattern recognition tasks by the capability of generating flexible network topologies.

3) The Accuracy Comparison of the M-HybridSNN with Other Models: In this section, we compare our proposed HybridSNN with other existing models using the entire MNIST dataset. All 60000 training samples and 10000 testing samples are used for performance comparison with several typical learning algorithms of SNN by recording the classification accuracy. We also assess the performance of HybridSNN by comparing the original Mostafa SNN with a similar network structure of a fixed topology.

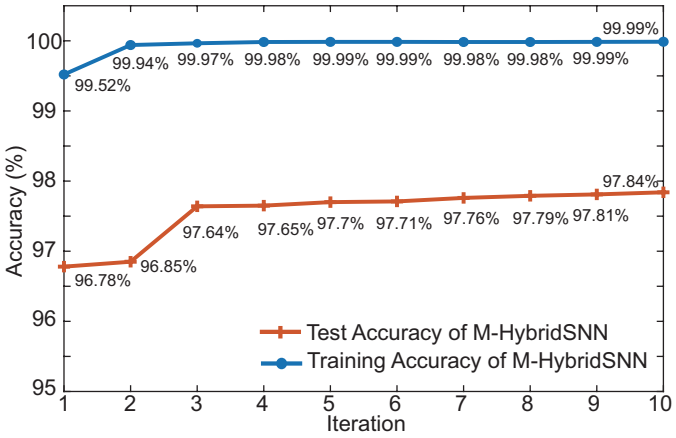


Fig. 6: The classification accuracy of M-HybridSNN on the entire MNIST dataset.

To evaluate the classification capability of M-HybridSNN, we analyze the accuracy changes during the iterations and then compare it with other types of SNN models. Firstly, the improvement of training accuracy and test accuracy during iterations are analyzed for M-HybridSNN. As shown in Fig. 6, it draws the classification accuracy of the M-HybridSNN model with 10 iterations on MNIST dataset. We find that the inference accuracy of M-HybridSNN starts to increase quickly within the first three iterations, then keeps enhancing slowly. It is reasonable because the assembled strong classifier within the first three iterations has improved most of the wrongly-classified samples' learning intensity by enhancing their sample weights. The following iterations could correct the samples that are harder to be categorized than the wrongly-classified samples within the first three iterations stage by stage. Then we compare the results of M-HybridSNN with other models. As illustrated in Table IV, our M-HybridSNN model achieves a test accuracy of 97.84%. The test accuracy of M-HybridSNN exceeds two commonly used supervised learning SNNs, namely BP-STDP [20] and Equilibrium Propagation (EP) [43], which train models with several shallow fully-connected layers directly. However, when compared with

TABLE I: The classification accuracy comparison for M-HybridSNN model with different parameters.

Method	es	Epoch	Train-23	Test-23	Train-58	Test-58	Train-01	Test-01	Train-12	Test-12	Average Accuracy
Mostafa	-	100	99.980%	98.920%	99.992%	98.242%	100.000%	99.811%	99.992%	99.585%	99.565%
M-HybridSNN	1	20	100.000%	99.168%	100.000%	99.089%	100.000%	99.905%	100.000%	99.631%	99.724%
M-HybridSNN	e*ln10	20	100.000%	99.461%	100.000%	99.089%	100.000%	99.905%	99.961%	99.631%	99.756%
M-HybridSNN	(ln10)/t	20	100.000%	99.070%	100.000%	98.875%	100.000%	99.905%	100.000%	99.539%	99.674%
M-HybridSNN	(e*ln10)/t	10	99.942%	99.265%	99.911%	99.196%	100.000%	99.905%	99.961%	99.631%	99.726%
M-HybridSNN	(e*ln10)/t	20	100.000%	99.412%	100.000%	99.196%	100.000%	99.905%	100.000%	99.585%	**99.762%**

TABLE II: The performance of M-HybridSNN on complex sample groups of the MNIST dataset. Depth is the number of cascade layers for the generated tree-like structure by M-HybridSNN.

Method	4&9		5&8		3&8		3&5		2&3		7&9		Average Accuracy	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Mostafa	99.98%	98.14%	99.99%	98.24%	99.98%	**98.84%**	99.99%	98.37%	99.98%	98.92%	**99.98%**	98.13%	99.98%	98.44%
M-HybridSNN	**100%**	**98.29%**	**100%**	**99.20%**	**100%**	98.69%	**100%**	**98.79%**	**100%**	**99.41%**	99.97%	**98.48%**	**99.99%**	**98.81%**
(Accuracy/Depth)	4		3		3		4		3		3		3.33	

TABLE III: The performance of M-HybridSNN on simple sample groups of the MNIST dataset. Depth is the number of cascade layers for the generated tree-like structure by M-HybridSNN.

