Neurocomputing 557 (2023) 126686

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Efficient asynchronous federated neuromorphic learning of spiking neural networks

Yuan Wang, Shukai Duan, Feng Chen *

College of Artificial Intelligence, Southwest University, Chongqing, 400715, PR China

ARTICLE INFO

Communicated by F. Perez-Pena

Keywords:
Asynchronous federated learning
Spiking Neural Network
Average spike rate
Model stalenss

ABSTRACT

Spiking Neural Networks (SNNs) can be trained on resource-constrained devices at low computational costs. There has been little attention to training them on a large-scale distributed system like federated learning. Federated Learning (FL) can be exploited to perform collaborative training for higher accuracy, involving multiple resource-constrained devices. In this paper, we introduce SNNs into asynchronous federated learning (AFL), which adapts to the statistical heterogeneity of users and complex communication environments. A novel fusion weight based on information age and average spike rate is designed, which aims to reduce the impact of model staleness. Numerical experiments validate SNNs on federated learning with MNIST, FashionMNIST, CIFAR10 and SVHN benchmarks, achieving better accuracy and desirable convergence under Non-IID settings.

1. Introduction

Spiking Neural Networks (SNNs) are proposed as a promising alternative to Artificial Neural Networks (ANNs) [1], which has gained widespread attention. Compared with ANNs, SNNs achieve approximate performance with higher energy efficiency and lower computational costs [2,3]. However, some training schemes that pre-train artificial neural networks (ANNs) before training SNNs require additional scores of training techniques with high energy consumption [4,5]. To improve training efficiency, many direct supervised learning rules were proposed to train SNNs [6–12], such as Spike Timing Dependent Plasticity (STDP) [13], Spatio-Temporal Back propagation (STBP) [4] and Back propagation Through Time (BPTT) [14], e.g. Recently, some studies have shown that the spike rate of SNNs has a great connection with training accuracy. Due to the binary spike activity of the spike function, current methods are restricted to shallow architectures. Meanwhile, it is difficult to harness large-scale datasets through SNNs for the problem of gradient vanishing. What is more, if the spike rate of neurons is too high, the neurons will be insensitive to the change of input, thus affecting training accuracy and increasing the cost of computation [15–17]. Much work has attempted to balance the spike rate of neurons [18–20] to improve the performance of SNNs.

Federated learning (FL) has played an important role in many fields such as medical and health care, smart grid, etc. [21,22]. As a modern collaborative on-device learning method, FL requires devices to periodically upload their local model parameters and download global model parameters from server to train more effectively [23]. Considering limited radio resource, inevitable transmission failures and heterogeneous computing capacity [24], an asynchronous federated learning (AFL) framework was proposed to improve flexibility and scalability of learning system. Under such circumstances, an AFL system will be in face of the possible delays in training and uploading the local models which cause varying degrees of staleness [25]. Novel fusion algorithms was designed by [26,27] to determine the fusion weight during the global update of AFL systems.

Establishing efficient models on mobile devices plays an important role in FL systems. Recently, the combination of FL and SNNs has been discussed by many scholars. [28] explores for the first time the implementation of collaborative FL for on-device SNNs. [29] design a federated learning framework to train low-power SNNs with BPTT in a distributed and privacy-preserving manner. [30] applies the FL framework with SNNs to distributed traffic sign recognition. The current work simply introduces SNNs into the FL framework to reduce energy consumption on edge devices, without considering the characteristics of SNNs. Device and data heterogeneity have always been a major challenge in FL, and although many works have improved upon FedAvg, such as FedProx [31], FedAsy [25], MOON [32], etc., there has been almost no research proposing solutions for heterogeneity in the FL-SNN scenario.

Through related literature and simulation experiments, we found that the training effectiveness of SNNs is closely related to their average spike rate. For example, if a client has few communication rounds, its network training is incomplete and its average spike rate is low.

* Corresponding author.
E-mail addresses: wangyuan_stu@163.com (Y. Wang), duansk@swu.edu.cn (S. Duan), fengchen.uestc@gmail.com (F. Chen).

https://doi.org/10.1016/j.neucom.2023.126686
Received 11 November 2022; Received in revised form 23 July 2023; Accepted 13 August 2023
Available online 16 August 2023

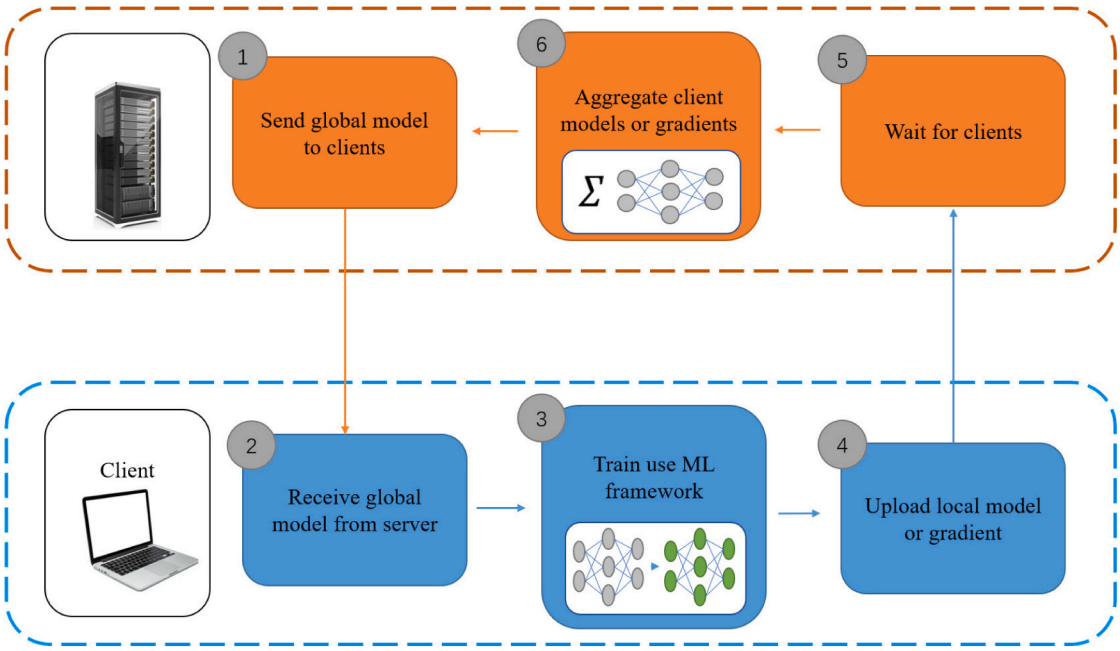

Fig. 1. The process of federated learning. The server works with multiple clients at the same time for training without exchanging raw data.