Method	0&1		1&2		3&4		5&6		7&8		8&9		Average Accuracy	
	Train	Test	Train	Test	Train	Test	Train	Train	Train	Test	Train	Test	Train	Test
Mostafa	**100%**	99.81%	99.99%	**99.59%**	**100%**	99.70%	99.99%	98.65%	**100%**	99.10%	**100%**	98.84%	**100%**	99.28%
M-HybridSNN	**100%**	**99.91%**	**100%**	**99.59%**	**100%**	**99.80%**	**100%**	**99.30%**	**100%**	**99.55%**	**100%**	**98.94%**	**100%**	**99.52%**
(Accuracy/Depth)	2		3		2		3		3		2		2.50	

TABLE IV: The classification accuracies comparison among different algorithms on the entire MNIST dataset.

Method	Network Structure	Test Accuracy
Tavanaei et al. (BP-STDP) [20]	784-500-150-10	97.2 %
O'Connor et al. (EP) [43]	784-500-500-10	97.66 %
Diehl et al. (ANN-SNN Conversion) [23]	784-1200-1200-10	98.64 %
Wu et al. (STBP) [44]	784-800-10	98.89 %
Shrestha et al. (SLAYER) [45]	28x28-12c5-2a-64c5-2a-10o	99.36 %
Mostafa (with one hidden layer) [35]	784-800-10	96.46 % (97.2 %)
Mostafa (with two hidden layers) [35]	784-800-800-10	97.09 %
Our M-HybridSNN	784-800∼800-10	**97.84 %**

the converted Spiking MLP model [23], our model falls behind by about 0.8% in accuracy. It is not surprising that the Spiking MLP, converted from powerful ANN, displays better accuracy. To our best knowledge, there is no directly trained SNN that can outperform a CNN-to-SNN model to date. Nonetheless, the Spiking MLP model requires more resources for conversion, hence lacking overall efficiency compared with our model, which is multi-layer by nature and can be trained directly. In addition, the result of M-HybridSNN falls behind the SLAYER [45] and STBP model [44]. These two models achieve high test accuracy by introducing convolutional layer to enhance the feature extraction or employing iterative LIF neurons to describe timing-dependent temporal domain information. In addition, the M-HybridSNN also improves the classification accuracy of basic weak learner from 96.46%, the score of the original Mostafa SNN [35], to 97.84%. Since there are two cascading layers (784-800∼800-10) in M-HybridSNN after the training process, we construct a Mostafa network with 2 hidden layers for better comparison. As a result, the M-HybridSNN performs better than the new Mostafa network with a similar two-layer structure, since the HybridSNN can preserve more input features efficiently. Although the performance of M-HybridSNN is lower than STBP and SLAYER, it is worth noting that our study is more focused on demonstrating the feasibility and effectiveness of the hybrid idea. Overall, our HybridSNN model performs competitively among the supervised learning SNN models.

E. The Performance of C-HybridSNN with Convolutional SNNs

From the above, the M-HybridSNN achieves the competitive performance on the MNIST dataset. However, the M-HybridSNN could not perform well on CIFAR-10 dataset because of its big scale and data complexity. Hence, the C-HybridSNN model with convolutional SNNs as weak learners is employed to implement the data recognition application for CIFAR-10 dataset in the this section. For C-HybridSNN model, the time step is set to be 100. Here we run multiple trials and compute the average accuracy. In our experiments, the random initializations are the same between C-HybridSNN and the baseline during one trial. That is, they have the same initial network weights at the beginning of training.

As shown in Table V, after 40 training epoches, the test accuracy of the basic deep convolutional VGG9 model is 85.31%. Using the C-HybridSNN framework with seven iterations, on the other hand, can improve the test accuracy to 87.05%. Moreover, with five weak learner and 120 epoches for each one, the C-HybridSNN model can achieve an accuracy of 91.15%, which is slightly better than 90.05% of the original deep convolutional VGG9 model [36], one of the newest competitors. Besides, we compare the test accuracy of C-HybridSNN with other deep convolutional SNNs. The results show that the test accuracy of C-HybridSNN is higher than

TABLE V: The classification accuracies comparison with other method on CIFAR-10 dataset.

Method	Epoch	Test Accuracy
Kim et al. (BNTT) [46]	-	90.5 %
Wu et al. (Tandem Learning) [47]	-	90.98 %
Wu et al. (NeuNorm) [48]	-	90.53 %
Ma et al. (Local TDLL) [49]	-	88.01 %
Park et al. (ANN-SNN conversion) [50]	-	91.4 %
Lee et al. (DCSNN) [36]	40	85.31 %
Lee et al. (DCSNN) [36]	120	90.05 % (90.45 %)
Our C-HybridSNN	40	87.05 %
Our C-HybridSNN	120	**91.15** %

the convolutional SNNs in [46] [47] [48] and the STDP-based spiking networks in [49]. Meanwhile, the test accuracy achieved by C-HybridSNN still could not beat the conversion method from ANN to SNN, such as [50]. Overall, C-HybridSNN achieves quite competitive test accuracy among different convolutional SNNs models on CIFAR-10 dataset. However, the final tree topology of C-HybridSNN has only one cascade layer, which means that this learned model is equivalent to the AdaBoost algorithm using SNN as the weak learner. The reason for the lack of connected nodes in the structure can be ascribed to the information loss during the feature extraction by the basic learner in the deep convolutional VGG9 model, which prevents the output of the prior iteration from being chosen as the input of the current stage.