On the other hand, the difference in average spike rates between different clients' models can be seen as a measure of heterogeneity. To mitigate the poor performance of FL-SNN in Non-IID scenarios, this paper proposes AdaFedAsy-SNN, a federated optimization algorithm based on SNNs that addresses the challenges of statistical heterogeneity both theoretically and empirically. The AdaFedAsy algorithm employs a weight aggregation strategy based on average spike rates and sample sizes and introduces an asynchronous framework.

2. Background

2.1. Conventional federated learning

AFL system consists of N clients and one server. It allows multiple clients to coordinately train a model without exposuring their local data to each other and even the server does not have permission to view the source data. The workflow of FL, as shown in Fig. 1, has four steps:

- The server initializes the global model and distributes the global model development to each participating clients.
- The clients train their model based on the received local data and upload the gradient obtained to the server after training.
- The server aggregates gradients collected from all clients and sends the trained parameters to the clients after updating the global model.
- Repeat steps 2 and 3 until convergence.

Denoting as $f(x,\theta)$ the loss function for any example x when the model parameter vector is θ, the local empirical loss at a device i is defined as:

$$F^{(i)}(\theta) = \frac{1}{|D^{(i)}|}\sum_{x\in D^{(i)}} f(\theta, x), \tag{1}$$

$D^{(i)}$ is the data set owned by the client i, $|D^{(i)}|$ represents the size of the dataset. The global optimization function is:

$$\min_{\theta} F(\theta) = \frac{1}{\sum_{i=1}^{N}|D^{(i)}|}\sum_{i=1}^{N}|D^{(i)}|F^{(i)}(\theta), \tag{2}$$

where $F(\theta)$ is the global empirical loss function based on the dataset $\cup_{i=1}^{N} D^{(i)}$.

At each iteration step: $t = 1,\ldots,T$, each client computes a local stochastic gradient descent:

$$\theta^{(i)}(t) = \theta^{(i)}(t-1) - lr\nabla f(\theta^{(i)}(t-1), x^{(i)}(t)), \tag{3}$$

where $\theta^{(i)}(t-1)$ is the model parameter in the previous iteration step, and $x^{(i)}(t)$ is a part of the dataset $D^{(i)}$. lr is the learning rate.

After receiving all parameters uploaded by clients, the server calculates the centralized averaged parameter:

$$\theta(t) = \frac{1}{\sum_{i=1}^{N}|D^{(i)}|}\sum_{i=1}^{N}|D^{(i)}|\theta^{(i)}(t). \tag{4}$$

2.2. Asynchronous federated learning

Synchronous federated learning requires the server to wait for all clients to upload information before aggregation, which may lead to too long a training time and increase the overhead of communication resources. There are many factors that cause asynchronous learning: different times to join learning, insufficient computing power of nodes, gradient compression, and learning interruption caused by various external factors. Under the asynchronous federated learning framework, once the server receives information uploaded by one or a batch of clients, it aggregates it without waiting for all clients to upload information. The training takes T global epochs. Compare to Eq. (4), in the t^{th} epoch, the server receives a locally trained model $\theta^{(i)}(k)$ from a client and updates the global model θ_{t-1} by weighted averaging [25]:

$$\theta_t = \gamma\theta^{(i)}(k) + (1-\gamma)\theta_{t-1}. \tag{5}$$

Where $\gamma \in (0,1)$.

However, there is a problem of gradient staleness under the asynchronous federated learning architecture. Gradient staleness can seriously affect training accuracy. Researchers use the gap in timestamp between the global parameters $\theta(t)$ used by the clients for this training epoch and the global parameters $\theta(t+\ell)$ to be used in the next training epoch to measure the staleness of the gradient.

2.3. Spiking neural network

The structure of a SNN is an arbitrary directed graph connected N spiking neurons. Through accumulating the incoming spikes scaled

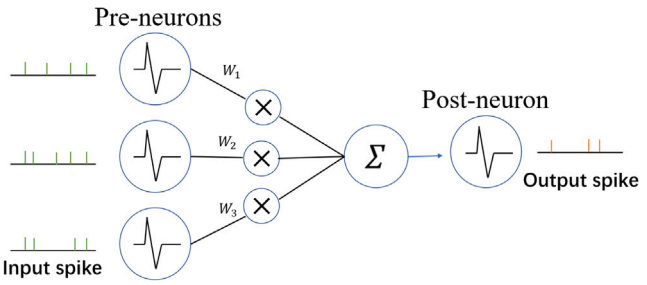

Fig. 2. Spike neuron propagation process.

by their corresponding synaptic weights, spike neuron generates a spike and reset its the membrane voltage when its membrane potential exceeds threshold. Therefore, Inappropriate threshold settings can lead to insufficient or over-activation of neurons. Common models of spiking neuron include the Leaky-Integrate-and-Fire (LIF) model and the Integrate-and-Fire (IF) model (see Fig. 2).

SNN neurons are Markov: the output at the current moment is only related to the input at the current moment and the state of the neuron itself. Any discrete spiking neuron can be described by three discrete equations: charge, discharge, and reset:

$$\begin{aligned} H(t) &= f(V(t-1), X(t)) \\ S(t) &= g(H(t) - V_{threshold}) = \Theta(H(t) - V_{threshold}) \\ V(t) &= H(t) \cdot (1 - S(t)) + V_{reset} \cdot S(t) \end{aligned} \tag{6}$$

where $V(t)$ is the membrane voltage of the neuron, $X(t)$ is the input, $H(t)$ is the hidden state of the neuron, $S(t)$ is the impulse fired by the neuron, and $g(x)$ is the impulse function:

$$g(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}, \tag{7}$$

LIF is the most commonly used model at present to describe the neuronal dynamics in SNNs, and it can be simply governed by:

$$\tau \frac{du(t)}{dt} = -u(t) + I(t), \tag{8}$$

where $u(t)$ is the neuronal membrane potential at time t, τ is a time constant and $I(t)$ denotes the pre-synaptic input. When the membrane potential u exceeds a given threshold v_th, the neuron generates a spike and resets its membrane voltage to u_{reset}.

The membrane potential u_i^t of neuron i at time step t can be defined as:

$$u_i^t = lu_i^{t-1} + \sum_j w_{ij} o_j^t, \tag{9}$$

where $l \in (0, 1)$ is the leakage factor, j represents the number of neurons in the previous layer, w_{ij} represents the weight of neuron i's connection to neuron j in the previous layer. o_j^t can be defined as:

$$o_i^t = \begin{cases} 0, u_i^t > v\_th \\ 1, otherwise \end{cases}, \tag{10}$$

for model updates, SNNs can update the weights of all layers similar to a recurrent neural network (RNN) through gradient descent [33]. The gradient of the lth layer can be defined as:

$$\Delta w_{ij} = \sum_{T=1}^{T} \frac{\partial L}{\partial w_{ij}^t} = \sum_{t=1}^{T} \frac{\partial L}{\partial o_i^t} \frac{\partial o_i^t}{\partial u_i^t} \frac{\partial u_i^t}{\partial w_{ij}^t}, \tag{11}$$

where L is the loss function.

To overcome the problem about nondifferentiable of the impulse function, the surrogate gradient descent method can be used as an approximation which is defined as [33]:

$$\frac{\partial o_i^t}{\partial u_i^t} = \varphi \max\{0, 1 - |\frac{u_i^t - v\_th}{v\_th}|\}. \tag{12}$$

Fig. 3. Example of Poisson encoder. Encode the image, simulate different timesteps. As the timestep increases, the superposition result of the spike matrix becomes closer to the original image.

Where φ is a decay factor for back-propagated gradients and v_th is the threshold value.

2.4. Training spiking neural network

BPTT was recently proposed to improve the SNNs training by leveraging batch normalization in temporal dimension [34], and the formula of BPTT can be defined as:

$$\begin{aligned} u_i^t &= lu_i^{t-1} + BPTT_{\lambda^t}(\sum_{j \in N} w_{ij} o_j^t) \\ &= lu_i^{t-1} + \lambda_i^t(\frac{\sum_{j \in N} w_{ij} o_j^t - \mu_i^t}{\sqrt{(\sigma_i^t)^2 + \varepsilon}}). \end{aligned} \tag{13}$$

Where μ_i^t and σ_i^t are the mean and variance from the samples in a batch b at time step t. λ_i^t is the learned weight of the BN layer, ε is a constant that prevents the denominator from being 0.

3. Asynchronous federated neuromorphic learning of spiking neural networks

Given a server and a set of N clients with each client i having its own private local dataset $D^{(i)}$. Before starting, the server sends a global model to clients. Note that different clients have different local datasets: $D^{(i)} \neq D^{(j)}, \forall i \neq j$.

As described in Algorithm 1, training thread starts by encoding the pixel values into spike trains of length T using Poisson rate coding(see Fig. 3). The Poisson encoder is a stateless encoder that converts the input data x into a spike with the same shape, conforming to a Poisson process, where the number of spikes during a certain period follows a Poisson distribution. A Poisson process is also known as a Poisson flow. A spike flow is a Poisson flow if it satisfies the requirements of independent increment, incremental stability, and commonality. Specifically, the number of spikes appearing in disjointed intervals in the entire spike stream is independent of each other, and in any interval, the number of spikes is related to the length of the interval but not the starting point of the interval.

The neuron layer has some construction parameters: tau, v_th, v_reset. tau is the membrane potential time constant, while v_th is neuron threshold voltage. v_reset represents the reset voltage of the neuron.

After completing the training, client i needs to upload the gradient information θ_k^i, its time stamp $(t_{upload})_k^i$, number of samples num_i and average spike rate of neurons $sr_{i,k}$. k is the training epoch of the clients.

We design aggregated weights based on sample size, neuron average spike rate, and information age. To begin with, clients with larger datasets should contribute more to global training. The sample weight is defined as the ratio of the number of samples held by client i to the number of samples held by all clients. The sample weights are designed as follows:

$$\psi_i = \frac{\left|D^{(i)}\right|}{\cup_{i=1}^{N} \left|D^{(i)}\right|}, \tag{14}$$

meanwhile, gradients with greater staleness should contribute less to the globally trained model, as the staleness of gradients can seriously

affect the accuracy of model training. Information age is defined as the difference between the time when the client uploads the gradient and that the server sends the global gradient. When the server/client finally received the model, this model was probably from several iterations ago. Traditional aggregation methods like FedAvg will result in a degradation of model performance [35]. The information age is designed as follows:

$$I_t^i = (t_{upload})_t^i - (t_{send})_t^i, \tag{15}$$

Then the information age weight can be defined as:

$$\beta_t^i = \frac{1}{1+\exp(I_t^i)}, \tag{16}$$

last but not least, when the average spike rate is low, few spikes and neuronal membrane potentials exist, and the information processed by neurons is not sufficiently expressed [36]. In contrast, when neurons spike too frequently, the network is insensitive to changes in the input with unnecessary energy expenditure.

We measure the average spike rate as defined by the following equation [19]:

$$sr = \frac{total\ spike}{total\ neurons \times timesteps}, \tag{17}$$

Poisson distributed spike trains were generated for each image pixel with spike rates proportional to the pixel's intensity value [37]. In the real scene, the spike rate of SNNs fluctuates due to external interference, which affects the performance. According to Table 1, we try to find the connection between spike rate and test accuracy through regulating v_th. The weights based on spike rate are designed as follows:

$$\zeta_t^i = \frac{1}{\sqrt{2\pi}\sigma_t}\exp(-\frac{(sr_{i,k}-\mu_t)^2}{2{\sigma_t}^2}), \tag{18}$$

where μ_t and δ_t are adaptive parameters. In summary, the weights for server aggregation are as follows:

$$\lambda_t = \kappa \times \beta_t^i \times \psi_i \times \zeta_t^i, \tag{19}$$

where κ are hyperparameters. After getting the weights, we can get new global model parameters:

$$\theta(t) = (1-\lambda_t)\theta(t-1) + \lambda_t\theta_i^k. \tag{20}$$

After completing the aggregation, the server transfers the global model parameters or gradient to the idle clients.