IV. Discussion

A. The Relationship Between the HybridSNN and Existing Models

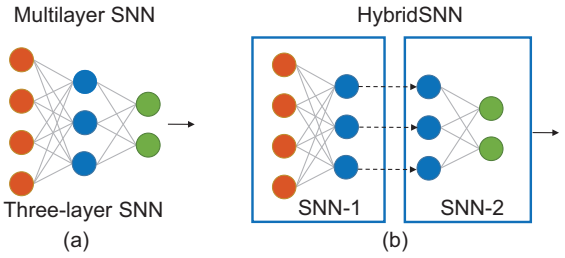


Fig. 7: The comparison between three-layer SNN and Hybrid-SNN.

Our HybridSNN framework uses a flexible network structure rather than the fixed topologies of the existing SNN models. The SNN with fixed structure is merely used as a basic unit in HybridSNN. Hence, if the training has only one cycle, the HybridSNN can be treated as a single-layer SNN.

The initial HybridSNN (Fig. 8 (b)) contains traditional multi-layer neural structures (Fig. 8 (a)) where many connections between neurons are missed. The computation of a traditional multi-layer neural network is described as $X \rightarrow H_1 \rightarrow H_2, ..., Y$, which satisfies:

$$\begin{aligned} H_1 = G(x), H_2 = G(H_1), ... \\ H_{i+1} = G(H_i), ..., Y = G(H_n), \end{aligned} \tag{14}$$

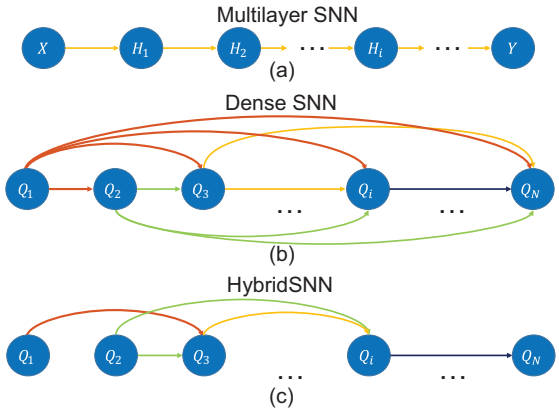


Fig. 8: The comparison between HybridSNN and other multi-layer SNNs. (a) Traditional multi-layer neural network topology; (b) Dense neural network topology (same as the initial state of HybridSNN); (c) Topology of trained HybridSNN.

where G is the operation function between connected layers in the network, such that the connections between layers are unidirectional and fixed. However, whether a connection is preserved or abandoned in HybridSNN is determined by our learning rule, mentioned previously.

The network topology of HybridSNN after training is shown in Fig. 8 (c). The trained topology has a connection density in between those of a traditional multi-layer SNN and a dense neural networks [51]. The training process optimizes the network connections of the HybridSNN model with different weights.

The trained HybridSNN Φ is a linear combination of M classifiers chosen from weak learner pools, which can also be regarded as an ensemble of multi-layer SNNs:

$$\begin{aligned} \Phi \ &= \alpha_1 T^1(X_1) + \alpha_2 T^2(X_2) + ... + \alpha_i T^i(X_i) + ... \\ &\quad + \alpha_N T^M(X_M) \\ &= (\alpha_k T^k(X_k) + ... + \alpha_l T^l(X_l)) + (\alpha_m T^m(X_m) + ... \\ &\quad + \alpha_o T^o(X_o)) + ... + (\alpha_r T^r(X_r) + ... + \alpha_s T^s(X_s)), \end{aligned} \tag{15}$$

where $T^i(X_i)$ is the i_{th} classifier chosen from the i_{th} weak learner pool with a weight score of α_i. The weighted classifiers from $\alpha_k T^k(X_k)$ and $\alpha_l T^l(X_l)$ use raw data as input, which thus constitute the first layer. Similarly, the second cascading layer consists of the weighted classifiers from $\alpha_m T^m(X_m)$ and $\alpha_o T^o(X_o)$, which utilize the output of the first layer as input data. In this way, the cascading layers can be considered as ensembles of multi-layer SNNs and the number of layers represents the depth of HybridSNN.

Fig. 7 illustrates the relationship between HybridSNN and multi-layer SNN wherein an M-HybridSNN with two connected single-layer SNN is displayed in contrast to a three-layer SNN.