4. Convergence analysis

Note that Eq. (20) is used to formulate the global model. we give a more general form as

$$\theta^t = \theta^{t-1} - \eta^t h(\theta^{t-1}, \xi^t) = \theta^{t-1} - \lambda_k(\theta^{t-1} - \theta_i^k) \tag{21}$$

where $h(\cdot)$ is the aggregation function while ξ^t describes the parameters uploaded by clients and some information such as ID. For convenience, $h(\theta^{t-1})$ is used to represent $h(\theta^{t-1}, \xi^t)$.

Most previous work analyzes synchronous federated learning methods [38–40]. There has been a lot of new work recently analyzing the convergence of asynchronous federated learning methods [41,42]. We make the following assumptions about the objective functions:

Assumption 1 (*Smoothness*). For $i = 1, 2, \ldots, N$, give constant β, the objective function $F(\theta)$ and $F_i(\theta)$ are $\beta - smooth$:

$$\left\|\nabla F(\theta) - \nabla F(\theta')\right\| \le \beta\left\|\theta - \theta'\right\|,$$
$$\left\|\nabla F_i(\theta) - \nabla F_i(\theta')\right\| \le \beta\left\|\theta - \theta'\right\|.$$

Assumption 2 (*The First and Second Moment Conditions*). Function $F(\theta)$ and $h(\theta)$ satisfy:

(a) $\left\|\nabla F_i(\theta)\right\|^2 \le G$

(b) When exits scalars $\varsigma_G > \varsigma > 0$, $\nabla F(\theta^t)^T \mathbb{E}(h(\theta^t)) \ge \varsigma\left\|\nabla F(\theta^t)\right\|^2$

(c) There exits scalars L satisfy

$$\mathbb{V}(h(\theta^t)) = \mathbb{E}(\left\|h(\theta^t)^2\right\|) - \mathbb{E}(\left\|h(\theta^t)\right\|^2) \le L + \left\|\nabla F(\theta^t)\right\|^2.$$

Algorithm 1 Asynchronous Federated SNN

Require:
Set of N clients with Local Datasets $D^{(i)}, i = 1, 2, ..., N$
Number of local epochs: e, Number of Timesteps: T, Number of rounds: R, learning rate: l
Two dimensional array $S_{N\times R}$
Ensure: Final model parameters θ_F

Process Server():
1. Initialize $\theta(0)$
2. Run Scheduler() thread and Updater() thread asynchronously in parallel

Thread Scheduler():
Broadcast the current parameter $\theta(i)$ to clients which have completed training

Thread Updater():
for epoch $t \in [R]$ **do**
Receive the pair $(\theta_t^i, t^i_{download}, num_i, sr_{i,t})$
Save data into cache while cache is not full
Use Algorithm 2 to update global model
Clear the cache
end

Process Client():
1. Receive parameter from Sever and its time stamp $(\theta(t), (t_{send})_t^i)$
2. Run Train() thread
3. Push the pair $(\theta_i^t, (t_{upload})_t^i, num_i, sr_{i,t})$ to Server

Thread Train():
for $t \in [T]$ **do**
$O \leftarrow PoissonGenerator()$
for $Layer \quad l \leftarrow 1 \quad to \quad L-1$ **do**
Forward propagate
end
end
Accumulate membrane potential at the last layer
Compute Loss and Backpropagate the model gradient

Algorithm 2 Adaptive Fusion Algorithm

Require:
Two dimensional array $S_{N\times R}$
Set of clients in Epoch t:$N_t \in N$
Set of pair: $\{p_{i,k} : k \in N_t, i \in N\}$
$p_{i,k} = (\theta_t^i, t^i_{download}, num_i, sr_t^i)$
Ensure: model parameters θ_t
1. Calculate the median of $\{sr_{i,k} : k \in N_t, i \in N\} : sr_t$
2. Set $S_{i,t} = sr_t$
3. Calculate σ_t and μ_t of $S_{i,*}$
4. Calculate the fusion weight use Equation (14)-(19)
5. Update global model use Equation (20)

Table 1
Spike rate on MNIST and FashionMNIST dataset with different v_th.

Dataset	v_th	sr	Test accuracy
MNIST	0.4	0.0336	0.96
	0.6	0.0315	0.97
	0.8	0.0307	0.97
	1.0	0.0285	0.95
	1.2	0.0271	0.89
	1.4	0.0225	0.96
	1.6	0.0000	0.10
FashionMNIST	0.4	0.0331	0.84
	0.6	0.0317	0.82
	0.8	0.0313	0.81
	1.0	0.0299	0.88
	1.2	0.0268	0.79
	1.4	0.0232	0.76
	1.6	0.0000	0.10

Assumption 3 (*Strong Convexity*)**.** For $i = 1, 2, \ldots, N$ and given constant c, function $F(\theta)$ and $F_i(\theta)$ are c-strong convex:

$$F(\theta') \geq F(\theta) + \nabla F(\theta)^T(\theta' - \theta) + \frac{1}{2}c\|\theta' - \theta\|^2$$
$$F_i(\theta') \geq F(\theta) + \nabla F_i(\theta)^T(\theta' - \theta) + \frac{1}{2}c\|\theta' - \theta\|^2$$

Theorem 1. *When c and β in Assumptions 1 and 3 satisfy $c \leq \beta$, we can set step size $\eta^t = \bar{\eta}$, where $0 < \bar{\eta} < \frac{\varsigma}{\beta(1+\varsigma_G^2)}$. the upper error bound of global model satisfies:*

$$\begin{aligned}(F(\theta^t) - F(\theta^*)) \leq & \frac{\bar{\eta}\beta L}{2c\varsigma} \\ & + (1 - \bar{\eta}c\varsigma)^{t-1}(F(\theta^1) - F(\theta^*) - \frac{\bar{\eta}\beta L}{2c\varsigma}).\end{aligned} \tag{22}$$

Proof of Theorem 1. Under Assumption 1. For any θ and θ', we have:

$$\begin{aligned}F(\theta) &= F(\theta') + \int_0^1 \frac{dF(\theta' + t(\theta - \theta'))}{dt}dt \\ &= F(\theta') + \int_0^1 \nabla dF(\theta' + t(\theta - \theta'))^T(\theta - \theta')dt \\ &= F(\theta') + \nabla F(\theta')^T(\theta - \theta') \\ &+ \int_0^1 \nabla dF(\theta' + t(\theta - \theta') - \nabla F(\theta'))^T(\theta - \theta')dt \\ &\leq F(\theta') + \nabla F(\theta')^T(\theta - \theta') \\ &+ \int_0^1 \beta \|t(\theta - \theta')\| \|\theta - \theta'\| dt \\ &= F(\theta') + \nabla F(\theta')^T(\theta - \theta') + \frac{1}{2}\beta\|\theta - \theta'\|^2,\end{aligned} \tag{23}$$

according to Eq. (18):

$$\begin{aligned}F(\theta^{t+1}) - F(\theta^t) &\leq \nabla F(\theta^t)^T(\theta^{t+1} - \theta^t) + \frac{1}{2}\beta\left\|\theta^{t+1} - \theta^t\right\|^2 \\ &= \nabla F(\theta^t)^T\eta^t h(\theta^t) + \frac{1}{2}\beta\left\|\eta^t h(\theta^t)\right\|^2,\end{aligned} \tag{24}$$

then we have:

$$\begin{aligned}\mathbb{E}(F(\theta^{t+1})) - F(\theta^t) \leq & -\eta^t \nabla F(\theta^t) \left\|\mathbb{E}(h(\theta^t))\right\| \\ & + \frac{1}{2}\eta^t \mathbb{E}(\left\|h(\theta^t)\right\|^2),\end{aligned} \tag{25}$$

using Assumption 2(b) and (c), we can get:

$$\begin{aligned}\mathbb{E}\left(\left\|h(\theta^t)\right\|^2\right) &\leq \left\|\mathbb{E}(h(\theta^t))\right\|^2 + L + \left\|\nabla F(\theta^t)\right\|^2 \\ &\leq L + (1 + \varsigma_G^2)\left\|\nabla F(\theta^t)\right\|^2 \\ &= L + L_G\left\|\nabla F(\theta^t)\right\|^2,\end{aligned} \tag{26}$$

Table 2
Parameters.

Notation	Description	Value
e	Local epoch	1
R	Global epoch(rounds)	100
l	Learning rate	0.001
T	Timesteps	10
tau	Membrane time constant	2.0
N	Number of clients	10
b	Data batch size	512
bs	Aggregation batch size	5
v_th	Neuron threshold voltage	1.0
v_reset	Reset voltage of the neuron	0.0

Table 3
Datasets.

Dataset	Sample	Class number	Image size
MNIST	70 000	10	28×28
FashionMNIST	70 000	10	28×28
CIFAR10	60 000	10	$32 \times 32 \times 3$
SVHN	99 289	10	$32 \times 32 \times 3$

using Eqs. (20) and (21), The following Equation can be obtained:

$$\begin{aligned}\mathbb{E}(F(\theta^{t+1})) - F(\theta^t) \leq & -\eta^t\varsigma\left\|\nabla F(\theta^t)\right\|^2 \\ & + \frac{1}{2}(\eta^t)^2\beta(L + L_G\left\|\nabla F(\theta^t)\right\|^2),\end{aligned} \tag{27}$$

we can define:

$$\Gamma(\hat{\theta}) = F(\theta^t) + \nabla F(\theta^t)^T(\hat{\theta} - \theta^t) + \frac{1}{2}c\left\|\hat{\theta} - \theta^t\right\|^2, \tag{28}$$

when $\hat{\theta} = \theta^t - \frac{\nabla F(\theta^t)}{c}$, Eq. (23) get the minimum value:

$$\Gamma_{\min} = F(\theta^t) - \frac{\left\|\nabla F(\theta^t)\right\|^2}{2c}, \tag{29}$$

according to Assumption 3, we can get:

$$2c(F(\theta^t) - F(\theta^*)) \leq \left\|\nabla F(\theta^t)\right\|^2, \tag{30}$$

then we have:

$$\mathbb{E}(F(\theta^{t+1}) - F(\theta^*)) \leq (1 - \eta^t\varsigma c)\mathbb{E}(F(\theta^t) - F(\theta^*)) + \frac{1}{2}(\eta^t)^2\beta L, \tag{31}$$

through simple calculation, we can get:

$$\begin{aligned}\mathbb{E}(F(\theta^{t+1}) - F(\theta^*)) - \frac{\bar{\eta}\beta L}{2c\varsigma} \leq & (1 - \eta^t\varsigma c)\mathbb{E}(F(\theta^{t+1}) - F(\theta^*) \\ & - \frac{\bar{\eta}\beta L}{2c\varsigma}).\end{aligned} \tag{32}$$

As shown above, we have completed the proof.

5. Experiments

The experiment is based on the SpikeJelly and FedLab platform with an NVIDIA GeForce RTX 3070 GPU with 8 GB of memory for all experiments.

5.1. Experiment setup

(a) Dataset under IID and Non-IID settings: In each round, half clients are randomly set to participate in the training. (i) IID — the data distribution is the same among all participants, meaning that each participant's dataset contains the same types and amounts of data. For example, the MNIST dataset is evenly partitioned among 10 clients in an IID manner, where each client has 10 types of data and the number of data samples for each type is 600; (ii) Non-IID - Divide the dataset

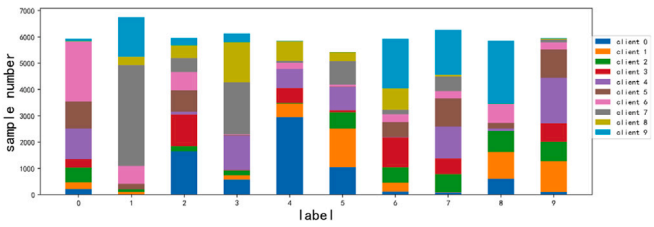

Fig. 4. Example of Non-IID settings on MNIST dataset with $\alpha = 0.5$.

Table 4
Structures of MNISTNet.