Assume that there are N_H and N_O neurons in the hidden and output layers, respectively. In the three-layer SNN, the membrane potential of neuron j in the hidden layer is given as Equation 5. Hence when the firing threshold is set to be 1,

it satisfies Equation 6, and t_j can therefore be simplified as:

$$exp(t_j) = \frac{\sum_{i \in C_j} w_i exp(t_i)}{\sum_{i \in C_j} w_i - 1}. \tag{16}$$

With the exponential transformation in mind, one can denote these spike time variables as in z-domain fashion, as shown in Equation 7. Similarly, the fire time of neuron k in the output layer can be computed as:

$$z_k = \frac{\sum_{j \in C_k} w_j z_j}{\sum_{j \in C_k} w_j - 1}. \tag{17}$$

And the loss function can be given by:

$$C_1(x) = L(g, z_o) = -ln \frac{exp(-z^o[g])}{\sum_k exp(-z^o[k])}. \tag{18}$$

Now consider an M-HybridSNN with two connected weak learners with N input samples. For the first stage, the spike time of neuron j in the output layer can be defined as the same as Equation 7. Hence the loss function is described as:

$$L(g, z_h) = -ln \frac{exp(-z^h[g])}{\sum_j exp(-z^h[j])} * N * W_0^s, \tag{19}$$

where W_0^s is the initial sample weights, being set to be $1/N$. Assume the wrongly classified samples by this weak learner are $U^{(1)} = \{X_{u_1}^1, X_{u_1}^2, ..., X_{u_1}^V\}$, and there are two classes ($K = 2$). Then the weight of this weak learner can be computed according to the AdaBoost algorithm:

$$\alpha^{(1)} \quad = log \frac{1 - error^*(1)}{error^*(1)} = log \frac{N-V}{V},$$

where

$$error^*(1) = \sum_{i=1}^{N} S_i \Gamma(C_i \neq T^{(1)}(X_i)) / \sum_{i=1}^{N} S_i. \tag{20}$$

For the mistaken samples, their weights are updated as:

$$\begin{aligned} S_i = \quad & S_i exp(\alpha^{(1)} \Gamma(C_i \neq T^{(1)}(X_i))) \\ & = \frac{1}{N} exp(log \frac{N-V}{V}) = \frac{N-V}{NV}. \end{aligned} \tag{21}$$

After re-normalizing S_i, the weights of correctly classified samples are $S_i = \frac{\frac{1}{N}}{\frac{1}{N}(N-V) + \frac{N-V}{NV} V} = \frac{1}{2N-2V}$, while the weights of wrongly classified samples are $S_i = \frac{\frac{N-V}{NV}}{\frac{1}{N}(N-V) + \frac{N-V}{NV} V} = \frac{1}{2V}$.

Moreover, for the second iteration of HybridSNN, the spike time of neuron k in the output layer can be described the same as Equation 17. We assume the set of mistaken samples is $U^{(2)} = \{X_{u_2}^1, X_{u_2}^2, ..., X_{u_2}^R\}$, which contains R_1 samples that are recognized correctly by the weak learner of the first iteration but wrongly by that of the second one, and R_2 samples that are recognized wrongly by both weak learners. Hence $error^*(2)$ is given by:

$$error^*(2) = \frac{R_1}{2N - 2V} + \frac{R_2}{2V}. \tag{22}$$

And

$$\alpha^{(2)} = log \frac{1 - (\frac{R_1}{2N-2V} + \frac{R_2}{2V})}{\frac{R_1}{2N-2V} + \frac{R_2}{2V}} = log(\frac{2V(N-V)}{R_1 V + (N-V) R_2} - 1). \tag{23}$$

Thus the loss function of the second-iteration weak learner is:

$$L(g, z_o) = -ln \frac{exp(-z^o[g])}{\sum_k exp(-z^o[k])} * N * W_1^s. \tag{24}$$

And the decision function of HybridSNN is:

$$\begin{aligned} C_2(x) = \quad & log \frac{N-V}{V} * (-ln \frac{exp(-z^h[g])}{\sum_j exp(-z^h[j])}) - \\ & (log \frac{2V(N-V)}{R_1 V + (N-V) R_2} - 1) * ln \frac{exp(-z^o[g])}{\sum_k exp(-z^o[k])}. \end{aligned} \tag{25}$$

Considering the difference between the Equation 18 and Equation 25, we let $log \frac{N-V}{V}$ be 0, then $N = 2V$. Meanwhile, $log \frac{2(N-V)V}{R_1 V + (N-V) R_2} - 1$ is set to be 1, then $V = e^2(R_1 + R_2)/2$. Under these conditions, the two-iteration HybridSNN model has similar transmitted information as a three-layer SNN. Therefore, the multi-layer neural network can be regarded as a specific type of HybridSNN under particular conditions.

In recent years with the upsurge of ANNs, another strategy is used to construct multi-layer SNNs, that is, to train ANNs with deep layers with various types of conversion algorithms which can transfer weights to equivalent deep SNNs [23] [52] [53]. However, these models cannot optimize the networks through temporal spike events during the training process. Moreover, these transforming algorithms are based strictly on layered structures that have exactly the same network topology and encoding mechanism as ANNs. It is our hope that these multi-layer neural networks converted from ANNs to SNNs can be fitted into the HybridSNN framework in the future.

B. Classifier Pruning

One potential pivotal advantage of the SNN-based architecture proposed herein is its high energy efficiency when implemented using the emerging classes of ultra-low-power-consuming spike-based neuromorphic hardware, such as TrueNorth [54], SpiNNaker [55], Tianjic [56]. Based on the theoretical estimation, the power consumption on advanced hardware is conducted to demonstrate the potential energy-efficiency of the proposed HybridSNN systems. In order to make the estimation convincing, we provide a detailed analysis and comparison of energy consumption between HybridSNN on neuromorphic hardware and HybridSNN on various hardware platforms, including CPU, and GPU. The energy estimation adopts a common methodology used in many studies [54] [57] [58] [59], that is, the energy consumption is roughly estimated by multiplying the energy per floating-point operation by the number of operations for CPU and GPU. Similarly, the energy consumption of SNN could also be estimated through multiplying the energy per synaptic operations per second by the time steps and the number of synaptic events. We take the M-HybridSNN model as an example and record the number of synaptic operations per second (SOPS) and floating-point operations per second (FLOPS) to estimate the energy consumption, respectively. The power of FLOPS on CPU and GPU are obtained from Titan V100 [60] and Xeon Platinum 9282 [61], respectively. The energy consumption of SOPS is referred from TrueNorth [54], where one time step is equal to 1ms. As shown in Tables VI and VII, the analytical results show that the proposed HybridSNN implemented on

neuromorphic hardware platform has consistently one to three orders of magnitude higher energy efficiency than it was implemented on other hardware platforms (CPU and GPU). The conclusion is close to the reported results regarding advantage of neuromorphic computing [58] [54]. In summary, the HybridSNN model has the benefits of low-power consumption on neuromorphic hardware.

TABLE VI: The energy estimation of M-HybridSNN model on GPU (Titan V100) [60] and CPU (Xeon Platinum 9282) [61]

Hardware	GFLOPS/W	Energy/Op	FLOPs	Energy (J)
GPU	56	-	6.47 M	1.16E-04
CPU	-	42.9pJ	6.47 M	2.78E-04

TABLE VII: The energy estimation of M-HybridSNN model on TrueNorth neuromorphic chips [54].

Hardware	GSOPS/W	SOPS	Power(W)	Time steps	Energy (J)
TrueNorth	400	6.47 M	1.618E-05	5000	8.09E-05

Except the power consumption, we study the training time and test time for HybridSNN. In our scenario, most of the computation takes place in the classifier selection process indeed. We run M-HybridSNN (using multi-layer SNN as the weak learner) on MNIST dataset to record the model's training time and test time. The running time of M-HybridSNN is measured on a single-thread CPU-E5-2620v4 and a GTX 1080Ti x4. The results show that the training and testing time of M-HybridSNN on the MNIST classification task are 21.08 hr for 60000 samples and 4.54 min for 10000 samples, respectively. As these results illustrate, most of the time is spent in the training stage. Considering the flexibility advantage of the proposed model, we believe such a level of time length on a single core PC for training is not a problem, compared to the training time of some commonly used machine learning algorithms.

The presented HybridSNN framework selects the best-performing SNN units from the weak learner pools and assembles them into a cohesive system. Such aggregation is optimized iteratively in a greedy fashion. The final structure of HybridSNN has a tree-like topology with a medium connection density. Although more parameters usually entail larger network capacity, they can also be superfluous. To find out more, we explore the relationship between the connection density and network performance using the pruning method.

As mentioned earlier, trained HybridSNN has a tree-like topology, whereof each node represents a weighted classifier, and all nodes contribute to the final decision. The denser the network, the more parameters it contains. In order to optimize the density, we introduce a pruning method to the trained HybridSNN and illustrate it in Fig. 9, taking the binary classification of $4\&9$ as an example. Post the training process, all 500 classifier weights are sorted by their values. The bigger the value, the more critical the node, and hence more likely to be kept during pruning. The x-axis denotes the number