Network	Layer	Input	Output
MNISTNet	Conv1	1 × 28 × 28	16 × 28 × 28
	BN1	16 × 28 × 28	16 × 28 × 28
	SN1	16 × 28 × 28	16 × 28 × 28
	Maxpool1	16 × 28 × 28	16 × 14 × 14
	Conv2	16 × 14 × 14	16 × 14 × 14
	BN2	16 × 14 × 14	16 × 14 × 14
	SN2	16 × 14 × 14	16 × 14 × 14
	Maxpool2	16 × 14 × 14	16 × 7 × 7
	DP3	16 × 7 × 7	16 × 7 × 7
	FC3	16 × 7 × 7	2048
	SN3	2048	2048
	DP4	2048	2048
	FC4	2048	100
	SN4	100	100
	Voting	100	10

into several sections, each client contains different types and amounts of data. We use dirichlet distribution to divide datasets (see Table 3):

$$P(X;\alpha m) \propto \prod_{i=1}^{k} x_i^{\alpha m_i - 1}, \sum_{i=1}^{k} m_i = 1, m_i > 0$$
$$X = (x_1, x_2, \ldots, x_k), x_i > 0, \sum_{i=1}^{k} x_i = 1 \quad (33)$$

Where α measures the sharpness of this distribution(see Fig. 4).The smaller the value, the stronger the heterogeneity of different clients.

(b) Model architecture: MNISTNet is a spiking neural network model provided by SpikingJelly for the task of MNIST handwritten digit recognition. The input of this model is a grayscale image of size 28 × 28, and the convolutional kernel size is 3 × 3. The output is a number from 0 to 9. MNISTNet is trained using the backpropagation algorithm with the cross-entropy loss function. The final output is determined by comparing the activation levels of different neuron units corresponding to each digit.

As shown in Table 4, the model structure of the MNISTNet used in our experiment consists of two convolution layers, two BN layers, two max pooling layers, two dropout layers and two fully connected layers. The network architecture of FashionMNISTNet is the same as that of MNISTNet, whereas the structure of CIFAR10Net is similar to that of MNISTNet but with six Conv layers. SpikingResnet18 is a SNN model provided by Spikejelly, which is based on the Resnet18 architecture and can be used for image classification tasks. Similar to its non-spiking counterpart, SpikingResnet18 consists of multiple residual blocks and utilizes skip connections to mitigate the vanishing gradient problem.

(c) Training parameters: According to Table 2, we evaluate models with $T = 10$, using the Adam optimizer. Reducing the simulation timestep T can help to reduce the number of input spikes per timestep while large timestep will help to avoid insufficient activation [5]. With the increase of T, the accuracy of training will be improved, but at the same time, it will also increase the cost of training.

Table 5
Performance of AdaFedAsy-SNN, FedAvg-SNN, FedAsy-SNN, FedProx-SNN and MOON-SNN under Non-IID settings.

Dataset	Model	Rounds	α	Method	Test accuracy
MNIST	MNISTNet	100	0.5	**Ours**	**98.21%**
				FedAvg-SNN	96.89%
				FedAsy-SNN	97.73%
				FedProx-SNN	96.92%
				MOON-SNN	97.73%
FashionMNIST	FashionMNISTNet	100	0.5	**Ours**	**83.70%**
				FedAvg-SNN	78.76%
				FedAsy-SNN	79.58%
				FedProx-SNN	80.25%
				MOON-SNN	79.63%
CIFAR10	CIFAR10Net	100	0.5	**Ours**	**38.94%**
				FedAvg-SNN	35.36%
				FedAsy-SNN	33.72%
				FedProx-SNN	36.52%
				MOON-SNN	37.72%
SVHN	Spiking-ResNet18	200	0.5	**Ours**	**70.25%**
				FedAvg-SNN	52.70%
				FedAsy-SNN	52.74%
				FedProx-SNN	56.21%
				MOON-SNN	59.08%

5.2. Experiment result

To verify the effectiveness of the AdaFedAsy-SNN algorithm, this paper introduces the FedAsy-SNN, FedProx-SNN, and MOON-SNN algorithms as comparative algorithms. FedProx adds a regularization term to the global model's loss function so that each client can better adapt its local model training to its local data distribution. Different from traditional federated learning methods, the MOON algorithm not only focuses on the performance of the global model, but also on the similarity and difference between models on different devices. Specifically, MOON uses contrastive learning techniques to form pairs of models on devices and tries to minimize the differences between models within the same pair, while maximizing the differences between different pairs. This approach helps improve the generalization ability of the model and enhances its robustness against various attacks.

From Table 5, it can be observed that as the complexity of the dataset increases, compared to the FedAsy-SNN algorithm, our algorithm achieved a lead of 0.48%, 4.12%, 1.22%, and 17.51% on the MNIST, FashionMNIST, CIFAR10, and SVHN datasets, respectively. Meanwhile, compared with FedProx-SNN and MOON-SNN algorithms, the proposed algorithm in this paper also shows significant improvement.

From Fig. 5, the AdaFedAsy-SNN algorithm has the fastest convergence rate, which is because the average spike rate of networks with fewer global communication rounds is low. AdaFedAsy-SNN gives these clients lower weights when aggregating on the server, while those with better communication conditions and stronger computational power have a greater impact on the global model. Of course, this paper also takes into account the sample size of clients to amplify the influence of clients with larger data volume.

Conclusion

The AFL system based on SNNs was investigated in this paper. We experimentally evaluate the performance on datasets CIFAR10, MNIST, FashionMNIST and SVHN. What is more, a new aggregation strategy was designed according to the characteristics of SNNs, which can address data heterogeneity, improving the training accuracy. SNNs are a viable alternative to ANNs in large-scale federated learning owing to their energy efficiency with the help of FL.