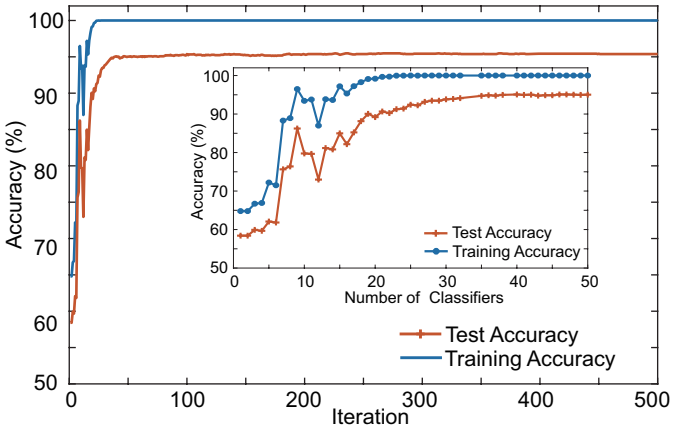


Fig. 9: The visualization for accuracy of weakening Learners. Statistical analysis of the accuracy variation in the classification of $4\&9$ during the weakening process.

of kept classifiers with the biggest weights. During pruning, if the weight of a parent node is smaller than that of the child node, both nodes will be discarded. As we can see, the accuracy increases along with the total weights until the curves flatten. Notably, performance fluctuation occurs when the number of the node is between 20 and 30, which is better visualized in the enlarged subfigure. This may result from the removal of essential nodes. There is thus a trade-off between network sparsity and performance. Such analysis also gives an indication of the optimal number of SNN units to keep for a specific task. The implementations of neuromorphic chips can also benefit from this pruning strategy.

TABLE VIII: The comparison between pruned M-HybridSNN and two-layer Mostafa models.

Model	Paramaters	Test Accuracy
Mostafa's Networks(784-800-800-10)	1275200	97.09%
Prunned M-HybridSNN (784-569-10)	1110688	97.24%

Next, we apply this classifier pruning method to the trained M-HybridSNN network. It turns out that with 569 hidden neurons, the parameters in M-HybridSNN are almost identical to those of the Mostafa model with 784-800-800-10 structure. This includes the case when there is only one layer of classifiers. We then run the M-HybridSNN model with 569 hidden neurons on the MNIST dataset. The depth of the trained topological tree turns out to be 3. The following pruning process appears to preserve the branch connecting the three most weighted nodes, and abandon the rest. As shown in Table VIII, the number of parameters after pruning operation is 784*569+569*10+(569*569+569*10)*2=1110688, smaller than that of the Mostafa model. Meanwhile, the test accuracy after pruning is 97.24%, better than 97.09% of the Mostafa model with 784-800-800-10 structure.

V. Conclusion

This paper proposes an adaptive learning framework, namely HybridSNN, for spiking neural networks. We ensemble the existing single- or multi-layer SNN models into a

deep and strong SNN system in a data-driven manner. HybridSNN combines the benefits of both biologically plausible model and overall data-driven optimization. Inspired by brain mechanisms, HybridSNN could learn the network structure adaptively and provide flexible network topologies to enhance the scalability and improve the computation efficiency for solving various tasks.

The trained tree-like topology is neither too dense nor too sparse, with a structure adaptive to the complexity of different tasks. Unlike the fixed structures of existing artificial neural networks, this framework gathers the output spikes from each weak SNN unit and feeds them back to the pool of classifiers. In order to show the potential of HybridSNN system, experiments are conducted on both MNIST and CIFAR-10 datasets. The results show that the proposed framework achieves competitive performance among supervised learning models of SNN. As a novel SNN model, HybridSNN could not only serve as the basic model in a multi-node cluster system but also plays a potentially powerful role in the multi-core neuromorphic hardware systems in the near future.

REFERENCES

[1] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

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

[3] Z. Hu, G. Pan, Y. Wang, and Z. Wu, "Sparse Principal Component Analysis via Rotation and Truncation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 4, pp. 875–890, Apr. 2016.

[4] Y. Wang, K. Lin, Y. Qi, Q. Lian, S. Feng, Z. Wu, and G. Pan, "Estimating Brain Connectivity With Varying-Length Time Lags Using a Recurrent Neural Network," *IEEE Transactions on Biomedical Engineering*, vol. 65, no. 9, pp. 1953–1963, Sep. 2018.

[5] M. Tsukada and X. Pan, "The Spatiotemporal Learning Rule and Its Efficiency in Separating Spatiotemporal Patterns," *Biological Cybernetics*, vol. 92, no. 2, pp. 139–146, Feb. 2005.

[6] S. Ghosh Dastidar and H. Adeli, "Spiking Neural Networks," *International Journal of Neural Systems*, vol. 19, no. 04, pp. 295–308, Aug. 2009.

[7] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[8] Y. Qi, J. Shen, Y. Wang, H. Tang, H. Yu, Z. Wu, and G. Pan, "Jointly Learning Network Connections and Link Weights in Spiking Neural Networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, Jul. 2018, pp. 1597–1603.

[9] R. Gütig and H. Sompolinsky, "The Tempotron: a Neuron that Learns Spike Timing-Based Decisions," *Nature Neuroscience*, vol. 9, no. 3, pp. 420–428, Mar. 2006.

[10] B. Widrow and M. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, Sep. 1990.

[11] F. Ponulak, "ReSuMe-New Supervised Learning Method for Spiking Neural Networks," *Institute of Control and Information Engineering, Poznoń University of Technology.*, 2005.

[12] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Precise-Spike-Driven Synaptic Plasticity: Learning Hetero-Association of Spatiotemporal Spike Patterns," *PLoS ONE*, vol. 8, no. 11, p. e78318, Nov. 2013.

[13] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike Pattern Association Neuron for Learning Spatio-Temporal Spike Patterns," *International Journal of Neural Systems*, vol. 22, no. 04, p. 1250012, 2012.

[14] T. Masquelier, R. Guyonneau, and S. J. Thorpe, "Competitive STDP-based Spike Pattern Learning," *Neural Computation*, vol. 21, no. 5, pp. 1259–1276, 2009.

[15] I. Sporea and A. Grüningm, "Supervised Learning in Multilayer Spiking Neural Networks," *Neural Computation*, vol. 25, no. 2, pp. 473–509, Feb. 2013.

[16] Q. Yu, H. Tang, J. Hu, and K. C. Tan, "Temporal Learning in Multilayer Spiking Neural Networks Through Construction of Causal Connections," in *Neuromorphic Cognitive Systems: A Learning and Memory Centered Approach*, ser. Intelligent Systems Reference Library, Cham, 2017, pp. 115–129.

[17] Q. Xu, J. Peng, J. Shen, H. Tang, and G. Pan, "Deep CovDenseSNN: A Hierarchical Event-Driven Dynamic Framework with Spiking Neurons in Noisy Environment," *Neural Networks*, vol. 121, pp. 512–519, Jan. 2020.

[18] S. M. Bohte, J. N. Kok, and H. La Poutré, "Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, Oct. 2002.

[19] S. McKennoch, D. Liu, and L. Bushnell, "Fast Modifications of the SpikeProp Algorithm," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, Jul. 2006, pp. 3970–3977.

[20] A. Tavanaei and A. S. Maida, "BP-STDP: Approximating Backpropagation using Spike Timing Dependent Plasticity," *Neurocomputing*, vol. 330, pp. 39–47, Nov. 2019.

[21] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks Using Backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, Nov. 2016.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[23] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer, "Fast-Classifying, High-Accuracy Spiking Deep Networks through Weight and Threshold Balancing," in *2015 International Joint Conference on Neural Networks*, Jul. 2015, pp. 1–8.

[24] P. U. Diehl and M. Cook, "Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity," *Frontiers in Computational Neuroscience*, vol. 9, p. 99, Aug. 2015.

[25] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning," *Frontiers in Neuroscience*, vol. 12, Aug. 2018.

[26] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-Based Spiking Deep Convolutional Neural Networks for Object Recognition," *Neural Networks*, vol. 99, pp. 56–67, Mar. 2018.

[27] C. Lee, G. Srinivasan, P. Panda, and K. Roy, "Deep Spiking Convolutional Neural Network Trained With Unsupervised Spike-Timing-Dependent Plasticity," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 3, pp. 384–394, Sep. 2019.

[28] Q. Fu and H. Dong, "An Ensemble Unsupervised Spiking Neural Network for Objective Recognition," *Neurocomputing*, vol. 419, pp. 47–58, Jan. 2021.

[29] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, "Towards deep learning with segregated dendrites," *Elife*, vol. 6, p. e22901, 2017.

[30] C. W. Lynn and D. S. Bassett, "The Physics of Brain Network Structure, Function and Control," *Nature Reviews Physics*, vol. 1, no. 5, pp. 318–332, 2019.

[31] 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.

[32] Y. Freund and R. E. Schapire, "A Short Introduction to Boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, Sep. 1999.

[33] L. Bachatene, V. Bharmauria, and S. Molotchnikoff, "Adaptation and Neuronal Network in Visual Cortex," *Visual Cortex - Current Status and Perspectives*, Sep. 2012.

[34] M. W. Reimann, M. Nolte, M. Scolamiero, K. Turner, R. Perin, G. Chindemi, P. Dlotko, R. Levi, K. Hess, and H. Markram, "Cliques of Neurons Bound into Cavities Provide a Missing Link between Structure and Function," *Frontiers in Computational Neuroscience*, vol. 11, 2017.

[35] H. Mostafa, "Supervised Learning Based on Temporal Coding in Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, Jul. 2018.

[36] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures," *Frontiers in Neuroscience*, vol. 14, Feb. 2020.

[37] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-Class AdaBoost," *Statistics and Its Interface*, vol. 2, no. 3, pp. 349–360, Jan. 2009.