In conclusion, this work achieves better performance than synchronization algorithm, enabling SNNs to solve more complex large-scale

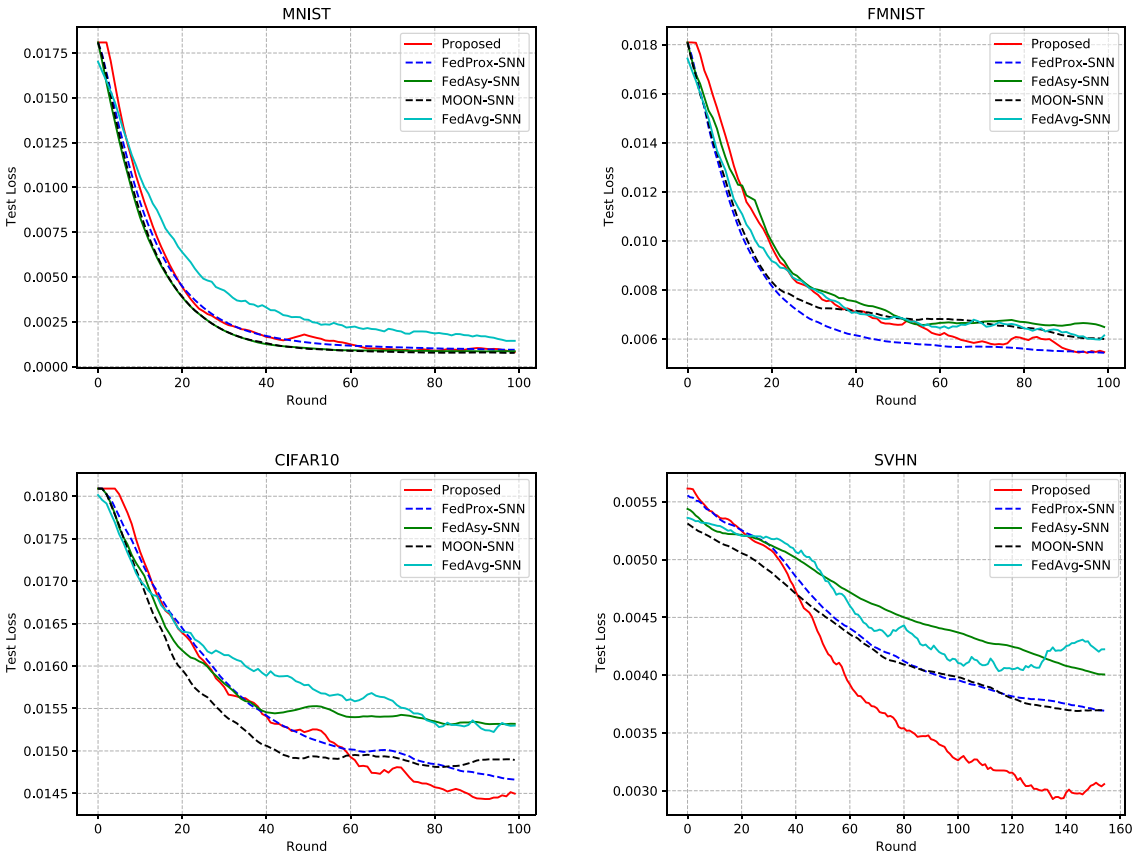

Fig. 5. Performance under Non-IID scenario.

problems through the gradient sharing mechanism, which is beneficial to the implementation on neuromorphic hardware and promotes the practical application of SNNs.

CRediT authorship contribution statement

Yuan Wang: Methodology, Validation, Software, Writing – original draft. **Shukai Duan:** Formal analysis, Investigation. **Feng Chen:** Conceptualization, Investigation, Supervision.

Declaration of competing interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1306600 and Grant 2018YFB1306604.

References

[1] K. Roy, A. Jaiswal, P. Panda, Towards spike-based machine intelligence with neuromorphic computing, Nature 575 (7784) (2019) 607–617.
[2] H. Jang, O. Simeone, B. Gardner, et al., An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications, IEEE Signal Process. Mag. 36 (6) (2019) 64–77.
[3] A. Sengupta, Y. Ye, R. Wang, C. Liu, K. Roy, Going deeper in spiking neural networks: Vgg and residual architectures, Front. Neurosci. 13 (2019) 95.
[4] Y. Wu, L. Deng, G. Li, et al., Spatio-temporal backpropagation for training high-performance spiking neural networks, Front. Neurosci. 12 (2018) 331.
[5] P.U. Diehl, D. Neil, J. Binas, M. Cook, S.C. Liu, M. Pfeiffer, Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing, in: 2015 International Joint Conference on Neural Networks, IJCNN, IEEE, 2015, pp. 1–8.
[6] S.R. Kheradpisheh, M. Mirsadeghi, T. Masquelier, BS4NN: Binarized spiking neural networks with temporal coding and learning, Neural Process. Lett. 54 (2022) 1255–1273, http://dx.doi.org/10.1007/s11063-021-10680-x.
[7] H. Mostafa, Supervised learning based on temporal coding in spiking neural networks, IEEE Trans. Neural Netw. Learn. Syst. 29 (7) (2018) 3227–3235, http://dx.doi.org/10.1109/TNNLS.2017.2726060.
[8] C. Lee, et al., Enabling spike-based backpropagation for training deep neural network architectures, Front. Neurosci. 14 (2020) 119.
[9] L. Jeongjun, et al., Spike-train level direct feedback alignment: Sidestepping backpropagation for on-chip training of spiking neural nets, Front. Neurosci. 14 (143) (2020).
[10] A. Tavanaei, A. Maida, BP-STDP: Approximating backpropagation using spike timing dependent plasticity, Neurocomputing 330 (2019) 39–47.
[11] C. Shi, et al., DeepTempo: A hardware-friendly direct feedback alignment multi-layer tempotron learning rule for deep spiking neural networks, IEEE Trans. Circuits Syst. II 68 (5) (2021) 1581–1585.
[12] H. Wang, et al., TripleBrain: A compact neuromorphic hardware core with fast on-chip self-organizing and reinforcement spike-timing dependent plasticity, IEEE Trans. Biomed. Circuits Syst. 16 (4) (2022) 636–650, http://dx.doi.org/10.1109/TBCAS.2022.3189240.
[13] P.U. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity, Front. Comput. Neurosci. 9 (2015) 99.
[14] Y. Kim, P. Panda, Revisiting batch normalization for training low-latency deep spiking neural networks from scratch, Front. Neurosci. (2020) 1638.