[38] Q. Xu, Y. Qi, H. Yu, J. Shen, H. Tang, and G. Pan, "CSNN: An Augmented Spiking based Framework with Perceptron-Inception," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, Jul. 2018, pp. 1646–1652.

[39] Q. Xu, J. Shen, X. Ran, H. Tang, G. Pan, and J. K. Liu, "Robust Transcoding Sensory Information with Neural Spikes," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[40] R. E. Schapire and Y. Singer, "Improved Boosting Algorithms Using Confidence-rated Predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[41] J. Friedman, T. Hastie, R. Tibshirani *et al.*, "Additive Logistic Regression: a Statistical View of Boosting (with Discussion and a Rejoinder by the Authors)," *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[42] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1. IEEE, 2001, pp. 511–518.

[43] P. O'Connor, E. Gavves, and M. Welling, "Training a Network of Spiking Neurons with Equilibrium Propagation," *PMLR*, vol. 89, pp. 1516–1523, Apr. 2019.

[44] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks," *Frontiers in Neuroscience*, vol. 12, 2018.

[45] S. B. Shrestha and G. Orchard, "SLAYER: Spike Layer Error Reassignment in Time," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 1412–1421.

[46] Y. Kim and P. Panda, "Revisiting Batch Normalization for Training Low-Latency Deep Spiking Neural Networks from Scratch," *arXiv preprint arXiv:2010.01729*, 2020.

[47] J. Wu, Y. Chua, M. Zhang, G. Li, H. Li, and K. C. Tan, "A Tandem Learning Rule for Effective Training and Rapid Inference of Deep Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[48] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct Training for Spiking Neural Networks: Faster, Larger, Better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 1311–1318.

[49] C. Ma, J. Xu, and Q. Yu, "Temporal Dependent Local Learning for Deep Spiking Neural Networks," in *2021 International Joint Conference on Neural Networks*. IEEE, 2021, pp. 1–7.

[50] S. Park, S. Kim, H. Choe, and S. Yoon, "Fast and Efficient Information Transmission with Burst Spikes in Deep Spiking Neural Networks," in *2019 56th ACM/IEEE Design Automation Conference*. IEEE, 2019, pp. 1–6.

[51] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, Jul. 2017, pp. 2261–2269.

[52] P. O'Connor and M. Welling, "Deep Spiking Networks," *arXiv:1602.08323 [cs]*, Feb. 2016.

[53] 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 in Neuroscience*, vol. 7, p. 178, Oct. 2013.

[54] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A Million Spiking-Neuron Integrated Circuit with a Scalable Communication Network and Interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[55] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, Dec. 2013.

[56] J. Pei and L. e. a. Deng, "Towards Artificial General Intelligence with Hybrid Tianjic Chip Architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, Aug. 2019.

[57] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha *et al.*, "Equivalent-Accuracy Accelerated Neural-Network Training Using Analogue Memory," *Nature*, vol. 558, no. 7708, pp. 60–67, Jun. 2018.

[58] S. Kim, S. Park, B. Na, and S. Yoon, "Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 11 270–11 277, Apr. 2020.

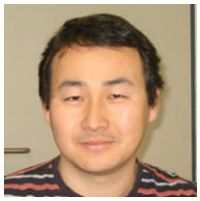
[59] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory Devices and Applications for In-Memory Computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[60] "NVIDIA, T. 2017. V100 gpu architecture."

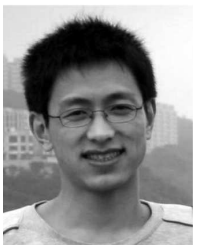
[61] "Intel Corporation, "Intel Xeon Platinum 9282 Processor."."

Jiangrong Shen received her Bachelor's degree in the Department of Computer Science and Technology from Hebei University in 2015, and is currently a Ph.D candidate in the College of Computer Science and Technology, Zhejiang University. Her research interests include neuromorphic computing, computer vision and cyborg intelligence.

Yu Zhao received the B.S. degree in Software Engineering from Zhejiang University of Technology in 2018. He is currently pursuing the M.S. degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include machine learning and computer vision.



Jian K. Liu received the Ph.D. degree in mathematics from the University of California at Los Angeles, Los Angeles, CA, USA, in 2009. He is currently a Lecturer with the School of Computing, University of Leeds, Leeds, U.K. His current research interests include computational neuroscience and brain-like computation.



Yueming Wang received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2007. He was a postdoctoral fellow in the Department of Information Engineering, Chinese University of Hong Kong from 2007 to 2010. He was an associate professor from 2010 to 2016 in the Qiushi Academy for Advanced Studies (QAAS), Zhejiang University, China. Since 2016, he has been a professor in QAAS, Zhejiang University. His current research interests include brain–machine interface, statistical pattern recognition, and neural signal processing.