[15] Ruizhi Chen, et al., Low latency spiking ConvNets with restricted output training and false spike inhibition, in: 2018 International Joint Conference on Neural Networks, IJCNN, IEEE, 2018.
[16] Hanle Zheng, et al., Going deeper with directly-trained larger spiking neural networks, 2020, arXiv preprint arXiv:2011.05280.
[17] Seijoon Kim, et al., Spiking-yolo: spiking neural network for energy-efficient object detection, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, 2020, No. 07.
[18] N. Rathi, K. Roy, Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization, IEEE Trans. Neural Netw. Learn. Syst. (2021).
[19] B. Han, G. Srinivasan, K. Roy, Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 13558–13567.
[20] B. Rueckauer, I.A. Lungu, Y. Hu, et al., Conversion of continuous-valued deep networks to efficient event-driven networks for image classification, Front. Neurosci. 11 (2017) 682.
[21] Q. Yang, Y. Liu, T. Chen, et al., Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) 1–19.
[22] P. Kairouz, H.B. McMahan, B. Avent, et al., Advances and open problems in federated learning, Found. Trends® Mach. Learn. 14 (1–2) (2021) 1–210.
[23] H.B. McMahan, E. Moore, D. Ramage, S. Hampson, B. Aguera y Arcas, Communication-effificient learning of deep networks from decentralized data, in: Proc. Int. Conf. Artifificial Intelligence and Statistics, Fort Lauderdale, Florida, 2017.
[24] Z. Wang, Z. Zhang, Y. Tian, et al., Asynchronous federated learning over wireless communication networks, IEEE Trans. Wireless Commun. (2022).
[25] C. Xie, S. Koyejo, I. Gupta, Asynchronous federated optimization, 2019, arXiv preprint arXiv:1903.03934.
[26] Y. Lu, X. Huang, Y. Dai, et al., Differentially private asynchronous federated learning for mobile edge computing in urban informatics, IEEE Trans. Ind. Inform. 16 (3) (2019) 2134–2143.
[27] B. Gu, A. Xu, Z. Huo, et al., Privacy-preserving asynchronous vertical federated learning algorithms for multiparty collaborative learning, IEEE Trans. Neural Netw. Learn. Syst. (2021).
[28] N. Skatchkovsky, H. Jang, O. Simeone, Federated neuromorphic learning of spiking neural networks for low-power edge intelligence, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2020, pp. 8524–8528.
[29] Y. Venkatesha, Y. Kim, L. Tassiulas, et al., Federated learning with spiking neural networks, IEEE Trans. Signal Process. 69 (2021) 6183–6194.
[30] K. Xie, Z. Zhang, B. Li, et al., Efficient federated learning with spike neural networks for traffic sign recognition, IEEE Trans. Veh. Technol. (2022).
[31] T. Li, A.K. Sahu, M. Zaheer, et al., Federated optimization in heterogeneous networks, Proc. Mach. Learn. Syst. 2 (2020) 429–450.
[32] Q. Li, B. He, D. Song, Model-contrastive federated learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 10713–10722.
[33] E.O. Neftci, H. Mostafa, F. Zenke, Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks, IEEE Signal Process. Mag. 36 (6) (2019) 51–63.
[34] Y. Kim, P. Panda, Revisiting batch normalization for training low-latency deep spiking neural networks from scratch, Front. Neurosci. (2020) 1638.
[35] Yuhao Zhou, Qing Ye, Jiancheng Lv, Communication-efficient federated learning with compensated overlap-fedavg, IEEE Trans. Parallel Distrib. Syst. 33 (1) (2021) 192–205.
[36] S. Kim, S. Park, B. Na, et al., Spiking-yolo: spiking neural network for energy-efficient object detection, Proc. AAAI Conf. Artif. Intell. 34 (07) (2020) 11270–11277.
[37] B. Zhao, R. Ding, S. Chen, et al., Feedforward categorization on AER motion events using cortex-like features in a spiking neural network, IEEE Trans. Neural Netw. Learn. Syst. 26 (9) (2014) 1963–1978.
[38] H. Yu, S. Yang, Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning, in: Proceedings of the AAAI Conference on Artificial Intelligence,Vol. 33, 2019, pp. 5693–5700.
[39] S.P. Karimireddy, M. Kale, S. Mohri, et al., Scaffold: Stochastic controlled averaging for federated learning, in: International Conference on Machine Learning, PMLR, 2020, pp. 5132–5143.
[40] T. Li, A.K. Sahu, M. Zaheer, et al., Federated optimization in heterogeneous networks, Proc. Mach. Learn. Syst. 2 (2020) 429–450.
[41] J. Nguyen, K. Malik, H. Zhan, et al., Federated learning with buffered asynchronous aggregation, 2021, arXiv preprint arXiv:2106.06639.
[42] M. Chen, B. Mao, T. Ma, Efficient and robust asynchronous federated learning with stragglers, in: Submitted To International Conference on Learning Representations, 2019.

Yuan Wang received the B.S. degree in Information and Computing Science from Changchun University of Technology in 2020. He is currently studying for a master's degree at the school of electronic and information engineering, Southwest University, Chongqing, China. His primary research interests include machine learning, federated learning, and intelligent information processing.

Shukai Duan is the Dean and Professor of the College of Artificial Intelligence, Southwest University, Chongqing, China. He is also the Director of National Local Joint Engineering Laboratory of Intelligent Transmission and Control Technology, the Director of the Key Laboratory of Brain-Like Computing and Intelligent Control in Chongqing, and the Deputy Director of Artificial Intelligence and Robotics Education Special Committee of China Automation Society. Prof. Duan received his Ph.D. degree in Computer Science in 2006 from Chongqing University, China. His research focuses on the circuit and systems, artificial neural networks, brain-like intelligence, etc. He has published four books, contributed to more than 300 research papers in Nature Communications, IEEE Transactions on Neural Networks and Learning Systems, IEEE Transactions on VLSI Systems, etc. He also serves as Associate Editor for IEEE Transactions on Neural Networks and Learning Systems, Artificial Intelligence Review, Neurocomputing.

Feng Chen received the Ph.D. degree from the University of Electronic Science and Technology of China in 2013. From 2008 to 2010, he visited in Prof. Emery Brown's research group with Massachusetts Institute of Technology, and also worked with Harvard University, Massachusetts General Hospital. Since 2020, he has been a Professor with the College of Artificial Intelligence, Southwest University, Chongqing, China. He has published over 50 articles in various journals and conference proceedings. His research interests include signal processing, distributed estimation, machine learning and wireless sensors networks.